Pontifícia Universidade Católica
do Rio de Janeiro

Daniel dos Santos Marques

A Decision Tree Learner for Cost-Sensitive Binary Classification

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–Graduação em Informática of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Eduardo Sany Laber

Rio de Janeiro
September 2016

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel dos Santos Marques

A Decision Tree Learner for Cost-Sensitive Binary Classification

Dissertation presented to the Programa de Pós–Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC–Rio, as partial fulfillment of the requirements for the degree of Mestre.

Prof. Eduardo Sany Laber
Advisor
Departamento de Informática — PUC–Rio

Prof. Ruy Luiz Milidiú
Departamento de Informática — PUC-Rio

Prof. Raúl Pierre Rentería
Departamento de Informática — PUC-Rio

Prof. Márcio da Silveira Carvalho
Coordinator of the Centro Técnico Científico — PUC–Rio

Rio de Janeiro, September 22nd, 2016

Daniel dos Santos Marques

Daniel dos Santos Marques graduated from Pontifícia Universidade Católica do Rio de Janeiro in Information Systems. As an undergraduate was a member of the ICPC team.

Ficha Catalográfica

Marques, Daniel dos Santos

A decision tree learner for cost-sensitive binary classification / Daniel dos Santos Marques; advisor: Eduardo Sany Laber. — 2016.

46 f. : il. (color.); 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Departamento de Informática, 2016.

Inclui bibliografia.

1. Informática – Teses. 2. Aprendizado de máquina. 3. Árvore de decisão. 4. Aprendizado sensível ao custo. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Abstract

Marques, Daniel dos Santos; Laber, Eduardo Sany (Advisor). **A Decision Tree Learner for Cost-Sensitive Binary Classification**. Rio de Janeiro, 2016. 46p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Classification problems have been widely studied in the machine learning literature, generating applications in several areas. However, in a number of scenarios, misclassification costs can vary substantially, which motivates the study of Cost-Sensitive Learning techniques. In the present work, we discuss the use of decision trees on the more general Example-Dependent Cost-Sensitive Problem (EDCSP), where misclassification costs vary with each example. One of the main advantages of decision trees is that they are easy to interpret, which is a highly desirable property in a number of applications. We propose a new attribute selection method for constructing decision trees for the EDCSP and discuss how it can be efficiently implemented. Finally, we compare our new method with two other decision tree algorithms recently proposed in the literature, in 3 publicly available datasets.

Keywords

Machine Learning; Decision Tree; Cost-Sensitive learning.

Resumo

Marques, Daniel dos Santos; Laber, Eduardo Sany. **Uma árvore de Decisão para Classificação Binária Sensível ao Custo**. Rio de Janeiro, 2016. 46p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Problemas de classificação foram amplamente estudados na literatura de aprendizado de máquina, gerando aplicações em diversas áreas. No entanto, em diversos cenários, custos por erro de classificação podem variar bastante, o que motiva o estudo de técnicas de classificação sensível ao custo. Nesse trabalho, discutimos o uso de árvores de decisão para o problema mais geral de Aprendizado Sensível ao Custo do Exemplo (ASCE), onde os custos dos erros de classificação variam com o exemplo. Uma das grandes vantagens das árvores de decisão é que são fáceis de interpretar, o que é uma propriedade altamente desejável em diversas aplicações. Propomos um novo método de seleção de atributos para construir árvores de decisão para o problema ASCE e discutimos como este pode ser implementado de forma eficiente. Por fim, comparamos o nosso método com dois outros algoritmos de árvore de decisão propostos recentemente na literatura, em 3 bases de dados públicas.

Palavras–chave

Aprendizado de máquina; Árvore de decisão; Aprendizado sensível ao custo.

Contents

1
Introduction

Classification problems have been widely studied in the machine learning literature, generating applications in several areas such as medical diagnosis, fraud detection and others.

However in a number of scenarios misclassification costs can vary substantially. Let's take the example of a bank which needs to decide whether or not to give a loan to a customer. Clearly lending $1,000,000 to a client which defaults his/her debt incurs a greater cost to the bank than a $1,000 default by another client. This generates a big imbalance on misclassification costs between clients which cost insensitive classification algorithms can't handle well.

In this dissertation we study the Example-Dependent Cost-Sensitive Problem (EDCSP), where misclassification costs vary with each example, for binary classification.

1.1
Problem Definition

Consider an universe U of samples where each sample is modeled by a set of attributes $A_1, \ldots, A_m$. Each attribute A_i can be thought as a function that maps each sample $s \in U$ into a value $A_i(s)$. In addition, we have two classes 0 and 1 and a cost function $C : U \times \{0, 1\} \mapsto R^+$ mapping each pair (s, c) into a real number that represents the cost of a sample s when it is assigned to class c.

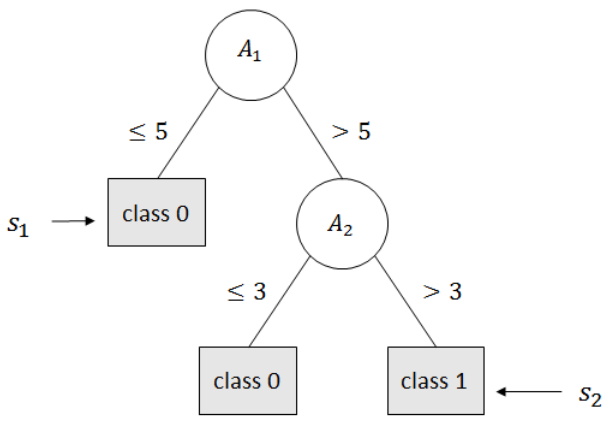
Our goal is to learn a function f that maps each sample into a class $c \in \{0, 1\}$ so that $\sum_{s \in U} C(s, f(s))$ is minimized. For that, we are given a subset S, randomly selected from U, and the values $C(s, 0)$ and $C(s, 1)$ for every $s \in S$.

Here, we restrict our attention to the case where f belongs to the class of decision tree functions. A decision tree D for a set of samples S is a rooted tree where each of its nodes is associated with an attribute and each of its leaves is associated with a class. Each edge e that leaves a node, associated with an attribute A, is associated with a subset of the values that A may assume. These subsets must be pairwise disjoint.

To classify a sample s using a decision tree D we follow a root to leaf path as follows: if the root of D is also a leaf we assign to s the class associated

with the leaf. Otherwise, if the root of D is associated with an attribute R, we recursively apply the procedure for the sub-tree rooted at the node we reach following the branch associated with $R(s)$.

In Figure 1.1 we show how we can classify two samples s_1 and s_2 in a decision tree using attributes A_1 and A_2.



	A_1	A_2
s_1	2	10
s_2	6	5

Figure 1.1: Decision Tree Example

One of the main motivations for using decision trees, in oppose to other methods, is that they provide a simple interpretation that is highly desirable in a number of applications.

1.2 Our Contributions

We propose a new decision tree method for the Example-Dependent Cost-Sensitive Problem (EDCSP) problem, denoted by PairTree.

Our method is based on a new metric to measure the quality of an attribute split in a tree node. This new metric considers each pair of examples and measures how much is gained by splitting them into separate nodes. We also take into account how a random attribute, one which randomly splits the dataset, performs on our metric. This way we can measure how good an attribute is compared to a random attribute. This helps to avoid overfit as we can stop growing the tree if all attributes perform no significantly better than a random split.

We discuss our method implementation aspects in order to make its computation feasible for larger datasets. We use the Binary Indexed Tree (Fenwick, 1994) data structure to calculate our metric for numerical attributes.

We make an experimental study on three publicly available datasets: Direct Marketing (Moro et al., 2014), Give Me Some Credit (Kaggle.com, 2011), and PAKDD2009 (PAKDD, 2009). Our method showed superior results

when compared to two other decision tree methods from the cost-sensitive learning literature proposed in (Aodha and Brostow, 2013) and (Bahnsen et al., 2015).

We compared our results using the *Savings* metric which shows how good a tree is when compared to the strategy of classifying all examples in one class.

$$Savings = 1 - \frac{\text{Tree Cost}}{\text{Cost of best class}}$$

Our PairTree method achieved the best results for both Give Me Some Credit (50.59%) and PAKDD2009 (29.82%) datasets, an improvement of 6% and 4% respectively when compared to the best benchmark. While for the Direct Marketing dataset our results were only 0.4% worse than the best benchmark.

1.3 Related work

Costs naturally occur in the classification task as it had already been noticed in Breiman's seminal book (Breiman et al., 1984). Two common sources of costs are the attributes and the misclassification. As an example, in a diagnosis procedure some blood exams may be much more expensive than others and, still in this context, classifying a patient as healthy when he is sick may be much more risky than the other direction. For a survey focused on cost-sensitive decision trees algorithm we recommend (Lomax and Vadera, 2013) in which the authors present over 50 algorithms.

Here, we focus on misclassification costs. This type of cost can be naturally introduced in 2 different ways: In the most general version, the Example-Dependent Cost-Sensitive Problem (EDCSP), the cost of classifying a sample depends on the sample and the predicted class. While in the restricted version, Class-Dependent Cost-Sensitive Problem (CDCSP), the cost depends on the true class and the predicted class. The latter is a particular case of the former as it can be seen as an EDCSP where the misclassification costs are the same for all examples. Here, however, we focus on the EDCSP.

In (Zadrozny et al., 2003), Zadrozny et al. presents techniques to address EDCSP that are based on a central theorem that shows how to change the sample distribution so that traditional classifiers 'become' cost-sensitive. These techniques differ mainly on how the samples are drawn from the new distribution. An experimental study with these methods is carried on the KDD98 and DMEF-2 datasets.

In (Aodha and Brostow, 2013), a Random Forest for the EDCSP is proposed. To build the decision trees that are used for the committee they

propose a new impurity measure that takes into account, for each example s, the difference between the cost of classifying s as i and j for each pair of classes i and j. They compare their method with two baselines: the Gini-based Cost-Sensitive Random Forest (GCSRF) and the naive Classification Random Forest (CLRF). They report improved results for 3 multi-class computer vision datasets at a price of a modest increase in the training time.

In (Bahnsen et al., 2015), Bahnsen et. al. define a simple cost-based impurity measure for building decision trees for the example dependent cost sensitive problem. The method is compared with some state-of-the-art methods in over 3 datasets. They report that their method outperforms all the others, including decision trees, logistic regression and random forests trained with the techniques proposed by Zadrozny (Zadrozny et al., 2003) and Elkan (Elkan, 2001). It must be mentioned that the metric presented in (Bahnsen et al., 2015) had already been mentioned/proposed in other papers (Pazzani et al., 1994), (Turney, 1995), (Ling et al., 2004) and (Greiner et al., 2002) in the context of the CCDCP.

1.4 Dissertation's Organization

In Chapter 2 we discuss some basic concepts necessary to the understanding of our decision tree method which is presented on Chapter 3.

The experimental study is described at Chapter 4 in which we execute our method on 3 publicly available datasets, and compare it with two other decision tree benchmarks from the cost-sensitive learning literature.

Chapter 5 holds some final remarks and a conclusion to this work.

2
Basic Concepts

In this chapter we discuss some concepts that are important to understand our work.

We start by introducing a general framework of decision tree construction, which will be used later when we discuss our method. We then discuss how to bound the sum of dependent variables using the edge coloring concept from graph theory.

2.1
Decision Tree Construction

As described by (Murthy, 1998) there are different ways to construct a decision tree, but the most commonly used one is the greedy top-down approach.

On greedy top-down methods we start with the entire dataset and split it into two or more disjoint subsets, according to some predefined rule. We then recursively call the method on the generated subsets. We can also use a stopping rule, to decide whether or not we should continue growing the tree.

We provide a pseud-code below.

Algorithm 1 Greedy Top-Down Tree Creation

```
1: procedure CreateTree(set of samples S)
2:     if S does not meet the stopping conditions then
3:         Find a way to split S into two or more subsets
4:         Recursively call CreateTree on the generated subsets of S
5:     else
6:         Stop growing the tree.
```

It's worth noticing that this approach can be used both for cost-sensitive and cost-insensitive classification problems.

2.2
Edge Coloring for Graphs

Let $G = (V, E)$ be a graph. A k-edge coloring for G is a function that maps each edge into an integer (color) in the set $\{1, \ldots, k\}$ such that edges that share an endpoint are mapped into different colors.

The chromatic index of a graph G is the minimum integer k^* for which G admits a k^*-coloring. It is known (Vizing., 1964) that the chromatic number

of a graph is either $\Delta(G)$ or $\Delta(G)+1$, where $\Delta(G)$ is the maximum degree of the vertices in G. It is also known that for bipartite graphs the chromatic index is $\Delta(G)$.

2.3 Bounding the Sum of Dependent Variables

Let $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_k\}$ be a set of real valued random variables, where Y_i takes values on the interval $[a_i, b_i]$ for $i = 1, \ldots, k$. A family $\mathcal{F}$ of subsets of $\mathcal{Y}$ is a cover for $\mathcal{Y}$ if the union of all subsets of $\mathcal{F}$ is equal to $\mathcal{Y}$. A subset $\mathcal{Z} \subseteq \mathcal{Y}$ is *independent* if the variables in $\mathcal{Z}$ are independent. A cover $\mathcal{F}$ for $\mathcal{Y}$ is *proper* if all subsets of $\mathcal{F}$ are independent. Let $\chi(\mathcal{Y})$ be the size of the proper cover for $\mathcal{Y}$ with minimum cardinality.

The following theorem was proved by Janson in (Janson, 2004).

Theorem 1. *(Janson, 2004) Let $X = \sum_{i=1}^{k} Y_i$ and let $t > 0$. Then,*

$$P(X \geq E[X] + t) \leq \exp\left(\frac{-2t^2}{\chi(\mathcal{Y}) \sum_{i=1}^{k} (b_i - a_i)^2}\right)$$

For our study, the case where the variables Y_i's can be seen as edges of a complete bipartite graph G and a set of variables is independent if and only if the underlying edges correspond to a matching in G is of particular interest. In this case, every edge coloring for G corresponds to a proper cover for $\mathcal{Y}$ and the size of the minimum proper cover for the set of variables is at most the chromatic index of G, which is equal to $\Delta(G)$. Hence, we can state the following corollary of Theorem 1

Corollary 1. *Let $G = (V, E)$ be a bipartite graph where $E = \{e_1, \ldots, e_m\}$ and the maximum degree of G is $\Delta(G)$. In addition, let $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$ be set of random variables that satisfies the following property: for all $S \subseteq \{1, \ldots, m\}$, the set of variables $\{Y_i | i \in S|\}$ is independent if and only if $\{e_i | i \in S|\}$ is a matching in G.*

Let $X = \sum_{i=1}^{m} Y_i$ and let $t > 0$. Then,

$$P(X \geq E[X] + t) \leq \exp\left(\frac{-2t^2}{\Delta(G) \sum_{i=1}^{k} (b_i - a_i)^2}\right),$$

3
The PairTree Method

In this chapter, we provide a description of our decision tree method for the two class Example-Dependent Cost-Sensitive Classification problem. The following notation is helpful for our discussion: for a set of samples S and a class κ we define

$$C(S, \kappa) = \sum_{x \in S} C(x, \kappa)$$

. In addition, we define

$$S_0 = \{x | x \in S \text{ and } C(x, 0) \leq C(x, 1)\}$$

and

$$S_1 = \{x | x \in S \text{ and } C(x, 0) > C(x, 1)\}$$

One of the key issues for designing decision tree learners is to define how an attribute A splits the set of samples S associated with a node of the tree. Our method defines partitions for nominal and numeric attributes in different ways.

Let A be a nominal attribute that takes values on the domain $\{v_1, \ldots, v_k\}$ for the samples of set S. In our method, A splits the samples of S into k groups $\{G_1, \ldots, G_k\}$, where G_i contains all samples s for which $A(s) = v_i$. On the other hand a numeric attribute with values in the range $[a, b]$, splits S into two groups (G_1^ℓ, G_2^ℓ), where G_1^ℓ contains all samples with values smaller than or equal to ℓ and G_2^ℓ contains all samples with values larger than ℓ. The value ℓ is chosen so as to generate a partition with best possible quality according to the metric proposed in the next section.

Given a training set S of N samples $\{s_1, s_2, \ldots, s_N\}$, our method, outlined in Algorithm 2, follows a top-down approach. First, it verifies whether the set S meets a stopping criterion that is also discussed further. In the positive case, we create a leaf and associate it with class 0 if $C(S, 0) < C(S, 1)$ and with class 1, otherwise. In the negative case, the procedure selects the attribute which provides the split with best quality for the set S, where the quality of a split is measured by a new cost-sensitive metric explained in the next section. This attribute, say A, is set as the root r of the decision tree. Next, for each group

in the partition of S, induced by A, we recursively construct a decision tree and set it as a child of r.

Algorithm 2 PairTree creation

```
1: procedure CreateTree(set of samples S)
2:     if S does not meet the stopping conditions then
3:         Select attribute A with best quality
4:         Recursively call CreateTree on all groups generated by A
5:     else
6:         Create a leaf in the current node and associate it with the class that
   provides the minimum cost for set S.
```

3.1 Evaluating the quality of attributes

In order to explain the quality of an attribute w.r.t. a set of samples S, we first define the gain $G(x, y)$ of separating two samples x and y as

$$\begin{aligned} G(x,y) = \min\{C(x,0) + C(y,0), C(x,1) + C(y,1)\} \\ -(\min\{C(x,0), C(x,1)\} + \min\{C(y,0), C(y,1)\}) \end{aligned} \tag{3.1}$$

This measure captures the benefit of separating x and y by considering the minimum possible cost when they are in the same group, which is $\min\{C(x,0) + C(y,0), C(x,1) + C(y,1)\}$ and the minimum possible cost when both are in different groups, which is $\min\{C(x,0), C(x,1)\} + \min\{C(y,0), C(y,1)\}$). Note that if the class of minimum cost for x and y is the same then $G(x, y) = 0$ because there is no gain in separating them. This generalizes the metric proposed in (Golovin et al., 2010; Cicalese et al., 2014).

The gain $Gain(A, S)$ of a nominal attribute A w.r.t. a set of samples S, is defined as the sum of $G(x, y)$ for all pairs of samples in S that are separated by A, that is, samples that belong to different groups in the partition of S induced by A:

$$Gain(A,S) = \sum_{x,y \in S | A(x) \neq A(y)} G(x,y) \tag{3.2}$$

For the sake of a simpler notation, whenever the context is clear we drop S from $Gain(A, S)$.

To motivate our metric we discuss how it compares with the impurity metric proposed in (Bahnsen et al., 2015). Let $C(S, x)$ be the cost of classifying all samples of S as x. The gain of a binary attribute X that splits S into groups A and B is given by

$$\min\{C(S,0),C(S,1)\} - \min\{C(A,0),C(A,1)\} - \min\{C(B,0),C(B,1)\} \quad (3.3)$$

One potential advantage of our metric with respect to this one is that it is more sensible to capture the progress made by an attribute for the classification task.

As an example, consider a set of 1000 samples formed by two groups, say X and Y, where X has 100 samples and Y has 900. The costs are either 0 or 1 where for all samples $x \in X$, $C(x,0) = 0$ and $C(x,1) = 1$ while for all $y \in Y$ we have $C(y,0) = 1$ and $C(y,1) = 0$.

Let us consider an attribute A that splits the samples in $X \cup Y$ into two groups $X \cup Y'$ and $Y - Y'$, where Y' is a subset of Y containing 150 samples, according to the figure below.

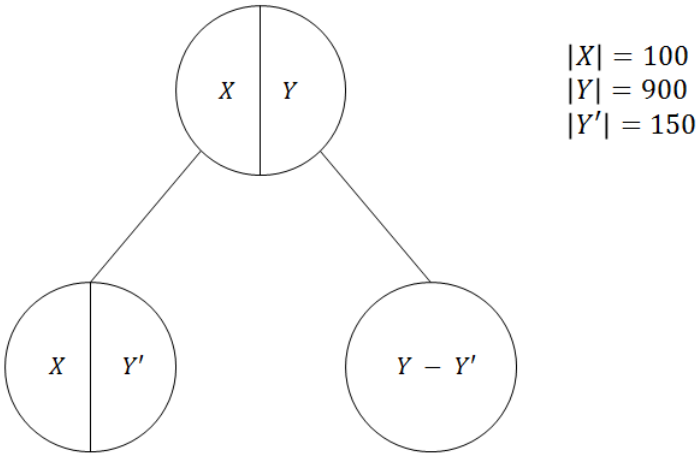


Figure 3.1: Metric Example

Attribute A does a very good job in separating samples from X and Y. However, since the minimum cost of both children and parent is achieved when using class 1, the metric of equation (3.3) is equal to 0 indicating there is no gain in using A.

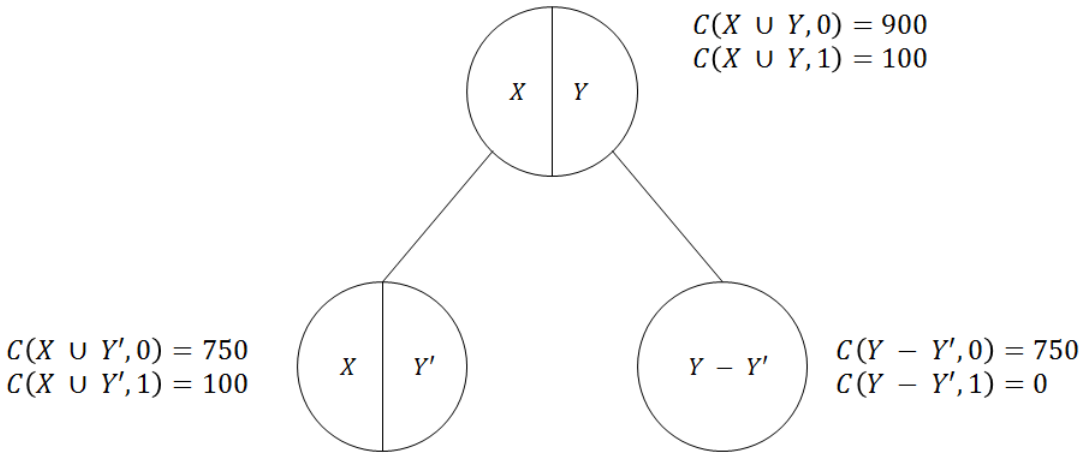


Figure 3.2: Metric Costs Example

For this example the metric proposed here can be written as the sum of $G(x, y)$ for the separated samples:

$$\sum_{\substack{x \in X \\ y \in Y - Y'}} G(x, y) = \sum_{\substack{x \in X \\ y \in Y - Y'}} 1 = 750 \times 100 = 75000$$

While the best possible attribute would've achieved a gain of $900 \cdot 100 = 90000$.

3.2 Adjusting the quality of attributes

One problem with our metric is that it tends to favor attributes that may assume a large number of distinct values. This effect has been reported in the literature and there are some methods available to cope with this problem.

To illustrate this issue let's consider a database of 3650 people, with 1825 men and 1825 women, that were born in a given year. Each person is modeled by two attributes: `IsTall` and `DayofBirth`. The attribute `IsTall` is TRUE if the person has height larger than 1.80m and FALSE, otherwise. The attribute `DayofBirth` is a number in the set $\{1, \ldots, 365\}$.

Our task is to classify people according to gender: male/female. Which attribute provides the best split?

The misclassification costs are either 0 or 1, where we have $C(x, 0) = 0$ and $C(x, 1) = 1$ for a all men x and for all women y follows $C(y, 0) = 1$ and $C(y, 1) = 0$. Table 3.1 illustrates an example of this dataset.

Index	DayOfBirth	IsTall	$\cdots$	$C(s,0)$	$C(s,1)$
1	64	TRUE	$\cdots$	0	1
2	105	FALSE	$\cdots$	1	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
3650	342	FALSE	$\cdots$	0	1

Table 3.1: Example Dataset

Clearly, we expect the attribute `IsTall` to be more effective than `DayofBirth`. Let's assume there is no correlation between the day of birth and gender, and we observe exactly 5 men and 5 women with the same birth date for every day of the year. So attribute `DayofBirth` splits the 3650 people on the dataset according to figure 3.3

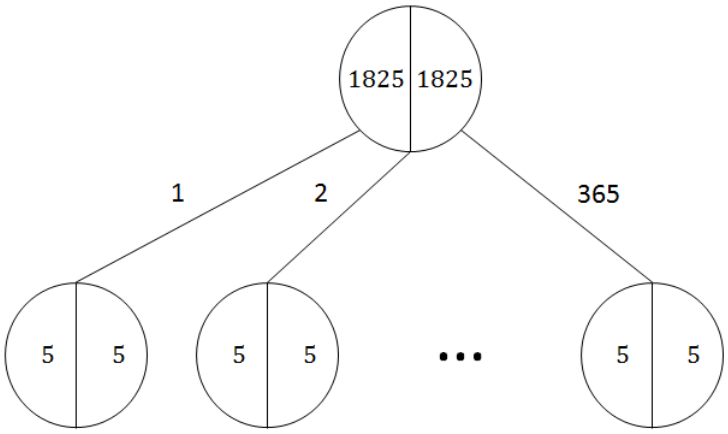


Figure 3.3: Attribute `DayofBirth`

To calculate our *Gain* metric for the `DayofBirth` attribute we need to count how many pairs of men and women were separated after the split. By Figure 3.3 we can see a man is separated from all women except for the 5 in his node (1820 in total). Since we have 1825 men in the datset we can write our *Gain* as:

$$Gain(DayofBirth) = 1820 \times 1825 = 3321500$$

To calculate the *Gain* for attribute `IsTall` let's assume 30% (1%) of men (women) are taller than 1.80m. In our dataset this gives a total of 548 tall men, and 2 tall women. Figure 3.4 shows a split formed by attribute `IsTall`.

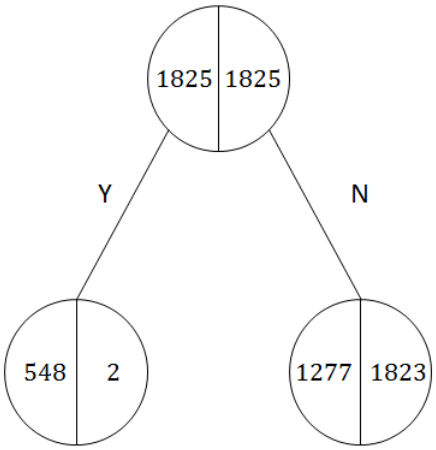


Figure 3.4: Attribute `IsTall`

We can calculate the *Gain* of `IsTall` using a similar approach as we used for `DayofBirth`, which gives us:

$$Gain(IsTall) = 548 \times 1823 + 2 \times 1277 = 1001558$$

Thus, according to our metric `DayofBirth` is better than `IsTall`.

To prevent the bias towards attributes with large number of values we adjust the *Gain* of the partition $\mathcal{P}(S, A)$ induced by an attribute A on the set of samples S by comparing it with the *Gain* of a 'random partition' R that has properties similar to $\mathcal{P}(S, A)$.

We say partition $\mathcal{P}(A, S)$ is a split, on S, formed by k disjoint groups G_i. Where a group G_i is the set of all samples in S that assume the k-th value in attribute A.

Let R be a partition of the set of samples S generated according to the following process:

`For` i=1,...,$|S|$

Assign sample i to group j with probability $|G_j|/|S|$

`End For`

Informally, A is a good split if the probability of a random partition R, obtained by the above process, generates a gain larger than that of A is small. Let

$$\alpha(S, A) = Pr[Gain(R) \geq Gain(A)] \tag{3.4}$$

where R is sampled according to our process.

We would like to select the attribute A such that $\alpha(S, A)$ is minimum. A problem with this approach is that is not clear how to calculate $\alpha(S, A)$ efficiently.

As an example, one approach for estimating $\alpha(S, A)$ might consist of randomly generating several partitions, according to our process, and counting how many had a *Gain* greater than or equal to $Gain(A)$. This is naturally a very slow solution, as it requires a great number of random draws.

Our approach to this problem makes use of Corollary 1 of Theorem 1. We need some definitions.

For a partition R, generated by our process, and two samples x and y, we define the variable

$$Y_{x,y} = \begin{cases} G(x,y), & \text{If x and y belongs to different groups in } R \\ 0, & \text{Otherwise} \end{cases}$$

By using the fact that $G(x, y) = 0$ if either $x, y \in S_0$ or $x, y \in S_1$ we get that

$$Gain(R) = \sum_{x,y \in S} Y_{x,y},$$

Thus,

$$\alpha(S, A) = Pr[Gain(R) \geq Gain(A)] = \tag{3.5}$$

$$Pr[\sum_{x,y} Y_{x,y} \geq Gain(A)] \tag{3.6}$$

The difficulty of bounding $Pr[\sum_{x,y} Y_{x,y} \geq Gain(A)]$, and as a consequence $\alpha(S, A)$, relies on the fact that the variables $Y_{x,y}$ are not independent so that Chernoff/Hoeffding's bounds cannot be used. However, Theorem 1 allows to deal with the sum of dependent variables.

To understand the dependence between our variables $Y_{x,y}$'s it is useful to identify the variables $Y_{x,y}$'s as edges of a complete bipartite graph G with bi-partition (S_0, S_1). It is not difficult to realize that a set of variables is independent if and only if the underlying edges correspond to a matching in the graph. Thus, it follows from Corollary 1 that

$$\alpha(S, A) = Pr[Gain(R) \geq E[Gain(R)] + t] \leq \exp\left(\frac{-2t^2}{\max\{|S_0|, |S_1|\} \sum_{x,y \in S} G(x,y)^2}\right)$$

where $t = Gain(A) - E[Gain(R)]$

Let

$$\hat{\alpha}(S, A) = \exp\left(\frac{-2t^2}{\max\{|S_0|, |S_1|\} \sum_{x,y \in S} G(x,y)^2}\right)$$

be the right-hand side of the equation above.

Instead of selecting the attribute A with minimum $\alpha(S, A)$, our method selects the attribute A with minimum $\hat{\alpha}(S, A)$.

3.3 Stopping Conditions

Our method uses two parameters `MaxProb` and `MinSamples` to determine whether the decision tree construction is pruned at a given node ν. In fact, the tree is pruned if one of the following conditions occurs:

i There is no attribute A with $\hat{\alpha}(S, A) \leq$ `MaxProb`;

ii The size of a set of samples associated with ν is smaller than `MinSamples`

iii All samples of S are in S_0 (or S_1)

Condition (i) requires that there is an attribute that produces a rare gain in the sense that it should be much better than that of a random partition. This condition could be used alone to prune the tree because it is not reasonable to keep growing the tree if all attributes produce 'random' gains.

The reason why we also use condition (ii), however, is because we do not have a sharp estimation for $\alpha(S, A)$. To understand this issue, assume that we set `MaxProb` $= 0.1$. Because the bound is not sharp it may happen that we are just accepting attributes whose gains are extremely rare, that is, the probability of occurring them is much smaller than 0.1. This way we are being too restrictive.

One attempt to handle this situation is using large values for `MaxProb`. However, this is also risky because we could be accepting attributes that behave as random partitions, which could lead to data overfitting. Using a lower bound on the number of samples reduces the risk of overfitting the data when we are working with large values for `MaxProb`.

If condition (iii) is met then it's useless to make any further splits as all the samples are best classified in the same class. Therefore node v should be a leaf.

At this point our method is completely specified as shows Algorithm 3. In the sequel we discuss how to implement it in an efficient way.

condition (iii) is not met

Algorithm 3 PairTree creation

1: **procedure** CREATETREE(set of samples S, parameter `MaxProb`, parameter `MinSamples`)
2: Select attribute A such that $\hat{\alpha}(S, A)$ is minimum
3: **if** $\hat{\alpha}(S, A) \leq$ `MaxProb` and $|S| \geq$`MinSamples` and $S \cap S_0 \neq \emptyset$ and $S \cap S_1 \neq \emptyset$ **then**
4: Recursively call CreateTree on all splits generated by A
5: **else**
6: Create a leaf in the current node and associate it with the class that provides the minimum cost for set S.

3.4 Implementation aspects

For an efficient implementation of our decision tree construction we need to evaluate $\hat{\alpha}(S, A)$ quickly. For that we need to calculate $Gain(A)$, $\sum_{x,y \in S} G(x, y)^2$ and $E[Gain(R)]$.

First, we consider the computation of $E[Gain(R)]$. Recall that A induces the partition $\mathcal{P}(A, S) = (G_1, \ldots, G_k)$ when applied to set of samples S. Let $p_i = |G_i|/(|G_1| + \cdots + |G_k|)$. We have that

$$E[Gain(R)] = E\left[\sum_{x,y \in S} G(x, y)\right] = \sum_{x,y \in S} E[G(x, y)] =$$

$$\sum_{x,y \in S} \sum_{i=1}^{k} G(x, y) \cdot p_i \cdot (1 - p_i) = \left(\sum_{x,y \in S} G(x, y)\right)\left(\sum_{i=1}^{k} p_i \cdot (1 - p_i)\right), \quad (3.7)$$

where the product $p_i(1 - p_i)$ is the probability that in a random partition sample x belongs to i-th group and sample y does not belong it.

To evaluate $E[Gain(R)]$ we need to compute $\sum_{x,y \in S} G(x, y)$, which can be seen as the gain of a 'perfect' attribute, say P, that splits the set S into S_0 and S_1. Thus, if we compute $Gain(P)$ efficiently we manage to compute $E[Gain(R)]$ efficiently as well.

In the sequel we discuss how to efficiently compute the gain of any attribute, in particular the attribute P, and how to compute $\sum_{x,y \in S} G(x, y)^2$. The latter calculation is pretty straightforward given that we know how to compute $Gain(P)$. We first consider nominal attributes and then numeric attributes.

3.4.1 Nominal Attributes

We wish to calculate the *Gain* of a nominal k-valued attribute A for a set of N samples S in an efficient manner. There is a direct algorithm which consists of trying all pairs of samples taking $O(N^2)$ time. We wish to speed up this process as a time complexity of $O(N^2)$ can be prohibitive for some large datasets.

We rewrite the *Gain* of an attribute A in a way that will make the computation faster. First, we define the absolute difference $D(s)$ of a sample $s \in S$ as

$$D(s) = |C(s,0) - C(s,1)| \tag{3.8}$$

For simplicity whenever we refer to a sample $x(y)$ we're dealing with a sample from the set $S_0(S_1)$. The following propositions turns out to be useful.

Proposition 1. $G(x,y) = \min\{D(x), D(y)\}$

Proposition 2.

$$\begin{aligned} Gain(A) = &\sum_{x \in S_0} D(x) \cdot |\{y | A(x) \neq A(y) \wedge D(x) \leq D(y)\}| \\ &+ \sum_{y \in S_1} D(y) \cdot |\{x | A(x) \neq A(y) \wedge D(x) > D(y)\}| \end{aligned} \tag{3.9}$$

We develop a procedure to calculate the quantities $|\{y|A(x) \neq A(y) \wedge D(x) \leq D(y)\}|$ for every $x \in S_0$ and $|\{x|A(x) \neq A(y) \wedge D(x) > D(y)\}|$ for every $y \in S_1$, described in Proposition 2, in linear time after an $O(n \log n)$ time preprocessing.

For that we create an array $DIFF$ containing all samples s ordered in ascending order by $D(s)$. If two samples x and y have the same $D(x)$ and $D(y)$ we put x first in the order.

Using vector $DIFF$ we define two new variables:

$TC_0^i =$ number of samples in S_0 in the subarray $DIFF[i:N]$
$TC_1^i =$ number of samples in S_1 in the subarray $DIFF[i:N]$

We note that, for $c = 0$ and $c = 1$,

$$TC_c^i = \begin{cases} TC_c^{i-1} - 1 & \text{if sample } s \text{ in position } i-1 \text{ of vector } DIFF \text{ is in } S_c \\ TC_c^{i-1} & \text{Otherwise} \end{cases}$$

In addition, we define the set of variables

$TCV^i_{c,j} =$ number of samples in S_c valued j in attribute A in the subarray $DIFF[i:N]$

We note that

$$TCV^i_{c,j} = \begin{cases} TCV^{i-1}_{c,j} - 1 & \text{if sample } s \text{ in position } i-1 \text{ of vector } DIFF \text{ is in } S_c \text{ and } A(s) = j \\ TCV^{i-1}_{c,j} & \text{Otherwise} \end{cases}$$

If $x \in S_0$ corresponds to the i-th element of $DIFF$ we have that

$$|\{y|A(x) \neq A(y) \wedge D(x) \leq D(y)\}| = TC^{i+1}_1 - TCV^{i+1}_{1,A(x)}.$$

Similarly, if $y \in S_1$ corresponds to the i-th element of $DIFF$ we have that

$$|\{x|A(x) \neq A(y) \wedge D(y) > D(x)\}| = TC^{i+1}_0 - TCV^{i+1}_{0,A(y)}$$

By using the expressions above and Proposition 2, we can rewrite $Gain(A)$ as:

$$Gain(A) = \sum_{i=0}^{N} D(s_i)(TC^i_c - TCV^i_{c,A(s_i)}), \tag{3.10}$$

where s_i is the sample in position i of vector $DIFF$ and $s_i \in S_c$.

We can now present the following algorithm to calculate $Gain(A)$:

Algorithm 4

```
1: function NominalGain(samples S, attribute A)
2:     Sort vector DIFF
3:     gain ← 0
4:     for i = 1 to N do
5:         Let s be the sample in position i of DIFF
6:         Let c such that s_i ∈ S_c
7:         Let c̄ such that s_i ∉ S_c̄
8:         gain ← gain + D(s) × (TC^i_c̄ − TCV^i_{c̄,A(s)})
9:         Update TC and TCV
       return gain
```

In the algorithm we can sort vector $DIFF$ in $O(N \log N)$ time so that the overall time complexity is $O(N \log N)$, which is a clear improvement over the $O(N^2)$ time spent by a brute force solution.

We can make one additional speed up by noting that the $O(N \log N)$ sorting of vector $DIFF$ only needs to be made once for the whole tree construction. We can create the $DIFF$ vector on a given non-root node

X in $O(N)$ time if we use the sorted $DIFF$ vector for the parent of X. The time complexity to analyze all nominal attributes in the tree becomes $O(N \cdot \log N + Depth(T) \cdot K \cdot N)$, where K is the number of nominal attributes in the tree and $Depth(T)$ is the depth of the tree T.

Finally, we note that $\sum_{x,y \in S} G(x,y)^2$ can be also calculated in $O(N \log N)$ time by running a slight variation of Procedure 4 in which $D(s)$ is replaced with $D(s)^2$ in line 8. This variation must be executed for an attribute P that splits S into S_0 and S_1.

3.4.2 Numeric Attribute

Numeric attributes present an additional difficulty as they can generate a large number of different partitions. Let's say we have a numeric attribute A ranging in the interval $[a, b]$ for the set of N samples S. If all samples have distinct values for attribute A then we have N possible choices for the splitting parameter ℓ. Every choice of ℓ gives us a different binary partition $\mathcal{P}(\ell)$ formed by groups $\{Left(\ell), Right(\ell)\}$ with an associated $Gain$.

If we try all possible values for the splitting parameter ℓ and use the method proposed on the last subsection for each of the induced partitions, we end up with an $O(N^2)$ time complexity solution.

We propose an alternative solution to calculate the gain of a numeric attribute, taking $O(N \log N)$ time.

As we mentioned before, there are at most N possible choices for the splitting parameter ℓ. For simplicity whenever we say ℓ on this section we are referring to one of the possible values $\ell_1 < \ell_2 < \cdots < \ell_N$ that the attribute under consideration may assume.

Let's pick one possible splitting parameter ℓ_i $(i = 1 \ldots N)$ and take a deeper look on how the samples are organized. The parameter ℓ_i generates a binary split where all samples valued less or equal to ℓ_i go to the left child (group $Left^i$) and the others to the right child (group $Right^i$).

A sample in the left (right) child can be from either S_0 or S_1. A sample s in the parent node appears on exactly one of the following groups:

- $Left_0^i$: Set of all samples s such that $s \in S_0$ and $A(s) \leq \ell_i$.
- $Left_1^i$: Set of all samples s such that $s \in S_1$ and $A(s) \leq \ell_i$.
- $Right_0^i$: Set of all samples s such that $s \in S_0$ and $A(s) > \ell_i$.
- $Right_1^i$: Set of all samples s such that $s \in S_1$ and $A(s) > \ell_i$.

For a fast implementation we wish to efficiently compute $Gain(A, \ell_{i+1})$ for $Gain(A, \ell_i)$. To construct $\{Left^{i+1}, Right^{i+1}\}$ from $\{Left^i, Right^i\}$ we need to move all samples s where $A(s) = \ell_{i+1}$ from $Right^i$ to $Left^{i+1}$. Hence,

$$\begin{aligned} Left^{i+1} &= Left^i \cup M \\ Right^{i+1} &= Right^i - M, \end{aligned} \tag{3.11}$$

where $M = \{s | s \in Right_i \wedge A(s) = \ell_{i+1}\}$.

We wish to examine how moving a sample $s \in M$ from $Right^i$ to $Left^{i+1}$ changes the gain of the attribute.

The gain $Gain(A, \ell_i)$ of numeric attribute A when using the splitting parameter ℓ_i is the sum of all $G(x, y)$ such that x and y are separated in partition $\mathcal{P}(\ell_i)$. We have:

$$Gain(A, \ell_i) = \sum_{x \in Left_0^i \wedge y \in Right_1^i} G(x, y) + \sum_{x \in Right_0^i \wedge y \in Left_1^i} G(x, y) \tag{3.12}$$

For the sake of a simpler presentation, we assume that s is best classified as class 0, that is, $s \in S_0$. In addition, we also assume that there is only one sample s in set M. We discuss how to drop this assumption by the end of this section.

In order to write $Gain(A, \ell_{i+1})$ from $Gain(A, \ell_i)$ we need to remove all contributions $G(y, s)$ from samples $y \in Left_1^i$ and add all contributions $G(s, y)$ from samples $y \in Right_1^i$. More formally, we can write:

$$Gain(A, \ell_{i+1}) = Gain(A, \ell_i) - \sum_{y \in Left_1^i} G(y, s) + \sum_{y \in Right_1^i} G(s, y) \tag{3.13}$$

This gives us the following pseudo-code to calculate the gain of attribute A for all ℓ_i.

Algorithm 5

```
1: function NumericGain(samples S, attribute A)
2:     Create sorted values ℓ_1 < ℓ_2 < ··· < ℓ_N
3:     gain ← Vector of size N + 1
4:     gain[0] ← 0
5:     for i = 1 to N do
6:         Let s be the sample such that A(s) = ℓ_i
7:         removed ← CalcRemoved(s)
8:         added ← CalcAdded(s)
9:         gain[i] ← gain[i − 1] − removed + added
       return max{gain[1 ... N]}
```

The time complexity of the above procedure heavily relies on how fast we can calculate the sums of removed and added contributions, represented by functions *CalcRemoved* and *CalcAdded*, respectively, on the pseudo-code above.

So let's examine closely the term $\sum_{y \in Left_1^i} G(s,y)$, representing the removed contributions on 3.13. At this point the reader should recall the definition of $D(s)$ given by equation 3.8. We can rewrite the sum as follows:

$$\sum_{y \in Left_1^i} G(s,y) = \sum_{y \in Left_1^i \wedge D(y) \leq D(s)} G(s,y) + \sum_{y \in Left_1^i \wedge D(y) > D(s)} G(s,y) \quad (3.14)$$

Let s_j be the sample at position j of $DIFF$. The following vectors will help us calculate the above sums:

$$DLeft_1[j] = \begin{cases} D(s_j) & \text{if } s_j \in Left_1^i \\ 0 & \text{Otherwise} \end{cases}$$

$$IndLeft_1[j] = \begin{cases} 1 & \text{if } s_j \in Left_1^i \\ 0 & \text{Otherwise} \end{cases}$$

Using the vectors above we can rewrite 3.14 as:

$$\sum_{y \in Left_1^i} G(s,y) = \sum_{j=1}^{pos-1} DLeft_1[j] + D(s) \cdot \sum_{j=pos+1}^{N} IndLeft_1[j], \quad (3.15)$$

where *pos* is the index of sample s in vector $DIFF$.

Let `GetSum`(V,i,j) be a a function that receives an array V of real numbers and two positions i and j and returns $\sum_{k=i}^{j} V[k]$. The following pseudo-code calculates the removed contributions:

Algorithm 6

1: **function** CALCREMOVED(sample s)
2: Let $\bar{c}$ such that $s \notin S_{\bar{c}}$
3: Let pos the index of sample s in vector $DIFF$
4: **return** `GetSum`($DLeft_{\bar{c}}$, 1, pos-1)+$D(s)\cdot$ `GetSum`($IndLeft_{\bar{c}}$, pos+1, N)

Note that the time complexity of *CalcRemoved* function depends on how fast we evaluate `GetSum` function.

Now we'll use a similar idea to write *CalcAdded* as a function of `GetSum`. We expand the added contributions term $\sum_{y \in Right_1^i} G(s,y)$, rewriting it as follows:

$$\sum_{y \in Right_1^i} G(s,y) = \sum_{y \in Right_1^i \wedge D(y) \leq D(s)} G(s,y) + \sum_{y \in Right_1^i \wedge D(y) > D(s)} G(s,y) \quad (3.16)$$

Again, we create 2 vectors as follows:

$$DRight_1[j] = \begin{cases} D(s_j) & \text{if } s_j \in Right_1^i \\ 0 & \text{Otherwise} \end{cases}$$

$$IndRight_1[j] = \begin{cases} 1 & \text{if } s_j \in Right_1^i \\ 0 & \text{Otherwise} \end{cases}$$

We then have

$$\sum_{y \in Right_1^i} G(s,y) = D(s) \cdot \sum_{j=pos+1}^{N} IndRight_1[j] + \sum_{j=1}^{pos-1} DRight_1[j], \quad (3.17)$$

where pos is the index of sample s in vector $DIFF$.

Finally, in pseudo-code:

Algorithm 7

1: **function** CALCADDED(sample s)
2: Let $\bar{c}$ such that $s \notin S_{\bar{c}}$
3: Let pos the index of sample s in vector $DIFF$
4: **return** $D(s)\cdot$ `GetSum`($IndRight_{\bar{c}}$, pos+1, N) + `GetSum`($DRight_{\bar{c}}$, 1, pos-1)

Again, the efficiency of *CalcAdded* relies on how fast we evaluate `GetSum` function.

Before proceeding the discussion, we recall that we assumed $s \in S_0$. The case $s \in S_1$ can be addressed in a similar way and it requires the usage of

vectors $DLeft_0, IndLeft_0, DRight_0, IndRight_0$ that are defined analogously to $DLeft_1, IndLeft_1, DRight_1, IndRight_1$

Back to our discussion, right after calculating $Gain(A, \ell_{l+1})$, we have to update the vectors $DLeft_0, IndLeft_0, DRight_0, IndRight_0$ as follows: $DLeft_0[pos] \leftarrow D(s_j)$, $IndLeft_0[pos] \leftarrow 1$, $DRight_0[pos] \leftarrow 0$ and $IndRight_0[pos] \leftarrow 0$.

The vectors $DLeft_1, IndLeft_1, DRight_1, IndRight_1$ are not updated because by assumption $s \in S_0$ so that they are not affected.

Algorithm 8 presents the full pseudo-code for calculating the gain of a numeric attribute.

Algorithm 8

```
1: function NumericGain(samples S, attribute A)
2:     Create sorted values ℓ_1 < ℓ_2 < ··· < ℓ_N
3:     gain ← Vector of size N + 1
4:     gain[0] ← 0
5:     for i = 0 to N − 1 do
6:         Let s be the sample such that A(s) = ℓ_{i+1}
7:         removed ← CalcRemoved(s)
8:         added ← CalcAdded(s)
9:         gain[i + 1] ← gain[i] − removed + added
10:        Update(IndLeft_0, DLeft_0, IndRight_0, DRight_0)
11:        Update(IndLeft_1, DLeft_1, IndRight_1, DRight_1)
       return max{gain[1 . . . N]}
```

For an efficient implementation all the vectors $IndLeft_c$, $DLeft_c$, $IndRight_c$, $DRight_c$, for $c \in \{0, 1\}$, are stored using a BIT data structure (Fenwick, 1994). Given a vector $A[1..n]$ of n real numbers this data structure requires $O(n)$ space and supports the following operations in $O(\log n)$ time.

- UPDATE(i,x): it replaces the values of $A[i]$ with x;
- SUM(i,j): Return $\sum_{k=i}^{j} A[k]$

Thus, both `GetSum` and $Update$ can be executed in $O(\log N)$ time while only using an extra $O(N)$ storage.

In our discussion we have assumed that there is only one sample s such that $A(s) = \ell_{i+1}$ In the more general case where there are multiple samples with the same value we can execute the same algorithm with the difference that instead of returning the maximum gain among all indexes in the set $\{1, 2, \ldots, n\}$, we return the maximum gain in the set $\{i | \ell_i < \ell_{i+1} \vee i = n\}$.

This way we avoid partitions that separate samples that have the same value for attribute A.

In summary, we have presented an $O(N \log N)$ time algorithm to calculate the gain of a numeric attribute.

3.5 Parameters Estimation

Let $D_{\alpha,M}$ be the tree constructed by `PairTree` when it is executed with parameters `MaxProb` $= \alpha$ and `MinSamples` $= M$. In order to estimate good parameter's values we define a list of candidates $C = (\alpha_1 < \alpha_2 < \cdots < \alpha_k)$ for `MinProb` and a list of candidates $C' = (M_1 < M_2 < \cdots < M_\ell)$ for `MinSamples`. Next, we execute a cross-validation to estimate the cost of $D_{\alpha,M}$, for each $(\alpha, M) \in C \times C'$, and then pick the combination that yield the minimum cost (possibly after running some smoothing procedure). This approach may be very expensive for large datasets because we need to run $k \times \ell$ cross validations.

However, this procedure can be significantly accelerated by using the following simple observation:

Proposition 3. *For each $(\alpha, M) \in C \times C'$, the tree constructed by PairTree for tree $D_{\alpha,M}$ is a sub-tree of D_{α_k,M_1}*

Let's look at an example to clarify Proposition 3. Assume we want to create the trees for parameters `MaxProb` $= \{0.1\}$ and `MinSamples` $= \{100, 500\}$. Figure 3.5 represents the tree $D_{0.1,100}$.

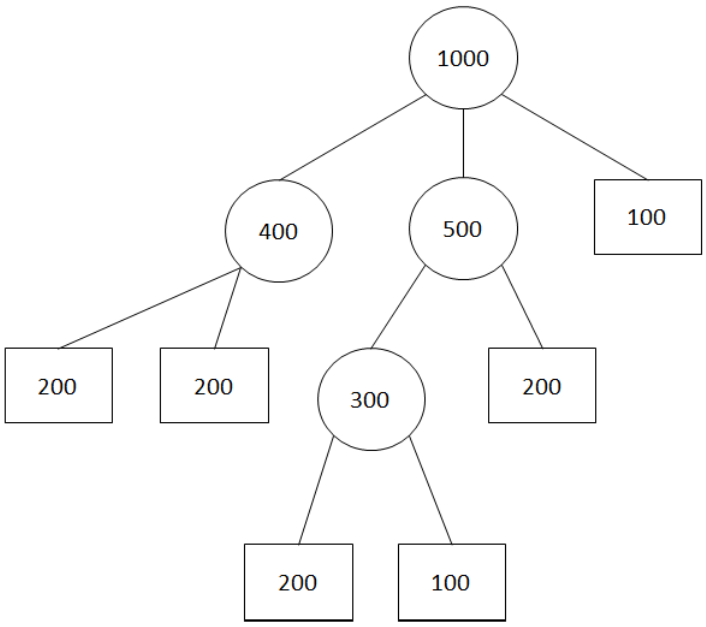


Figure 3.5: Tree for `MaxProb` $= 0.1$ and `MinSamples` $= 100$

It's simple to see we can recreate $D_{0.1,500}$ from $D_{0.1,100}$ simply by removing some nodes, resulting in Figure 3.6.

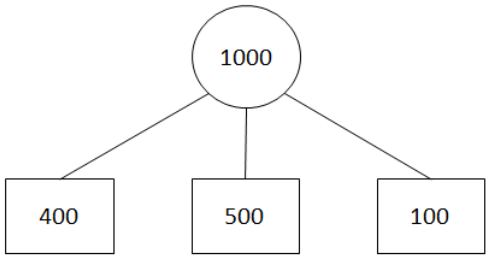


Figure 3.6: Tree for `MaxProb` $= 0.1$ and `MinSamples` $= 500$

The approach for speeding up the $k \times \ell$ cross-validations consists of building an augmented version of tree D_{α_k,M_1}, denoted by D^{aug}. The trees D^{aug} and D_{α_k,M_1} have the same nodes but for each node ν in D^{aug} we store three additional information:

i $|samples(\nu)|$: the number of samples associated with ν;

iii $class(\nu)$, the class with minimum cost for the samples associated with ν;

iii $\hat{\alpha}(\nu) = \min\{\hat{\alpha}(samples(\nu), A) | A \text{ is an attribute}\}$.

This augmented tree contains all the information required to quickly classify a sample s for every tree D_{α_i,M_j}.

For storing D^{aug} we need 3 additional integers/floats per node and the additional time to build D^{aug} is $O(Depth(D_{\alpha_k,M_1})N)$, which is negligible when compared with the time complexity to build D_{α_k,M_1}.

In order to classify a sample s for the different trees $D_{\alpha,M}$ we follow a root-to-leaf path in tree D^{aug} maintaining pointers p_1 and p_2, where:

- p_1 stores the largest i for which we have already visited some node ν' with $\hat{\alpha}(\nu') > \alpha_i$ and
- p_2 stores the minimum integer j for which we have already visited some node ν' with $N(\nu') < M_j$.

Let p_1^a and p_2^a be, respectively, the values of p_1 and p_2 right before visiting node ν. Thus, we can conclude that every tree $D_{\alpha_{i'}}, M_{j'}$ with $i' \leq p_1^a$ or $j' \geq p_2^a$ is pruned at an ancestor of ν (including ν)

Let p_1^b and p_2^b be, respectively, the values of p_1 and p_2 right before visiting node ν.

Thus, we assign $class(\nu)$ to sample s for every tree D_{α_i,M_j} that simultaneously satisfies the formula

$$(i > p_1^b) \wedge (j < p_2^b) \wedge (i \leq p_1^a \vee j \geq p_2^a)$$

We initialize $p_1 = 0$ and $p_2 = \ell + 1$ and it should be clear that p_1^a and p_2^a can be easily obtained given p_1^b and p_2^b.

For classifying a sample s we incur an additional $O(k \times \ell)$ time with regards to the time require to classify s using tree D_{α_k, M_1}. Since the most expensive part of the cross-validation process, in general, is the construction of the decision trees rather than the classification of the samples, we can conclude that the time spent by $k \times \ell$ cross-validations to estimate the best parameters should be slightly larger than that required for executing a single cross validation with parameters α_k and M_1.

4
Experiments

We make an experimental study of our PairTree method on 3 publicly available datasets. As way of comparison we present our results along with other decision tree benchmarks found on the literature for those datasets.

We restrict our analysis to decision tree methods as they are easily interpretable, a highly desirable quality in some applications.

4.1
Data Sets

The datasets were not originally cost-sensitive, however we were able to calculate the misclassification costs using methods described in the literature. These methods are based on each dataset characteristics and are described below.

4.1.1
Direct Marketing

This dataset contains records from clients of a Portuguese bank who were contacted by phone and offered to sign up for a financial product. The clients could either accept or deny the offer. If a client decides to adopt the product the Bank would gain an amount of cash that would depend on the client.

To calculate the cost of classifying a client in either class we use a cost matrix proposed in (Bahnsen et al., 2015) which we show here in table 4.1.

	Predicted	
	Accept	Deny
Accept	C^a	Int_e
Deny	C^a	0

Table 4.1: Cost matrix for sample e

We call the cost of contacting any client C^a and Int_e the amount the bank would make if client e accepts the offer.

This dataset was proposed in (Moro et al., 2014) and is publicly available on the UCI Machine Learning Repository (Lichman, 2013). The dataset originally contained 45211 instances, some of which had missing attributes. After removing instances with missing values we generated a dataset with

37932 instances with 16 attributes which can be either nominal or numeric. In total 12.6% of clients accepted the offer.

4.1.2 Give Me Some Credit

This dataset contains records on clients of a bank and an indicator of whether or not they default their debt in the next 2 years. The cost for the bank of a client default (classified as *bad*) depends on each client.

To calculate the financial cost of a client default we used the framework proposed by (Bahnsen et al., 2015) generating a table shown 4.2.

	Predicted	
	Good	Bad
Good	0	$r_e + C^a$
Bad	$Cl(e) \cdot L_{gd}$	0

Table 4.2: Cost matrix for sample e

The cost of classifying a *good* client as *bad* is given by the amount the bank would've gained with that client r_e plus the expected gain of lending the requested amount to another client C^a. In order to calculate the cost of classifying a *bad* client as *good* we use the assumption that clients don't default their entire debt however they only default a percentage of it, which is given by L_{gd}. So the cost of classifying a *bad* client as *good* is given by the amount of money requested $Cl(e)$ times the average default percentage L_{gd}.

This dataset was provided in the Kaggle competition Give Me Some Credit (Kaggle.com, 2011) and was divided into a training set and a test set. As the competition did not provide the real classification for the test set, we can only use the training set in this work. The dataset originally contained 150000 instances, some of which had missing attributes. We removed the instances with missing values, generating a dataset with 112000 instances with 10 numerical attributes each. In total 6.74% of the clients asked for a default.

4.1.3 PAKDD 2009

This dataset contains records of a Brazilian credit card operator regarding their clients monthly payments. Clients that delayed any monthly payments were labeled as *bad*, and their misclassification costs were specific for each client.

To calculate the cost of classifying a client in either class we selected the same cost matrix by (Bahnsen et al., 2014) used in dataset Give Me Some Credit 4.2.

This dataset was provided as part of the PAKDD 2009 competition (PAKDD, 2009). The dataset originally contained 40000 instances, some of which had missing attributes. We removed the instances with missing values, generating a dataset with 38938 instances and 26 attributes which can be either nominal or numeric. In total 19.8% of the clients are labeled as *bad*.

Table 4.3 presents a summary of our datasets. The column Numeric and Nominal present, respectively, the number of Numeric and Nominal attributes. The column Class Distrib. is the percentage of samples in the least frequent class.

Dataset	Samples	Numeric	Nominal	Class Distrib.
DirectMarketing	37932	3	7	12.6%
GiveSomeCredit	150000	10	0	6.74%
PAKDD2009	38938	11	9	19.8%

Table 4.3: Summary of Datasets

4.2 Benchmarks

In order to compare our method and enrich our analysis we selected two decision tree algorithms from the Example-Dependent Cost-Sensitive Learning literature.

The methods in this section were originally proposed to be used only with binary splits. So to use them in datasets containing k-valued nominal attributes we used two approaches.

The first approach is to adapt each Benchmark to be used with non-binary splits. We explain how this is done in each method below.

The second approach is to transform a k-valued nominal attribute An into k new 0-1 attributes $\{An_1, An_2, \ldots, An_k\}$ where $An_i(e) = 1$ if and only if $An(e) = i$. This technique might prove too slow, and impractical, for large datasets.

4.2.1 Aodha

In (Aodha and Brostow, 2013), it is discussed how to use Random Forest to address the EDCSP for multiple classes. For that, they defined a cost-sensitive impurity measure for building the trees that compose the ensemble.

For a sample s and two classes i and j, define $d^s_{i,j} = C(s,i) - C(s,j)$ if $C(s,i) - C(s,j) \geq 0$ and $d^s_{i,j} = 0$, otherwise

In addition, define

$$f(i,j) = \frac{\sum_{s \in S}(d^s_{i,j})^2}{\sum_{s \in S}(d^s_{i,j} + d^s_{j,i})}$$

The impurity measure for two classes is defined as

$$I_{CS}(S) = \frac{f(0,1)(1 - f(0,1)) + f(1,0)(1 - f(1,0))}{2}$$

The information gain for a binary attribute A that splits the set S into S_L and S_R is given by

$$E_{inf}(S,A) = I_{CS}(S) - \left(\frac{|S_L|}{|S|} I_{SC}(S_L) + \frac{|S_R|}{|S|} I_{SC}(S_R)\right)$$

We can make a natural modification and define the information gain for a k-valued attribute An which splits S into $\{S_1, S_2, \ldots, S_k\}$.

$$E_{inf}(S,An) = I_{CS}(S) - \left(\frac{|S_1|}{|S|} I_{SC}(S_1) + \cdots + \frac{|S_k|}{|S|} I_{SC}(S_k)\right)$$

In their proposal, the attribute with maximum information gain is selected. As a pruning criteria they use the height of the tree and the number of samples per leaf.

We do not use the Random Forest proposed by (Aodha and Brostow, 2013). However we use the decision tree induced by the impurity measure defined above as a benchmark.

4.2.2 Bahnsen

This method was proposed by (Bahnsen et al., 2014) and defines an impurity measure, similar to the one proposed by (Pazzani et al., 1994), which takes into account the cost-sensitive nature of the problem. It uses a top-down approach where in each node the attribute which maximizes a gain metric is selected to make a split, the method then recursively continues on all children.

This algorithm only uses binary splits, where for a numeric attribute A in the range $[a,b]$ it tries all possible splitting parameters $l \in [a,b]$ and selects the one maximizing a gain metric, defined later.

For simplicity we use the notation $C(S,c)$ to denote the cost of classifying all samples in set S as class c, where $c \in \{0,1\}$.

Given a set of samples S an impurity measure, which gives the minimum cost of classifying all samples in S on the same class, is defined:

$$I(S) = \min\{C(S, 0), C(S, 1)\}$$

Using $I(S)$ the gain of a binary attribute A which splits the set S into S_L and S_R is defined, by the authors, as:

$$Gain(A) = (I(S) - (I(S_L) + I(S_R)))/I(S)$$

We can extend the authors $Gain$ definition for a k-valued nominal attribute An which splits S into $\{S_1, S_2, \ldots, S_k\}$.

$$Gain(An) = (I(S) - (I(S_1) + \cdots + I(S_k)))/I(S)$$

The tree stops growing when no attribute generates a $Gain$ greater than some user defined parameter $MinGain$.

We will not use the pruning technique proposed in (Bahnsen et al., 2014) as the authors argued the pruning gave no improvement on their results.

4.3 Testing Environment

All experiments were executed under the following configurations of hardware and software:

Processor	Intel Core i7-4500U
Memory	16GB
OS	Windows 10

Table 4.4: Enviroment specs

All decision tree methods presented in this work were implemented in C++.

4.4 Results

To carry on our experiments we randomly split our dataset into a training set, with 80% of the samples, and a test set containing the remaining 20%.

For evaluating the quality of our methods we used the *savings* metric employed in (Bahnsen et al., 2015). It is defined as the ratio between the cost achieved by a tree T on the set of samples S and the minimum cost of classifying all samples of S on the same class. In formulae,

$$Savings(S,T) = 1 - Cost_T(S)/\min\{Cost(S,0), Cost(S,1)\}, \qquad (4.1)$$

where $Cost_S(T)$ is the total cost achieved by tree T on set S.

4.4.1 PairTree Parameter's estimation

Before executing our decision tree method on the test set we need to decide the value of parameters α and M. This is done by following the procedure described in Section 3.5, using the list of values $\alpha \in \{0.05, 0.1, \ldots, 1\}$ and $M \in \{30, 50, 100, 300, 500, 1000, 2000, 3000, 5000\}$, on a 10-fold cross-validation on the training set of each dataset.

For each combination of α and M we calculate the average Savings of the folds validation sets.

Tables 4.5, 4.6 and 4.7 shows the Savings obtained on some (α, M) combinations. With the highest value of each table in bold letters.

α/M	30	50	100	200	300	500	1000	2000	3000	5000
0.1	46.03	46.03	46.03	46.03	46.03	45.88	45.89	45.10	44.87	43.04
0.2	46.12	46.12	46.12	46.12	46.12	45.79	45.80	44.96	44.74	43.04
0.3	46.30	46.30	46.30	46.30	46.29	45.96	45.95	44.96	44.74	43.04
0.4	46.45	46.45	46.45	46.46	46.46	46.13	46.15	44.96	44.74	43.04
0.5	46.77	46.80	46.82	46.83	46.84	46.36	46.36	44.96	44.74	43.04
0.6	47.38	47.44	47.46	47.46	47.40	46.84	46.36	44.96	44.74	43.04
0.7	47.44	47.48	47.50	**47.58**	47.48	47.17	46.36	44.96	44.74	43.04
0.8	46.30	46.83	47.34	47.43	47.26	47.07	46.31	44.96	44.74	43.04
0.9	44.51	45.83	46.74	47.04	47.24	47.15	46.42	44.98	44.76	43.04
1	30.50	38.57	44.04	46.47	46.92	47.29	46.61	44.98	44.76	43.04

Table 4.5: Cross-Validation for Direct Marketing

α-M	30	50	100	200	300	500	1000	2000	3000	5000
0.1	49.97	49.97	49.97	49.97	49.97	49.97	49.97	49.97	49.97	48.19
0.2	50.52	50.52	50.52	50.52	50.52	50.52	50.53	50.53	50.53	48.72
0.3	50.54	50.54	50.54	**50.54**	50.54	50.48	50.51	50.52	50.58	48.72
0.4	50.3	50.3	50.29	50.29	50.35	50.25	50.4	50.41	50.62	48.72
0.5	50.01	50.03	49.95	49.99	49.98	49.9	50.1	50.23	50.49	48.65
0.6	49.16	49.26	49.14	49.09	49.15	49.35	49.94	50.16	50.55	48.7
0.7	47.45	47.47	47.37	47.61	47.87	48.42	49.51	50.02	50.43	48.66
0.8	45.46	45.63	45.86	47.13	47.93	48.65	49.76	49.95	50.41	48.66
0.9	41.43	42.81	43.99	46.78	47.91	48.89	49.96	49.96	50.39	48.66
1	39.08	41.71	43.5	46.31	47.65	48.89	49.93	49.96	50.39	48.66

Table 4.6: Cross-Validation for GiveMeSomeCredit

α/M	30	50	100	200	300	500	1000	2000	3000	5000
0.1	27.31	27.31	27.31	27.31	27.31	27.31	27.31	27.23	27.23	24.59
0.2	27.20	27.20	27.20	27.20	27.20	27.20	27.26	27.23	27.23	24.59
0.3	27.50	27.50	27.51	27.51	27.72	27.63	27.54	27.21	27.23	24.59
0.4	27.42	27.41	27.44	27.54	27.75	27.67	27.59	27.24	27.23	24.59
0.5	27.62	27.60	27.71	27.59	27.87	27.98	27.60	27.30	27.23	24.59
0.6	26.77	26.79	27.05	26.94	27.37	27.99	27.60	27.30	27.23	24.59
0.7	23.82	24.55	26.23	26.77	27.40	28.15	27.79	27.30	27.23	24.59
0.8	20.31	21.86	25.16	25.88	27.61	28.32	27.91	27.30	27.23	24.59
0.9	16.09	18.79	24.15	25.42	27.64	**28.30**	27.91	27.30	27.23	24.59
1	15.79	18.91	24.11	25.41	27.64	28.30	27.91	27.30	27.23	24.59

Table 4.7: Cross-Validation PAKDD 2009

It's interesting to observe the impact of parameters α and M by looking at their smallest and largest values.

Large values of α means we are only using our *Gain* metric and the probability bounds proposed in Chapter 3 to select attributes in a node. But not as a way to stop growing the tree when we think the attributes are not better than the "'random attribute"', which probably causes an overfit. While for small values of α we might be too strict and stop growing the tree too soon, causing an underfit.

Large values of M have particularly worst impact on smaller datasets such as Direct Marketing and PAKDD2009 as those combinations often create the same small tree.

Another important aspect to look at is how the Savings on a cell behaves when compared to it's neighbors. We want to select a combination (α_i, M_j) with high Savings on it and around it. This shows the region shall be stable and without much random noise.

We define a simple smoothing rule to make the parameters selection.

$$Smooth(i,j) = \frac{Sv(i,j) + 0.8 \cdot (Sv(i-1,j) + Sv(i+1,j) + Sv(i,j-1) + Sv(i,j+1))}{4.2},$$

where $Sv(i,j)$ is the Savings of combination (α_i, M_j).

As a future work it makes sense to consider the effect of applying a more sophisticated procedure such as a Gaussian smoothing.

Table 4.8 shows the selected parameters using the smoothing rule for each dataset.

Dataset	α	M
Direct Marketing	0.65	200
Give Me Some Credit	0.25	200
PAKDD2009	0.9	500

Table 4.8: PairTree best parameters

Table 4.9 presents the elapsed time to execute a 10-fold cross-validation for estimating the best parameters in the training set.

Dataset	**Samples**	**Numeric**	**Nominal**	**Time (sec)**
DirectMarketing	37932	3	7	20
GiveSomeCredit	112000	10	0	180
PAKDD2009	38938	11	9	53

Table 4.9: Time required for parameter estimation

The Give Me Some Credit dataset showed to be the slowest one as it has the largest number of samples. This result is expected as we showed before on Section 3.4 the cost of calculating *Gain* of each numeric and nominal attributes is $O(N \log N)$, where N is the number of samples.

4.4.2 Benchmark Parameter's estimation

To estimate the benchmarks parameters we used the same approach previously discussed for our PairTree method. The only difference being the estimated parameters.

We believe this to be a fair approach as the authors don't mention how to make the parameter selection for their methods.

For Aodha's tree we only had one parameter in which to optimize, the minimum number of samples in each node M. While Bahnsen had two, $MinGain$ which gives the minimum $Gain$ necessary to continue growing the tree, and the minimum number of samples per node M.

After executing the parameter estimation procedure we used the smoothing function described in the last subsection, to get the best parameter combination. This combination was then used on the test set to get the final results.

Table 4.10 shows the selected parameters for the benchmarks using binary splits.

	Aodha	Bahnsen	
	M	$MinGain$	M
Direct Marketing	1000	0.009	200
Give Me Some Credit	1000	0.0015	3000
PAKDD2009	1000	0.005	50

Table 4.10: Benchmarks parameters

4.4.3 Results on the Testing Set

Here we present our results on the test set along with the Benchmarks for comparison.

Table 4.11 shows the Savings for fitting our decision tree method on the training set, and testing it on the test set. In each dataset we show results using only binary splits labeled as "'Binary"' and using binary splits for numeric attributes and k-valued splits for nominal attributes labeled as "'All"'. The Give Me Some Credit dataset was composed of only numeric attributes, therefore we always used binary splits.

Dataset	**Split**	**Bahnsen**	**Aodha**	**PairTree**
DirectMarketing	Binary	47.24%	**47.36%**	47.16%
DirectMarketing	All	**47.16%**	46.82%	46.49%
GiveSomeCredit	All	47.51%	45.41%	**50.59%**
PAKDD2009	Binary	28.37%	28.64%	**29.82%**
PAKDD2009	All	28.37%	28.65%	**29.82%**

Table 4.11: Results on testing set

The Direct Marketing dataset showed the least difference in Savings among the three methods and was the only dataset in which our method lost to both benchmarks.

Our method, PairTree, showed the best results for Give Me Some Credit, 6% lead over second best, and PAKDD2009 with a 4% lead.

In Table 4.12 we show the times, in seconds, necessary to create each tree before testing it on the test set.

Dataset	**Split**	**Bahnsen**	**Aodha**	**PairTree**
DirectMarketing	Binary	< 1	2	8
DirectMarketing	All	< 1	1	2
GiveSomeCredit	All	2	4	14
PAKDD2009	Binary	< 1	2	10
PAKDD2009	All	< 1	2	5

Table 4.12: Tree creating times on testing set in seconds

Although our method is slower than the others, it is still fast enough to address relatively large datasets. Our metric also has the added advantage of protecting against overfit.

5
Final remarks

On this dissertation we discussed the Example-Dependent Cost-Sensitive Problem, focusing on decision tree methods. We presented recent results from the literature and how it compared to our proposed method.

Our PairTree method is based on a *Gain* metric which tries to capture the potential of an attribute to separate pairs of examples. We used a bound on the sum of random dependent variables, proposed on (Janson, 2004), to estimate how an attribute's *Gain* performs when compared to random attributes.

We achieved superior results when compared to the decision tree benchmarks (Bahnsen et al., 2015) and (Aodha and Brostow, 2013) on two datasets Give Me Some Credit and PAKDD2009. And similar results on the third dataset Direct Marketing.

As future work it's natural to think of an extension of our PairTree method to work on a multi-class classification tasks. This can probably be achieved with some changes to our *Gain* measure and an update on the implementation aspects in order to keep the time complexity low.

Another interesting extension to consider is the application of the PairTree method on Random Forests. This idea has the potential to increase the Savings metric.

6
Bibliography

AODHA, O. M.; BROSTOW, G. J. Revisiting example dependent cost-sensitive learning with decision trees. In **Proceedings of the IEEE International Conference on Computer Vision**. [S.l.: s.n.], 2013. p. 193–200. 1.2, 1.3, 4.2.1, 5

BAHNSEN, A. C.; AOUADA, D.; OTTERSTEN, B. Example-dependent cost-sensitive logistic regression for credit scoring. In IEEE. **Machine Learning and Applications (ICMLA), 2014 13th International Conference on**. [S.l.], 2014. p. 263–269. 4.1.3, 4.2.2

BAHNSEN, A. C.; AOUADA, D.; OTTERSTEN, B. Example-dependent cost-sensitive decision trees. **Expert Systems with Applications**, Elsevier, vol. 42, no. 19, p. 6609–6619, 2015. 1.2, 1.3, 3.1, 4.1.1, 4.1.2, 4.4, 5

BREIMAN, L. et al. **Classification and regression trees**. [S.l.]: CRC press, 1984. 1.3

CICALESE, F.; LABER, E. S.; SAETTLER, A. M. Diagnosis determination: decision trees optimizing simultaneously worst and expected testing cost. In **ICML**. JMLR.org, 2014. (JMLR Workshop and Conference Proceedings, vol. 32), p. 414–422. Available from Internet: <http://jmlr.org/proceedings/papers/v32/>. 3.1

ELKAN, C. The foundations of cost-sensitive learning. In LAWRENCE ERLBAUM ASSOCIATES LTD. **International joint conference on artificial intelligence**. [S.l.], 2001. vol. 17, no. 1, p. 973–978. 1.3

FENWICK, P. M. A new data structure for cumulative frequency tables. **Software: Practice and Experience**, Wiley Online Library, vol. 24, no. 3, p. 327–336, 1994. 1.2, 3.4.2

GOLOVIN, D.; 0001, A. K.; RAY, D. Near-optimal bayesian active learning with noisy observations. In LAFFERTY, J. D. et al. (Ed.). **NIPS**. Curran Associates, Inc, 2010. p. 766–774. Available from Internet: <http://papers.nips.cc/book/advances-in-neural-information-processing-systems-23-2010>. 3.1

GREINER, R.; GROVE, A. J.; ROTH, D. Learning cost-sensitive active classifiers. **Artificial Intelligence**, Elsevier, vol. 139, no. 2, p. 137–174, 2002. 1.3

JANSON. Large deviations for sums of partly dependent random variables. **RSA: Random Structures Algorithms**, vol. 24, 2004. 2.3, 1, 5

KAGGLE.COM. **Give Me Some Credit — Kaggle**. 2011. Available from Internet: <https://www.kaggle.com/c/GiveMeSomeCredit/>. 1.2, 4.1.2

LICHMAN, M. **UCI Machine Learning Repository**. 2013. Available from Internet: <http://archive.ics.uci.edu/ml>. 4.1.1

LING, C. X. et al. Decision trees with minimal costs. In ACM. **Proceedings of the twenty-first international conference on Machine learning**. [S.l.], 2004. p. 69. 1.3

LOMAX, S.; VADERA, S. A survey of cost-sensitive decision tree induction algorithms. **ACM Computing Surveys (CSUR)**, ACM, vol. 45, no. 2, p. 16, 2013. 1.3

MORO, S.; CORTEZ, P.; RITA, P. A data-driven approach to predict the success of bank telemarketing. **Decision Support Systems**, Elsevier, vol. 62, p. 22–31, 2014. 1.2, 4.1.1

MURTHY, S. K. Automatic construction of decision trees from data: A multi-disciplinary survey. **Data mining and knowledge discovery**, Kluwer academic publishers, vol. 2, no. 4, p. 345–389, 1998. 2.1

PAKDD. **13th Pacific-Asia Knowledge Discovery and Data Mining conference**. 2009. Available from Internet: <http://sede.neurotech.com.br/PAKDD2009>. 1.2, 4.1.3

PAZZANI, M. et al. Reducing misclassification costs. In **Proceedings of the Eleventh International Conference on Machine Learning**. [S.l.: s.n.], 1994. p. 217–225. 1.3, 4.2.2

TURNEY, P. D. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. **Journal of artificial intelligence research**, vol. 2, p. 369–409, 1995. 1.3

VIZING., V. G. On an estimate of the chromatic class of a p-graph (in russian). **Diskret. Analiz**, vol. 3, p. 23–30, 1964. 2.2

ZADROZNY, B.; LANGFORD, J.; ABE, N. Cost-sensitive learning by cost-proportionate example weighting. In IEEE. **Data Mining, 2003. ICDM 2003. Third IEEE International Conference on**. [S.l.], 2003. p. 435–442. 1.3

A
Proofs

Proposition 1. $G(x,y) = \min\{D(x), D(y)\}$

*Proof.*Assuming $x \in S_0$ and $y \in S_1$ we can rewrite $G(x,y)$ as

$$\begin{aligned} G(x,y) = \min\{C(x,0) + C(y,0), C(x,1) + C(y,1)\} \\ -C(x,0) - C(y,1) \end{aligned} \tag{A.1}$$

Let's consider two cases:

i)

$$D(x) \leq D(y) \tag{A.2}$$

$$C(x,1) - C(x,0) \leq C(y,0) - C(y,1) \tag{A.3}$$

$$C(x,1) + C(y,1) \leq C(y,0) + C(x,0) \tag{A.4}$$

Applying A.4 to A.1

$$G(x,y) = C(x,1) + C(y,1) - C(x,0) - C(y,1)$$

$$G(x,y) = C(x,1) - C(x,0) \tag{A.5}$$

As $x \in S_0$ therefore $C(x,1) > C(x,0)$, follows

$$G(x,y) = D(x) \tag{A.6}$$

ii)

$$D(x) > D(y) \tag{A.7}$$

$$C(x,1) - C(x,0) > C(y,0) - C(y,1) \tag{A.8}$$

$$C(x,1) + C(y,1) > C(y,0) + C(x,0) \tag{A.9}$$

Applying A.9 to A.1

$$G(x,y) = C(x,0) + C(y,0) - C(x,0) - C(y,1)$$

$$G(x,y) = C(y,0) - C(y,1) \tag{A.10}$$

As $y \in S_1$ therefore $C(y,0) > C(y,1)$, follows

$$G(x,y) = D(y) \tag{A.11}$$

Now we can write $G(x, y)$ as

$$G(x, y) = \min\{D(x), D(y)\} \tag{A.12}$$

□

Proposition 2.

$$\begin{aligned} Gain(A) = &\sum_{x \in S_0} D(x) \cdot |\{y | A(x) \neq A(y) \wedge D(x) \leq D(y)\}| \\ &+ \sum_{y \in S_1} D(y) \cdot |\{x | A(x) \neq A(y) \wedge D(x) > D(y)\}| \end{aligned} \tag{A.13}$$

*Proof.*Assuming $x \in S_0$ and $y \in S_1$.

$$Gain(A) = \sum_{A(x) \neq A(y)} G(x, y) \tag{A.14}$$

$$= 1/2 * (\sum_{x} \sum_{y | A(x) \neq A(y)} G(x, y) + \sum_{y} \sum_{x | A(x) \neq A(y)} G(x, y)) \tag{A.15}$$

We'll consider pairs (x, y) where $D(x) \leq D(y)$ in the first sum and the other pairs on the second sum

$$= \sum_{x} \sum_{y | A(x) \neq A(y) \wedge D(x) \leq D(y)} G(x, y) + \sum_{y} \sum_{x | A(x) \neq A(y) \wedge D(x) > D(y)} G(x, y) \tag{A.16}$$

Applying Proposition 1

$$= \sum_{x} \sum_{y | A(x) \neq A(y) \wedge D(x) \leq D(y)} D(x) + \sum_{y} \sum_{x | A(x) \neq A(y) \wedge D(x) > D(y)} D(y) \tag{A.17}$$

$$\begin{aligned} &= \sum_{x} D(x) \cdot |\{y | A(x) \neq A(y) \wedge D(x) \leq D(y)\}| \\ &+ \sum_{y} D(y) cdot |\{x | A(x) \neq A(y) \wedge D(x) > D(y)\}| \end{aligned} \tag{A.18}$$

□