Pontifícia Universidade Católica
do Rio de Janeiro

Felipe de Albuquerque Mello Pereira

A Framework for Generating Binary Splits in Decision Trees

Dissertação de Mestrado

Thesis 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
March 2018

Pontifícia Universidade Católica
do Rio de Janeiro

Felipe de Albuquerque Mello Pereira

A Framework for Generating Binary Splits in Decision Trees

Thesis presented to the Postgraduate Program in Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

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

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática — PUC-Rio

Prof. Marco Serpa Molinaro
Departamento de Informática — PUC-Rio

Prof. Márcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico da PUC-Rio

Rio de Janeiro, March 9^{th}, 2018

Felipe de Albuquerque Mello Pereira

Bachelor's in Electrical Engineering and Pure Mathematics at the Pontifícia Universidade Católica do Rio de Janeiro (2010 and 2011). Masters' in Mathematics at the Pontifícia Universidade Católica do Rio de Janeiro (2013).

Bibliographic data

Pereira, Felipe de Albuquerque Mello

A Framework for Generating Binary Splits in Decision Trees / Felipe de Albuquerque Mello Pereira ; advisor: Eduardo Sany Laber. — 2018.
56 f. : il. ; 30 cm

Dissertação (Mestrado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2018.
Inclui bibliografia

1. Informática – Teses. 2. Árvores de Decisão; Problema de Corte Máximo; Algoritmos Aproximativos. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

TODO: acknowledgment.

Thanks to CNPq for the conceded scholarship during my Masters.

Abstract

Pereira, Felipe de Albuquerque Mello; Laber, Eduardo Sany (advisor). **A Framework for Generating Binary Splits in Decision Trees**. Rio de Janeiro, 2018. 56p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In this dissertation we propose a framework for designing splitting criteria for handling multi-valued nominal attributes for decision trees. Criteria derived from our framework can be implemented to run in polynomial time in the number of classes and values, with theoretical guarantee of producing a split that is close to the optimal one. We also present an experimental study, using real datasets, where the running time and accuracy of the methods obtained from the framework are evaluated.

Keywords

Decision Trees; Max-cut Problem; Approximation Algorithms

Resumo

Pereira, Felipe de Albuquerque Mello; Laber, Eduardo Sany. **Um Framework para Geração de Splits Binários em Árvores de Decisão**. Rio de Janeiro, 2018. 56p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Nesta dissertação é apresentado um framework para desenvolver critérios de split para lidar com atributos nominais multi-valorados em árvores de decisão. Critérios gerados por este framework podem ser implementados para rodar em tempo polinomial no número de classes e valores, com garantia teórica de produzir um split próximo do ótimo. Apresenta-se também um estudo experimental, utilizando datasets reais, onde o tempo de execução e acurácia de métodos oriundos do framework são avaliados.

Palavras-chave

Árvores de Decisão; Problema de Corte Máximo; Algoritmos Aproximativos

Contents

List of Figures

List of Tables

1
Introduction

Decision Trees and Random Forests are among the most popular methods for classification tasks. Decision Trees, specially small ones, are easy to interpret, while Random Forests usually yield more accurate classifications. One of the key issues in these methods is how to select an attribute to associate with a node of the tree/forest. An important related issue is how to split the samples once the attribute is selected.

There is a number of papers discussing aspects related with attribute selection, such as: how to design criteria to evaluate the quality of different types of attributes; whether binary or multi-way splits shall be used and how to remove bias from splitting criteria. For recent surveys on this topic we refer to Goethals & Rokach (2005), Loh (2014) and Barros et al. (2015).

Many criteria, with different properties, have been proposed to evaluate the quality of different types of attributes, including continuous and categorical ones. Among the most popular criteria, we have the Gini Gain and the Information Gain (Breiman et al. (1984), Quinlan (2014)).

Despite the large body of work we believe there are still questions to be answered. One of them is to how to properly handle nominal attributes that may assume a large number of values. Before explaining the reason behind our statement we would like to remark that this kind of attribute appears naturally in some applications (e.g.: states of a country or letters from some alphabet). In addition, they may arise as the result of aggregating attributes that have few distinct values with the goal of capturing possible correlation between them, as pointed out by Chou (1991). As an example, consider 5 binary attributes (e.g. medical tests) and a target binary variable that has large probability of being positive if at least 3 out of the 5 binary tests are positives. By aggregating the 5 binary variables we obtain a new attribute with $2^5 = 32$ values that captures this relation. If we used the 5 attributes separately we would need 5 levels in the tree to be able to capture the relation between them and the target class, thus incurring a large fragmentation of the set of samples.

To properly face multi-valued nominal attributes we have to deal with the computational time required to compute good splits. Our contribution, explained in the next section, is related with this issue.

A brute force search to compute the best binary split requires $\Omega(2^n)$ time, where n is the number of distinct values the attribute may assume. The computational efficiency can be improved if a n-ary split is used rather than a binary one. However, this may lead to a severe fragmentation of the sample space, which is not desirable: the number of samples available for each of the children of the split node may be small and, as a consequence, the underlying classification tasks may become significantly more difficult. When the target variable is binary, a family of impurity measures that include both the Gini Gain and the Information Gain can be computed efficiently, as shown in the influential monograph by Breiman et al Breiman et al. (1984). However, when the number of classes k is larger than 2, most, if not all, of the available exact solutions take exponential time in (n, k). The Twoing method, also from Breiman et al. (1984), is an interesting case since its running time is $O(2^{\min\{n,k\}})$ rather than $O(2^n)$ while being equivalent to Gini Gain when $k = 2$.

When both n and k are large, in the sense that an exhaustive search does not run in a reasonable time, one can rely on heuristics to compute the best binary split. As an example, the GUIDE algorithm Loh (2009), the last of a series of algorithms/developments designed by Loh and its contributors, deals with a nominal variable X as follows: if $k = 2$ or $n \leq 11$ the Gini Index is computed; if $k \leq 11$ and $n > 20$ a new variable X' with at most k distinct values is created according to a certain rule and an exhaustive search is performed over it; finally, if $k > 11$ or $n \leq 20$, X is binarized and a Linear Discriminant Analysis (LDA) is employed. These rules reflect the difficulty in dealing with multi-valued nominal attributes. Other interesting heuristics are the PC and PC-ext criteria, which calculate the principal component of the class probability vectors and uses the order given by the vector projections in this direction to look for splits. In general, the main drawback of using heuristics is the lack of a theoretical guarantee about their behavior.

1.1 Our Contribution

Given this scenario, in chapter 3 we propose a framework for designing criteria, with nice theoretical properties, for evaluating the quality of multi-valued nominal attributes. In general, finding the best binary partition according to an impurity measure has been proved to be NP-complete in Laber et al. (2018). Nonetheless, criteria generated according to our framework run in polynomial time in both the number of values and classes and have a theoretical guarantee that they are close to optimal. The key idea consists of formulating the problem of finding the best binary partition for a given at-

tribute A as the problem of finding a cut with maximum weight in a complete graph whose nodes are associated with the values that A may assume and the edges' weights capture the benefit of putting values in different partitions. The motivation behind the use of the max-cut problem is the existence of efficient algorithms with approximation guarantee, in particular the one proposed by Goemans & Williamson (1995), with 0.878 approximation, and local search algorithms with 0.5-approximation as shown in Angel et al. (2017).

We discuss two criteria that are derived from this framework: the first one can be seen as a natural variation of the Gini Gain, while the second criterion uses the χ^2-test to set the edges' weights. For that, each edge e_{ij} between nodes v_i and v_j is thought as a binary attribute $A(i, j)$ with values v_i and v_j. After discussing these criteria, we show how to extend them to handle numeric attributes.

We also present a number of experiments that suggest that one of our criteria is competitive with the Twoing method, which is – as far as we know – the only well-established criterion with binary splits that can be optimally computed for large n when $k > 2$. However, in contrast with our methods, Twoing cannot handle datasets that also have a large number of classes. Some criteria based on heuristics, such as the PC-ext and Hypercube Cover, are also part of the comparison. In addition, the experiments also provide evidence of the potential of aggregating attributes for improving the accuracy of decision trees.

TODO: falar no paragrafo acima um pouco mais da comparacao com outros metodos.

1.2 Related Work

Many splitting criteria have been proposed to deal with continuous and nominal attributes. Arguably, the Gini Gain—used by CART—and entropy-based measures—such as the Information Gain, adopted by C4.5—are among the most popular (Goethals & Rokach (2005); Loh (2014); Barros et al. (2015)).

There has been some investigation on methods to compute the best split efficiently (Breiman et al. (1984); Chou (1991); Burshtein et al. (1992); Coppersmith et al. (1999)). For the 2-class problem, Breiman et al. (1984) proved a theorem which states that an optimal binary partition, for a certain class of splitting criteria, can be determined in linear time on n, the number of distinct values of the attributes, after ordering. The Gini Gain belongs to this class. The other three papers generalize this theorem in different directions and show necessary conditions that are satisfied by optimal partitions for a

certain class of splitting criteria. These conditions, though useful to restrict the set of partitions that need to be considered, do not yield a method that is efficient (polynomial time) for large values of n and k. These papers also present heuristics, without approximation guarantee, to obtain good splits. Another related result is a theorem from Coppersmith et al. (1999) that guarantees that the optimum split can be found by separating the class probability vectors by a hyperplane. This motivated the creation of the PC criterion, as will be shown in the next chapter. Two other existing heuristics are the SLIQ (Mehta et al. (1996) and the FlipFlop criterion (Nádas et al. (1991)). While they execute in polynomial time, they have no approximation guarantee in terms of the impurity of the optimal partition. Moreover, they usually finds worse partitions than the PC criterion, as shown in the experiments section of Coppersmith et al. (1999). Lastly, Laber et al. (2018) presents two heuristics that have approximation guarantees: the Hypercube Cover and the Largest Class Alone. The first has exponential running time on the number of classes and is similar to the Twoing criterion, but has an approximation guarantee of 2 for every impurity measure as defined in the paper. The second one, Largest Class Alone, runs in quasilinear time on the number of values and linear time on the number of classes. This criterion has an approximation guarantee of 2 for the Gini impurity and 3 for the Entropy impurity.

Other proposals to speed up the attribute selection phase include Mola & Siciliano (1997); Shih (2001). The first presents a simple heuristic to reduce the number of binary splits considered to choose the best nominal variable among the m available ones. The second extends the method for another class of impurity measures.

In order to properly handle nominal attributes with a large number of values, apart from efficiently computing good splits, it is important to prevent bias in the attribute selection. Indeed, it is widely known that many splitting criteria have bias toward attributes with a large number of values. There are some proposals available to cope with this issue (Dobra & Gehrke (2001); Shih (2004); Hothorn et al. (2006)). This topic, though relevant, is not the focus of this dissertation.

1.3 Organization

In Chapter 2 we explain how decision trees are used for classification problems and how they are constructed. We also present the main impurity measures and splitting criteria used in the literature, together with their execution-time complexity.

Chapter 3 contains the framework for generating splitting criteria that run in polynomial time. Its relation with the Max-Cut problem and its approximation algorithms are explained and some criteria obtained from this framework are presented.

Later, in Chapters 4 and 5, we compare the proposed criteria and see how they perform in practice. In Chapter 4 we explore how the many heuristics used to find splits with optimal impurity compare among themselves. This suggests a couple of criteria that perform better and can be used when the number of values and classes are large. In Chapter 5 we analyze these methods on real datasets that contain attributes with large number of values and classes. This analysis is done using different maximum depth allowed for the decision tree, which is allows us to see the the advantages and shortcomings of each criterion. Lastly, in Chapter 6 we present our study conclusions.

2
Background

2.1
Decision Trees

Decision trees are trees where each internal node is associated with an attribute and each of the edges leading out from it are associated with a subset of this attribute's values. The leaf nodes are associated with a class. In order to classify a sample, we start at the root of the tree and follow the edges given by the sample attributes' values. Once we get to a leaf node, we've found the class prediction for this sample.

As an example, let's look at the Titanic Survival dataset (Kaggle (2012)). The Figure 2.1 shows the first 10 samples from the training dataset, where we are trying to predict the value for the Survived field.

PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived
1	3	male	22	1	0	7.25	S	0
2	1	female	38	1	0	71.2833	C	1
3	3	female	26	0	0	7925	S	1
4	1	female	35	1	0	53.1	S	1
5	3	male	35	0	0	8.05	S	0
6	3	male		0	0	8.4583	Q	0
7	1	male	54	0	0	51.8625	S	0
8	3	male	2	3	1	21075	S	0
9	3	female	27	0	2	11.1333	S	1
10	2	female	14	1	0	30.0708	C	1

Figure 2.1: First ten samples from the Titanic Survival dataset.

A possible decision tree is the one given in the Figure 2.2. This tree correctly classifies all the first 10 samples. For instance, since the passanger with Id 1 is male and older than 9.5 years, the decision tree predicts him as someone who did not survive.

The main theme of this dissertation is how to (quickly) create such trees, specially when we have categorical attributes with a large number of values.

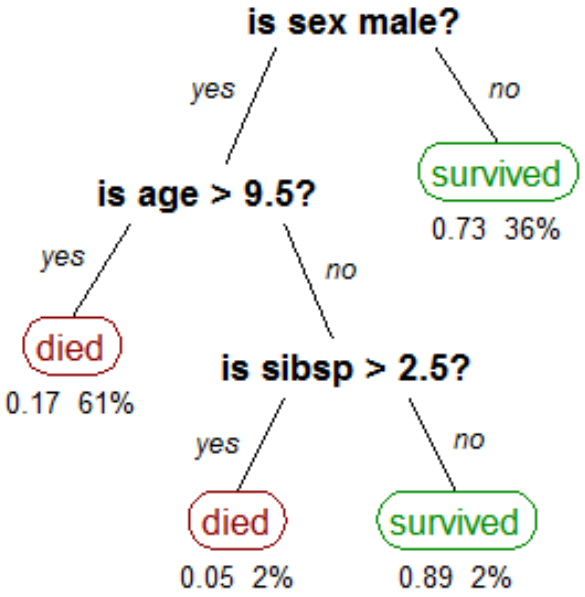

Figure 2.2: Possible decision tree for the Titanic Survival dataset.

2.2 Notation

We adopt the following notation throughout the dissertation. Let S be a set of N samples and $C = \{c_1, \ldots, c_k\}$ be the domain of the class label. In addition, for an attribute A, we use $A(s)$ to denote the value taken by attribute A on sample s; we use $V = \{v_1, \ldots, v_n\}$ to denote the set of values taken by A; A_{ij} to refer to the number of samples from class c_j for which A takes value v_i; N_i for the number of samples with value v_i for attribute A and S_j for the number of samples from class c_j. Given a partition (L, R) of the values V of an attribute A, we use S_L and S_R to denote the set of samples in S that takes values in L and R, respectively, in attribute A. Furthermore, we let $p_j = S_j/N$ and $p_{ij} = Pr[C = c_j|A = v_i]$. We observe that the estimator of maximum likelihood for p_{ij} is A_{ij}/N_i.

2.2.1 A Typical Algorithm for Inducing Decision Trees

Many of the decision tree inducers follow the structure of Algorithm 1. This algorithm has a few under-specified parts, namely the stopping and splitting criteria.

The stopping criterion is used to avoid over-fitting the decision tree. Stopping criteria usually involve stopping when every sample has the same

Algorithm 1 CreateTree(S: set of samples, $List\_A$: list of attributes information)

```
if S does not meet the stopping criterion then
  for attribute A in List_A do
    s_A = split of A given by a splitting criterion using I
  end for
  (L, R) = s_{A*} that is best among all I(s_A), according to the splitting criterion
  CreateTree(L)
  CreateTree(R)
end if
```

class or when the number of samples is small. Another possibility sometimes used is to skip an attribute when a statistical test indicates its contingency table has almost no information—i.e., it is not statistically different from a randomly generated one—, or in the obvious case where all the samples have the same value. Finally, the splitting criteria will be the object of our studies in the rest of this chapter.

2.3 Impurity Measures

Breiman et al. (1984) suggests that a natural family of splitting criteria are the ones that select splits which have the smallest impurity according to a certaing impurity measure. Therefore we will make impurity measures our main object of study in this section, while splitting criteria will be studied in the Section 2.4.

First, it is important to note that most impurity calculations on a nominal attribute are done based on what's called a contingency table. It consists of a $n \times k$ matrix where the ij-th entry corresponds to the number of samples that have value i and belong to class j. We'll assume that every decision tree node has the contingency table pre-calculated for all nominal attributes, which takes $O(N)$ time for each attribute and cannot be avoided[1]. Therefore, whenever we mention the time complexity for a decision tree constructing algorithm/criterion, this cost will not be mentioned since it does not affect their comparison.

The only thing left is to define what we mean by an impurity measure. Breiman et al. (1984) defines a class of impurity measures that are natural in this context.

[1]The only exception is the attribute used to split the parent node, which can be calculated in $O(n \times k)$.

Definition 1 (Impurity Measure Function) *A function $\phi(p)$ is a node impurity function iff*

1. $\phi : [0, 1] \to \mathbb{R}_+$;
2. ϕ *has continous second derivatires;*
3. $\phi(0) = \phi(1) = 0$;
4. $\phi(p) = \phi(1 - p)$ *(ϕ is symmetric);*
5. $\phi''(p) < 0$ *for* $p \in (0, 1)$ *(ϕ is concave).*

Definition 2 (Node Impurity Measure) *A node impurity measure I is a function $I : [0, 1]^k \to \mathbb{R}$. It is given by*

$$I(\mathbf{p}) = \sum_{j=1}^{k} \phi(p_j)$$

where ϕ is a node impurity function.

The node impurity measure is frequently denoted by $I(S)$, where S is a set of samples. This indicates the impurity measure calculated on the vector $\mathbf{p} = (c_1/N, \ldots, c_k/N)$.

Definition 3 (Split Impurity Measure) *A split impurity measure is the impurity difference between the original node and its child nodes, weighted by the fraction of samples in each node. For a binary partition $V = (L, R)$,*

$$\Delta I(L, R) = I(S) - p_L \cdot I(S_L) - p_R \cdot I(S_R)$$

where p_L and p_R are equal to the ratio of the samples in L and R, respectively.

These impurity measures have the nice property that splitting V into (L, R) always decreases the impurity, as also shown in Breiman et al. (1984).

Theorem 4 *Given an impurity measure I and any partition of values V into (L, R), we always have*

$$\Delta I(L, R) \geq 0$$

Equality happens iff the contingency table in both sides have the same frequency distribution.

Another interesting result proven in Breiman et al. (1984) is that, when there are only 2 classes, we can find the partition with best impurity in linear time after sorting.

Theorem 5 *Let I be an impurity measure and suppose we only have two classes c_1 and c_2. The best partition of the values V has the form*

$$L = \{v_i : p_{i1} \leq v\}$$

$$R = \{v_i : p_{i1} > v\}$$

for some value v.

The result above is used by many splitting criteria, as will be mentioned in Section 2.4.

Now let's present the two most common impurity measures found in the literature. Both can be used to generate binary splits and, as a consequence, binary decision trees.

Definition 6 *The Gini impurity for a set of samples S is given by*

$$Gini(S) = \sum_{i=1}^{k} p_i(1 - p_i) = 1 - \sum_{i=1}^{k} (p_i)^2 \tag{2-1}$$

Definition 7 *The Entropy for a set of samples S is given by*

$$Entropy(S) = \sum_{i=1}^{k} p_i \log\left(\frac{1}{p_i}\right) = -\sum_{i=1}^{k} p_i \log(p_i) \tag{2-2}$$

2.4 Splitting Criteria

In this section we recall some well-known splitting criteria.

First note that, for numeric attributes, most criteria follow the same algorithm to choose the best split: the values are ordered and the valid splits have the form

$$L = \{v_i : v_i \leq v\}$$

$$R = \{v_i : v_i > v\}$$

for some chosen value v. This is because all the reasonable splits should preserve and use the order structure.

The criteria evaluate the impurity of these splits and chooses the one with the smallest impurity. Since we only have to evaluate at a single value v between

each pair of consecutive values v_i and v_{i+1}, it takes $O(n \log n)$ time. Since this is polynomial and quite fast, its complexity is largely ignored throughout this dissertation.

2.4.1 Gini Gain

The Gini Gain, Δ_G, induced by a binary partition (L, R) of the set of values V is given by

$$\Delta_G(L, R) = Gini(S) - p_L Gini(S_L) - p_R Gini(S_R), \tag{2-3}$$

where $S_L = \{s \in S | A(s) \in L\}$, $S_R = \{s \in S | A(s) \in R\}$, $p_L = |S_L|/N$ and $p_R = |S_R|/N$. Therefore, the largest the Gini Gain is, the better the partition.

To generate the partition with the largest gain, one can consider all 2^n binary values' split and select the partition with maximum Gini Gain. For the two-class problem this optimal partition can be computed in $O(n \log n)$ time by using Theorem 5. Since we only need to test a single value of v that is between each pair of consecutive values v_i, v_{i+1}, the time complexity follows.

For problems with more than two classes, however, there is no efficient procedure with theoretical approximation guarantee to compute the Gini Gain in subexponential time in n. Heuristics and approximation algorithms are discussed in Section 2.5.

2.4.2 Twoing

The Twoing criterion for a binary partition (L, R) of the set of values V is given by

$$0.25 \cdot p_L \cdot p_R \cdot \left(\sum_{i=1}^{k} |p_L^i - p_R^i| \right)^2$$

where

$$p_L^i = \frac{|\{s \in S_L : s \text{ belongs to class } c_i\}|}{|S_L|}$$

and

$$p_R^i = \frac{|\{s \in S_R : s \text{ belongs to class } c_i\}|}{|S_R|}$$

When the Twoing criterion is used to generate binary decision trees, the binary partition with maximum twoing shall be selected at each node.

In Breiman et al. (1984) there is a theorem that states that the optimal partition of Twoing can be found in a different way: by considering all possibilities of partitioning the classes into two superclasses and solving the

2-class problem with Gini impurity on each of them, picking the best partition overall. Therefore, by using Theorem 5, the best partition can be found in $O(\min\{n(k + \log n)2^k, 2^n\})$ time. Lastly, we shall remark that, for the two-class problem, the Twoing criterion and the Gini Gain compute the same binary partitions.

2.4.3 Information Gain

The Information Gain, IG, induced by a binary partition (L, R) of the set of values V is given by

$$IG(L, R) = Entropy(S) - p_L Entropy(S_L) - p_R Entropy(S_R), \tag{2-4}$$

where $S_L = \{s \in S | A(s) \in L\}$, $S_R = \{s \in S | A(s) \in R\}$, $p_L = |S_L|/N$ and $p_R = |S_R|/N$. Therefore, the largest the Information Gain is, the better the partition.

This criterion works exactly the same as the Gini Gain, but replacing the Gini impurity by the Entropy.

A related criterion is the Gain Ratio, where the Information Gain of an attribute is normalized by that attribute's potential information. This is used as a way of decreasing the bias of the k-ary Information Gain criterion towards attributes with larger number of values. Since we are only interested in binary splits in this dissertation, we will not go into its details.

2.4.4 χ^2-criterion

The χ^2 is a popular criterion that was used in Mingers (1987). It is also the first one shown here not based on impurity measures, and it generates k-ary (instead of binary) splits. Is is mentioned here because of its relation to the framework that will be presented in Chapter 3.

The χ^2-criterion chooses the attribute A that maximizes

$$\sum_{i=1}^{n} \sum_{j=1}^{k} \frac{(A_{ij} - E[A_{ij}])^2}{E[A_{ij}]}, \tag{2-5}$$

where $E[A_{ij}] = N_i p_j$.

2.4.5 Conditional Inference Trees

Conditional Inference Trees are actually a framework for creating criteria that are free of bias on the number of values in an attribute. It was published

by Hothorn et al. (2006) and is a method of obtaining criteria that do not have any bias towards attributes with larger number of values.

It works by first choosing the best attribute to split at the current node and then evaluating all possible binary splits using any given impurity measure, picking the best one found.

To choose the attribute in which to split, first one has to calculate the permutation test's conditional expectation $\mu \in \mathbb{R}^{nk}$ and covariance $\Sigma \in \mathbb{R}^{nk \times nk}$ of every attribute[2]. Then, in order to compare the attributes, we need to calculate the p-value of a univariate test statistics c calculated on μ and Σ of every attribute. The only exact form of doing this comparison is by using the quadratic form c_{quad} (see equation 2-6), which follows an asymptotic χ^2 distribution with degrees of freedom given by the rank of Σ. Since this involves the calculation of a pseudoinverse, which has cubic complexity on the dimension of Σ, this criterion can be very time consuming.

$$c_{quad}(t, \mu, \Sigma) = (t - \mu)\Sigma^{+}(t - \mu)\top \tag{2-6}$$

This method, although very complicated and quite slow, is employed by the data mining community when accuracy counts for more than time spent training[3]. Thus this criterion will be used in our experiments in Chapter 5 to compare the different accuracies obtained when changing the splitting criterion used to choose the attribute's values split.

2.5 Heuristics for Splitting Decision Tree Nodes

As seen in the previous section, calculating the optimal binary split takes exponential time in the number of values or classes. Therefore many heuristics were created to construct decision trees in this situation. Some of the most used ones as well as some recent ones proposed by Laber et al. (2018) are listed below. All of them work with any impurity measure (e.g.: Gini or Entropy), but some of them work best with just one of them. When this is the case, it will be mentioned.

[2]Since the formulae are very complicated and won't be used throughout this dissertation, they are ommited. The interested reader can obtain them in the section 3 of Hothorn et al. (2006).

[3]Since this method does not have any bias when choosing which attribute to split, the accuracy of these trees tend to be higher than the trees obtained by biased criteria.

2.5.1 PC and PC-ext

These heuristics are based on the Principal Component of the contingency table and were presented in Coppersmith et al. (1999). They are based on a theorem that states that the optimal partition of values can be found by separating the class probability vectors $\pi(\mathbf{v})$ by a hyperplane, where the vector $\mathbf{v}$ denotes the row associated with the value v in the contingency table and $\pi(\mathbf{v}) = \mathbf{v}/\|\mathbf{v}\|_1$.

Theorem 8 *Hyperplanes lemma Let I be an impurity measure. If (L, R) is an optimal partition for the values V then there is a vector $\mathbf{d} \in \mathbb{R}^k$ such that*

$$\mathbf{d} \cdot \pi(\mathbf{v}) < 0, \forall v \in L$$

$$\mathbf{d} \cdot \pi(\mathbf{v}) > 0, \forall v \in R$$

where $\pi(\mathbf{v}) = \mathbf{v}/\|\mathbf{v}\|_1$.

In other words, in order to find the optimal partition we just need to choose the right hyperplane. This motivates the PC criterion, which look at the hyperplanes in $\mathbb{R}^k$ whose normal vector is the principal direction of the attribute's normalized contingency table.

In more details, one first calculates the class probability distribution of every value, which is done by normalizing the contingency table rows to measure 1 in the sum norm. Then the values are grouped into "supervalues" where all values in the same supervalue have the same class probability distribution. Now the first principal component of these supervalues' contingency table is calculated. One then calculates the inner product of each class probability vector of the supervalues with the principal component and sort the supervalues by it. Denote by $v_1, \ldots, v_{n^*}$ the n^* supervalues sorted in this manner and denote by $\mathbf{p}$ the first principal component. We then calculate the impurity gain of all the supervalues' splits of the form

$$L = \{v_i | \mathbf{v_i} \cdot \mathbf{p} \leq t\}$$

$$R = \{v_i | \mathbf{v_i} \cdot \mathbf{p} > t\}$$

where t is a chosen threshold. Once we find the supervalues split with the largest impurity gain among them, we choose it and translate the supervalues into original values to obtain a valid partition.

PC-ext is a simple extension of this algorithm, where instead of only testing the supervalues splits given by the equations above, we also test

the splits given by exchanging the last supervalue on the left with the first supervalue on the right (where first and last are given by the order after calculating the inner product).

These procedures require $O(k^3)$ operations to find the principal component and $O(n)$ impurity calculations and inner products in the class probability space.

2.5.2 SLIQ and SLIQ-ext

SLIQ was presented in Mehta et al. (1996) and it's a very simple greedy heuristic. Given an attribute, one starts with all the values going to the left split, and none on the right split. We then choose a value to go from the left to the right split. This value is the one that, when changing from the left to the right sides, decreases the impurity (increases the impurity gain) the most. This is repeated until there is no way of moving a value from the left to the right and decreasing the impurity.

SLIQ-ext is a simple extension, where we keep changing values from the left to the right until the left side is empty (that is, we move from the left to the right even if that increases the impurity). Once again the value to move is chosen in a greedy fashion. SLIQ-ext returns the values' split seen that had the lowest impurity.

Since Coppersmith et al. (1999) has experimental results that shows that PC obtains better partitions than SLIQ in general, we will not use this criterion in our experimental studies.

2.5.3 FlipFlop

The FlipFlop criterion, proposed in Nádas et al. (1991), uses the superclass trick from Twoing iteratively in the classes and values, until it converges to a solution. In order to explain in more detail, we first need to define the dual binary partition problem.

Definition 9 *[Dual partition problem]*

Given a binary partition problem and its contingency table, the dual partition problem is a problem where we want to find a (binary) partition of the classes that has the smallest impurity. This makes sense, since the impurity is calculated from the samples' split.

Mathematically, we just solve a binary partition problem in the transposed contingency table, treating the original values as the classes and the original classes as the values.

Algorithm 2 FlipFlop

Choose a random binary partition of values $V = (L_V, R_V)$
while True **do**
Calculate the supervalues contingency table by grouping the values that are in the same partition side of (L_V, R_V);

Solve the dual binary partition problem on the supervalues contingency table, finding the best partition of classes $C = (L_C, R_C)$;

Calculate the superclasses contingency table by grouping the classes that are in the same partition side of (L_C, R_C);

Solve the binary partition problem on the superclasses contingency table, finding the best partition of values $V = (L_V, R_V)$. If the new partition is the same as the previous one, return it.
end while

The FlipFlop is shown in Algorithm 2. Using Theorem 5, we see that each iteration inside the FlipFlop algorithm is fast. Nonetheless, there is no guarantee that the number of iterations is small. We only know, as proved in Nádas et al. (1991), that the values partition eventually converges. There is also no guarantee that the partition found is close to optimal.

Similarly to the SLIQ criterion, FlipFlop obtains worse results than PC in practice, as shown in Coppersmith et al. (1999). Thus, we will not use this criterion in our experimental studies. It is just shown here for completeness.

2.5.4 Hypercube Cover

The Hypercube cover criterion works exactly the same as the Twoing criterion except when it comes to the split impurity calculation. Instead of calculating the impurity based on the two superclasses, Hypercube Cover calculates it using the original k classes. This guarantees, for instance, that the split impurity, when measured with respect to the original classes, is never worse than that of Twoing. This method was first suggested in Laber et al. (2018), together with its approximation guarantee of 2 for both the Gini and Entropy impurities.

2.5.5 Largest Class Alone

This is a heuristic that, when using the Gini impurity, has an approximation guarantee of 2 compared with the optimal partition found by Gini Gain.

First one calculates the most frequent class and group the other classes in a single superclass. One then applies the Gini Gain criterion on this two-class problem. Since calculating the class frequencies can be done using the contingency table, this heuristic takes $O(N + nk + n \log n)$ time in total.

It can also be used with the Information Gain impurity, instead of Gini Gain, but its approximation guarantee increases to 3. These bounds are all proved in Laber et al. (2018).

2.5.6 List Scheduling

Similarly to the Largest Alone heuristic, List Scheduling is a criterion that has an approximation guarantee of 2 for the Informatio Gain criterion when using the Entropy impurity.

First one calculates the frequency of every class. Then, one groups the classes into 2 superclasses as balanced as possible in terms of number of samples. Lastly the Information Gain criterion is applied on this two-class problem.

Since finding the most balanced partition is NP-complete, we settle for a List Scheduling algorithm to find a partition with a 4/3-approximation guarantee to the optimal one. Since calculating the class frequencies can be done using the contingency table and the List Scheduling algorithm is linear in the number of classes, this takes $O(N + nk + n \log n)$ time in total. Note that, since we are not necessarily using the most balanced partition, there is no guarantee that we will be within a factor of 2 of the optimal impurity.

This criterion is based on a result proved in Laber et al. (2018) which states that, for the entropy impurity, the best form of grouping classes into superclasses is by balancing them the most.

3
Framework for Generating Splitting Criteria for Multi-valued Attributes

First we recall some definitions and results for the Max-Cut problem. These definitions will be used in the following section, when we define our framework.

3.1
The Maximum Weighted Cut Problem

We recall some definitions from graph theory. A cut X in a weighted graph $G = (V, E)$ is a subset of vertexes of V. The weight of a cut X, denoted here by $w(X)$, is the sum of the weights of the edges that have one endpoint in X and the other one in $V - X$.

The problem of computing the cut X^* with maximum weight in a graph with non-negative weights is NP-Hard. However, there are good approximation algorithms available. A remarkable one is the randomized algorithm proposed in Goemans & Williamson (1995) that relies on a formulation of the max-cut problem via semidefinite programming (SDP). This algorithm, denoted throughout this dissertation by GW, returns a cut X that satisfies $E[w(X)] \geq 0.878w(X^*)$. It involves solving an SDP on the graph weights matrix, calculating the Cholesky decomposition of it and then generating a random partition of the values based on the inner product of the decomposition column vectors with a randomly generated vector on the sphere of dimension n. As solving such an SDP takes $O(n^4)$ arithmetic operations (see Navascues et al. (2009)) and calculating the Cholesky decomposition takes $O(n^3)$ operations, the time complexity is high but polynomial.

Another possibility to solve the Max-Cut problem is by using the `GreedyCut` algorithm, presented in Algorithm 3. It obtains a cut X such that $w(X) \geq 0.5w(X^*)$, as proved in Sahni & Gonzalez (1976). The algorithm starts with two empty sets X and X'. Then, it scans the nodes and assigns each of them to the set that provides the maximum improvement on the weight of the current cut (ties are broken arbitrarily). Is it easy to see that the time complexity of this greedy algorithm is $O(n^2)$.

The solutions obtained by both GW and `GreedyCut` can be improved

Algorithm 3 GreedyCut(V: set of nodes)

$X \leftarrow \emptyset$;$X' \leftarrow \emptyset$
for $j = 1, .., n$ **do**
  If

$$\sum_{v \in X} w(v_i, v) > \sum_{v \in X' - V} w(v_i, v)$$

  add v_i to X' **Else** add v_i to X
end for
Return X and X'

Algorithm 4 LocalSearch(X, X'): set of nodes

label : *loop_start*
for $i = 1, ..., n$ **do**
  if switching v_i's side improves cut weight **then**
    switch v_i and update cut weight, X, X'
    goto loop_start
  end if
end for
for pair $(v_i, v_j) \in X \times X'$ **do**
  if switching v_i and v_j improves cut weight **then**
    switch v_i and v_j, update cut weight, X, X'
    goto loop_start
  end if
end for
Return X, X'

via a local search. In its simplest version, it moves a node from one group to the other while some improvement on the cut weight is possible. Although this algorithm is not polynomial in the worst case, it has polynomial behavior in the smoothed analysis framework (see Angel et al. (2017)). In addition, it is always possible to set a limit on the number of moves. A more refined version allows exchanging a pair of nodes as long as the weight of the cut is improved. In our experiments this is the version we use, as presented in Algorithm 4.

3.2 A Framework for Generating Splitting Criteria

In this section we explain our approach to building binary splitting criteria for multi-valued nominal attributes.

Let A be a nominal attribute that takes values in the domain $V = \{v_1, \ldots, v_n\}$. Our framework to produce a splitting criterion I consists of three steps:

1. Create a complete graph $G = (V, E)$ with n vertexes.

2. Assign a non-negative weight w_{ij} to the edge that connects v_i to v_j. This value shall reflect the benefit of putting v_i and v_j in different partitions. Different definitions of w_{ij} yield to different criteria, as we explain further.

3. Ideally, the value of the criterion I for attribute A is the weight of the cut with maximum weight in G. However, this is not a reasonable possibility for large n since, as mentioned before, the problem of computing the cut X^* with maximum weight in a graph with non-negative weights is NP-Hard. Thus, the value of criterion I is given by the weight of the cut obtained by some algorithm, with approximation guarantee, for the maximum cut problem in G.

What distinguishes the criteria generated by our framework is how the weights of the edges are set and what method is employed to compute the cut on graph G. Here, we discuss two ways to set the weights: the first one yields to a criterion that are related with the Gini Gain, while the second is built upon some given splitting criterion that works well for binary attributes.

3.2.1 The Squared Gini Criterion

Here, we discuss how to set the weights so that we obtain a criterion that can be seen as a variation of the Gini Gain discussed in Section 6.

In fact, Lemma 10 below show that it is possible to define the weights of the edges so that

$$w(S_L) = Gini(S) - p_L^2 \cdot Gini(S_L) - p_R^2 \cdot Gini(S_R) \tag{3-1}$$

for every partition (L, R) of V.

Note that the weight of the cut S_L in the above identity is similar to the expression for the Gini Gain given by equation (2-3). The difference is that p_L and p_R are replaced with p_L^2 and p_R^2, respectively. Because of the squares, this new criterion tends to favor more balanced partitions. Another practical observation is that one can define the edges without the $2/N^2$ term in 3-1 and find the same maximum cut, since this constant appears in all edges weights.

For the proof of Lemma 10, recall that A_{ix} is the number of samples of class x that have value v_i, and that C is used to denote the set of classes.

Lemma 10 *For every i, j, with $i \neq j$ and $i, j \in \{1, \ldots, n\}$, let*

$$w_{ij} = \frac{2 \sum_{\substack{x,y \in C \\ x \neq y}} A_{ix} A_{jy}}{N^2} \tag{3-2}$$

Then, for every partition (L, R) *of* V *we have*

$$w(S_L) = Gini(S) - p_L^2 \cdot Gini(S_L) - p_R^2 \cdot Gini(S_R)$$

Proof: Let $S_{x,L}$ and $S_{y,R}$ be the number of samples of classes x and y in groups L and R, respectively. Moreover, let N_L and N_R be the number of samples in L and R, respectively. It follows from equation (2-1) that

$$N^2 Gini(S) = N^2 - \sum_{x=1}^{k} (S_{x,L} + S_{x,R})^2$$

$$N_L^2 Gini(S_L) = N_L^2 - \sum_{x=1}^{k} S_{x,L}^2$$

and

$$N_R^2 Gini(S_R) = N_R^2 - \sum_{x=1}^{k} S_{x,L}^2.$$

Since $N = (N_L + N_R)$ it follows that

$$N^2 Gini(S) - N_L^2 Gini(S_L) - N_R^2 Gini(S_R) =$$

$$2N_L N_R - 2 \sum_{x \in C} S_{x,L} S_{x,R} =$$

$$2 \sum_{x \in C} S_{x,L} \sum_{x \in C} S_{x,R} - 2 \sum_{x \in C} S_{x,L} S_{x,R} =$$

$$2 \sum_{\substack{x \neq y \\ x,y \in C}} S_{x,L} S_{y,R} = 2 \sum_{\substack{x \neq y \\ x,y \in C}} \left(\sum_{i \in L} \sum_{j \in R} A_{ix} A_{jy} \right) =$$

$$N^2 \sum_{i \in L} \sum_{j \in R} w_{ij} = N^2 w(S_L)$$

Dividing the first term and the last term by N^2 in the above expression and, using $N_L = p_L \cdot N$ and $N_R = p_R \cdot N$, we establish the lemma. ■

It is worth mentioning that symmetric misclassification costs can be easily introduced in this case. In fact, let $mix(x, y)$ be the cost of mixing samples from classes x and y. We can define

$$w_{ij} = \sum_{\substack{x,y \in C \\ x \neq y}} mix(x, y) p_{ix} p_{jy}.$$

This measure favors the separation of the classes that incur a large cost in the

case they are mixed.

Another natural question is whether the maximum weighted cut problem, which is NP-complete in general, is not easier for the case where the weights are set by equation 3-2. The answer is no, as proved in the theorem below.

Theorem 11 *Finding the maximum weighted cut in a complete graph whose weights are set by equation 3-2 is NP-complete.*

Proof: The idea is to show a reduction from the NP-complete problem PARTITION using the fact that, for some specific instances of our problem, the optimal partition is the most balanced one.

Recall that, in the PARTITION problem, we are given a multiset of integers and want to decide whether it can be partitioned into two multisets with the same sum of elements. Consider an instance given by a multiset $U = \{u_1, \ldots, u_k\}$ of integers. Create an instance of our decision tree problem as follows: for each $u_i \in U$ add a value v_i whose row in the contingency table is given by $u_i e_i$. In other words, in this instance every value has a single class and every class appears for a single value.

W.l.o.g. we can ignore the $2/N^2$ factors in 3-2. Hence every edge w_{ij} in the associated Squared Gini graph will have weight equal to ij. Therefore any cut partitioning V into (L, R) will have value equal to

$$cut(L, R) = \Big(\sum_{i|v_i \in L} u_i \Big) \cdot \Big(\sum_{i|v_i \in R} u_i \Big)$$

Note that this formula is maximized when the two terms are as close as possible, i.e. the two partition sides are as balanced as possible. Thus, if we can find a polynomial time algorithm that finds the best partition of this instance's values, we can solve the associated PARTITION problem in polynomial time. This concludes the reduction. ■

3.2.2 Setting weights according to other splitting criteria

Our second way of defining the weights makes use of some given splitting criterion for binary nominal attributes. Such criterion is used to measure the quality of separating samples with value v_i from those with value v_j, for each i and j, and, thus, defining the edges' weights. Here, we investigate the criterion obtained by defining w_{ij} as the value of the χ^2-test for the attribute A when evaluated over the restricted dataset that contains only the samples of S with

values v_i and v_j:

$$w_{ij} = \sum_{\ell=1}^{k} \frac{(A_{i\ell} - E[A_{i\ell}])^2}{E[A_{i\ell}]} + \sum_{\ell=1}^{k} \frac{(A_{j\ell} - E[A_{j\ell}])^2}{E[A_{j\ell}]}$$

where $E[A_{i\ell}] = N_i p_\ell$ and $E[A_{j\ell}] = N_j p_\ell$.

In order to reduce the bias towards attributes with many values, we divide w_{ij} by $n - 1$, for every pair (i, j). We make this adjustment because each value contributes to the weight of $n - 1$ edges.

We shall remark that, although not explored in this work, other criteria such as Information Gain or Gini Index could be used, instead of the χ^2-test, to set the weights.

3.2.3 Example

Let's assume we are using the Squared Gini criterion on a dataset with a single attribute (the marital status) and we try to predict whether it's a male or female. The contingency table and associated Squared Gini graph are given in Figure 3.1.

As an illustration of the calculation of the edges' weights, we show below how they are done for the Single-Married edge. We number the values and classes in the order shown in the contingency table in Figure 3.1. In particular, Single and Married are values number 1 and 2, respectively; while Male and Female are classes number 1 and 2 respectively. Thus

$$w_{12} = \frac{2}{N^2} \sum_{\substack{x,y \in C \\ x \neq y}} A_{1x} A_{2y} = \frac{2}{(237)^2}(18 \cdot 30 + 20 \cdot 29) = 0.0399$$

Since this example is small (there are only 7 non-empty distinct cuts), we can run through all the possible cuts and see that the one shown in red is actually the maximum one. In larger examples we need to use approximated algorithms such as the one mentioned before, since the number of possible cuts grows exponentially with the number of nodes/values.

3.2.4 Handling Numeric Attributes

We observe that criteria from our framework can handle a numeric attribute A with t distinct values $v_1, \ldots, v_t$ by considering it as collection of $t - 1$ binary attributes, where the j-th attribute, A^j, splits the samples into the groups $\{s|A(s) \leq v_j\}$ and $\{s|A(s) > v_j\}$. The split obtained by criterion

Value \ Class	Male	Female	Total
Single	18	20	38
Married	29	30	59
Divorced	53	51	104
Widowed	10	26	36
Total	110	127	237

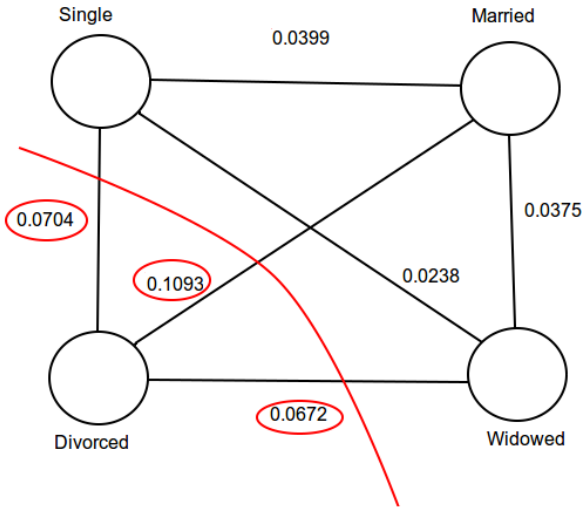

Figure 3.1: Contingency table, associated Squared Gini graph and maximum cut.

I on a numeric attribute A matches the split of the best attribute among $A^1, \ldots, A^{t-1}$, according to I.

4
Experiments on Splits with Reduced Impurity

In this chapter we compare the ability of different heuristics in finding the values' split with lowest impurity. We are interested in choosing what heuristic/criterion to use when the exact ones don't run in reasonable time. In the next chapter the performance of the best heuristics will be compared on real datasets against Twoing and criteria generated from our framework.

Our experiments are very similar to those proposed in Coppersmith et al. (1999) except for a few details. All experiments are Monte Carlo simulations with 10,000 runs, each using a randomly-generated contingency table for the given number of values n and classes k. Each table was created by uniformly picking a number in $\{0, \ldots, 7\}$ for every entry. This guarantees a substantial probability of a row/column having some zero frequencies, which is common in practice. Differing from Coppersmith et al. (1999), if all the entries corresponding to a value or a class are zero, we re-generate the contingency table, otherwise the number of actual values and classes would not match n and k. We evaluated Hypercube Cover, PC-ext, Largest Class Alone and List Scheduling for both the Gini and Entropy impurities. Note that we don't evaluate Twoing because the split selected by Hypercube Cover is always purer by construction. Moreover, it is also more natural, since we are measuring the performance of these criteria on the k-class impurity.

Tables 4.1 and 4.2 show, for different values of n and k, the percentage of times that each criterion found the best Gini and Entropy impurities, respectively. Note that the percentages do not necessarily sum exactly 100% since there were ties. In these tables we only show results for $k \leq 9$ because for larger values of k Hypercube Cover becomes non-practical due to its running time. In addition, we do not present results for small values of n because in this case the optimal partition can be found quickly by testing all possible partitions, so that there is no motivation for heuristics.

In general, we observe an advantage for Hypercube Cover with both impurities, being more clear for the Entropy, followed by PC-ext. This suggests that the Largest Class Alone and List Scheduling heuristics are not competitive with them in terms of split impurity found.

Another possible comparison between them is to see what happens

n	k	HcC	PC-ext	LCA	LS
12	3	97.3	91.2	42.8	42.8
	5	99.2	88.0	19.1	17.8
	7	99.9	86.6	11.5	10.8
	9	100	85.0	8.5	8.4
25	3	73.9	72.7	24.3	24.3
	5	65.8	62.4	5.9	4.0
	7	73.3	58.8	2.0	1.7
	9	85.3	53.2	0.8	0.9
50	3	51.4	50.6	16.0	16.0
	5	33.1	41.1	3.3	1.3
	7	31.0	40.7	1.0	0.4
	9	33.9	37.8	0.4	0.1

Table 4.1: Percentage of times each criterion finds the smallest Gini impurity, compared among themselves.

n	k	HcC	PC-ext	LCA	LS
12	3	98.3	80.2	33.5	33.5
	5	99.4	74.2	13.6	15.3
	7	100	73.2	8.3	10.1
	9	100	72.4	6.8	8.0
25	3	83.3	54.4	18.5	18.5
	5	76.9	42.7	5.3	2.6
	7	81.0	39.2	2.0	1.2
	9	87.7	37.7	1.3	0.8
50	3	70.0	29.5	13.4	13.4
	5	57.4	22.0	3.7	0.6
	7	53.5	21.7	1.6	0.1
	9	52.5	22.1	0.8	0.1

Table 4.2: Percentage of times each criterion finds the smallest Entropy impurity, compared among themselves.

when Hypercube Cover and PC-ext find different partitions. To measure this difference, let us define the relative excess (in percentage) of a partition P w.r.t. a partition Q as $100 \times (I(P)/I(Q) - 1)$. The results of this measurement are shown in Tables 4.3 and 4.4. For the Gini impurity (Table 4.3), we can observe that Hypercube Cover finds partitions closer to the optimal than PC-ext. This behavior is even stronger for the Entropy impurity (Table 4.4), where both the average and maximum relative excess of Hypercube Cover over PC-ext is much smaller than PC-ext's over Hypercube Cover. These numbers suggest that the risk of finding a "bad" partition is smaller when Hypercube Cover is used, specially for the Entropy impurity.

Lastly, we note that, due to Largest Class Alone and List Scheduling running times, they might be used when both n and k are very large and

n	k	HcC excess over PC-ext		PC-ext excess over HcC	
		Average	Max	Average	Max
12	3	0.15	0.97	0.37	2.36
	5	0.06	0.22	0.14	0.98
	7	0.02	0.05	0.08	0.49
	9	—	—	0.05	0.36
25	3	0.14	0.83	0.24	1.72
	5	0.05	0.29	0.1	0.84
	7	0.32	1.84	0.32	1.62
	9	0.19	1.06	0.2	1.16
50	3	1.35	7.27	1.37	8.66
	5	0.39	2.02	0.38	2.1
	7	0.19	0.95	0.18	1.11
	9	0.11	0.62	0.11	0.53

Table 4.3: Relative excess impurity, in percentage, for experiments where Hypercube Cover and PC-ext found different partitions using the Gini impurity.

n	k	HcC excess over PC-ext		PC-ext excess over HcC	
		Average	Max	Average	Max
12	3	0.17	1.07	0.77	6.59
	5	0.09	0.39	0.37	2.64
	7	0.04	0.04	0.24	2.11
	9	—	—	0.18	1.42
25	3	0.14	0.81	0.5	4.26
	5	0.08	0.49	0.26	2.02
	7	0.53	3.23	0.58	3.12
	9	0.38	2.11	0.42	2.31
50	3	1.33	7.87	1.46	7.31
	5	0.5	2.69	0.58	3.07
	7	0.29	1.5	0.35	1.85
	9	0.21	1.2	0.25	1.25

Table 4.4: Relative excess impurity, in percentage, for experiments where Hypercube Cover and PC-ext found different partitions using the Entropy impurity.

speed is an issue. When $n = 200$ and $k = 100$, using a single core, they are almost 50 times faster than PC-ext, with the latter using 8 cores. In addition, they could be used together with PC-ext, incurring a negligible overhead, to guarantee that the ratio between the impurity of the partition found and the optimal one is bounded.

Taking into account these experiments, those reported in Coppersmith et al. (1999) and the theoretical properties of the available algorithms, Table 4.5 suggests guidelines on which criterion to use to solve the problem of finding the binary partition of minimum impurity in practical situation. Of course small, medium and large depend on the available hardware and the time one accepts to wait for training/testing classification models.

n	**k**	**Suggested Method**
small	any	Exact
not small	small	Hypercube Cover
not small	not small	PC-ext

Table 4.5: Guidelines on how to solve the problem of finding the partition with minimum impurity in practice.

5
Experiments on Real Datasets

In this chapter we describe our experimental study on real datasets. First, we describe the chosen datasets. Next, we discuss the max-cut algorithms employed and, then, we present our results. We will compare the performance between Twoing, Hypercube Cover, PC-ext and some criteria generated from our framework. Both Hypercube Cover and PC-ext were chosen based on the experiments results of the previous chapter. We will use the Gini impurity with them throughout this chapter.

All experiments described in the following sections were executed on a machine with the following settings: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz with 32 GB of RAM. The code was developed using Python 3.6.1 with the libraries numpy, scipy, scikit-learn and cvxpy. The project can be accessed in `github.com/felipeamp/dissertation-code`.

TODO: tornar repositorio publico e adicionar os resultados agregados.

5.1
Datasets

We employed 11 datasets in total. Eight of them are from the UCI repository: Mushroom, KDD98, Adult, Nursery, Covertype, Cars, Contraceptive and Poker (Asuncion & Newman (2007)). Two others are available in Kaggle: San Francisco Crime and Shelter Animal Outcome SF-OpenData (2015); Austin Animal Center (2016). The last dataset was created by translating texts from the Reuters database Asuncion & Newman (2007) into phonemes, using the CMU pronouncing dictionary (see Weide (1998)).

We chose these datasets because they have at least 1000 samples and they either contain multi-valued attributes or attributes that can be naturally aggregated to produce multi-valued attributes. From the KDD98 dataset we derived the datasets KDD98-k, for $k = 2$ and 9. These datasets contain only the positive samples (people that donate money) of KDD98 and the target attribute, Target_D, is split into k classes, where the i-th class correspond to the i-th quantile in terms of amount of money donated. For the Reuters Phonemes dataset, we extracted 10000 samples containing the 15 most common phonemes as class and try to predict when they are about to happen given

Dataset	Samples	k	m_{nom}	m_{nom}^{ext}	m_{num}
`Mushroom`	5644	2	22	N/A	0
`Adult`	30162	2	8	N/A	6
`KDD98`	4843	`Reg`	65	N/A	314
`Nursery`	12960	5	8	11	0
`CoverType`	581012	7	44	46	10
`Car`	1728	4	6	8	0
`Contracep`	1473	3	7	9	2
`Poker`	25010	10	10	0	0
`Shelter`	26711	22	5	N/A	1
`S.F. Crime`	878049	39	3	N/A	2
`Phonemes`	10000	15	3	N/A	0

Table 5.1: Information about the employed datasets after data cleaning and attributes aggregation. Column k is the number of classes and `Reg` stands for Regression; columns m_{nom} and m_{nom}^{ext} are the number of nominal attributes in the original and the extended datasets (when it exists), respectively; column m_{num} is the number of numeric attributes.

the 3 preceding phonemes. This dataset is motivated by Spoken Language Recognition problems, where phonotactic models are used as an important part of the classification system, as seen in Navratil (2006). For the San Francisco Crime dataset, we try to predict the crime category given the month, day of the week, police department district and latitute/longitude. Lastly, for the Shelter Animal Outcomes dataset, we converted the age into a numeric field containing the number of days old and separated the breed into two categorical fields, repeating the breed in both in case there was only one originally. We also removed the AnimalID, Name and the DateTime. For this dataset we try to predict the outcome type and subtype (concatenated into a single categorical field). For both San Francisco Crime and Shelter Animal Outcomes datasets we created a version of them (`S.F. Crime-15` and `Shelter-15`), containing only 15 classes, instead of the 39 and 22 original ones, respectively. This was done by grouping the rarest classes into a single one.

We also created extended versions of some of the above datasets by adding nominal attributes obtained by aggregating some of the original ones, as we detail below. Our goals are examining the impact of multi-valued attributes in the classification performance and also understanding how the different splitting criteria handle them.

Table 5.2 illustrates this construction.

– `Nursery-Ext`. This dataset is obtained by adding three new attributes to dataset Nursery. The first attribute has 15 distinct values and it is constructed through the aggregation of 2 attributes from group EMPLOY,

parents	has_nurs	Aggregated Attribute
usual	proper	usual-proper
usual	less_proper	usual-less_proper
usual	improper	usual-improper
usual	critical	usual-critical
usual	very_crit	usual-very_crit
pretentious	proper	pretentious-proper
pretentious	less_proper	pretentious-less_proper
pretentious	improper	pretentious-improper
pretentious	critical	pretentious-critical
pretentious	very_crit	pretentious-very_crit
great_pret	proper	great_pret-proper
great_pret	less_proper	great_pret-less_proper
great_pret	improper	great_pret-improper
great_pret	critical	great_pret-critical
great_pret	very_crit	great_pret-very_crit

Table 5.2: Aggregation of attributes parents and has_nurse from dataset `Nursery`.

one with 5 values and the other with 3 values. The second attribute has 72 distinct values corresponding to the aggregation of attributes from the attributes in group `STRUCT_FINAN`. The third attribute, with 9 distinct values, is the combination of the attributes in group `SOC_HEALTH`.

- `Covertype-Ext`. We combined 40 binary attributes related with the soil type into a new attribute with 40 distinct values. The same approach was employed to combine the 4 binary attributes related with the wilderness area into a new attribute with 4 distinct values. This is an interesting case because, apparently, the 40 (soil type) binary attributes as well as the 4 (wilderness area) binary attributes were derived from a binarization of two attributes, one with 40 distinct value and the other with 4 distinct values.
- `Cars-Ext`. To obtain this dataset, the 2 attributes related with the concept `PRICE`, `buying` and `maint`, were combined into an attribute with 16 distinct values. Moreover, the 3 attributes related with concept `CONFORT` were combined into an an attribute with 36 distinct values.
- `Contraceptive-Ext`. The 2 attributes related with the couple's education were combined into an attribute with 16 distinct values. Moreover, the 3 attributes related with the couple's occupations and standard of living were aggregated into a new attribute with 32 distinct values.

Samples with missing values were removed from the datasets. Table 5.1 provides some statistics.

5.2 Computing the Maximum Cut

The GW algorithm requires the solution of a semidefinite program (SDP), which may be computationally expensive despite its polynomial time worst case behavior. As an example, for an attribute with 100 distinct values, the solution of the corresponding SDP takes in average about 2 second in our machine. One the one hand, this is a tiny amount of time compared with that required to perform an exhaustive search on the 2^{100} possible binary partitions. On the other hand, faster alternatives are desirable, even at the cost of losing part of the theoretical approximation guarantee.

To avoid solving a SDP, we also evaluated a procedure that first executes the `GreedyCut` algorithm presented in Section 3.1 and then runs a local search as described in the Algorithm 4 in the same section. The use of this approach combined with the two ways of setting the edges' weights lead to Greedy LocalSearch SquaredGini (GLSG) and Greedy LocalSearch χ^2 (GLχ^2) criteria, respectively. For attributes with 100 distinct values this approach is 60-70 times faster than the one based on the GW algorithm.

Even though solving the SDP takes polynomial time, the GW criteria evaluated in this chapter were too slow compared with the others. Therefore their results won't be shown in the following Sections except for the Section for maximum depth 5 (5.3.2), which were the first experiments to be run.

5.3 Experimental Results

We performed a number of experiments to evaluate how the proposed methods behave with real datasets. All experiments consist of building decision trees with a predefined maximum depth. In addition, to prevent the selection of non-informative nominal attributes, we used a χ^2-test for each attribute at every node of the tree: if the χ^2-test on the contingency table of attribute A has p-value larger than 10% at a node ν, then A is not used in ν. Furthermore, attributes with less than 15 samples associated with its second most frequent value are also not considered for splitting. This helps avoid data overfitting.

5.3.1 Maximum Depth 1

Table 5.3 presents the results of an experiment to compare the accuracy of Decision Trees built by our methods with those built by Twoing, Hypercube Cover and PC-ext. In this experiment, the decision tree can have a single level and we considered just the nominal attributes of the datasets. The motivation

Dataset	Twoing		GLSG		GLχ^2		PC-ext		HcC	
Adult	75.11	(0)	75.11	(0)	75.11	(0)	75.11	(0)	75.11	(0)
Mushroom	98.44	(0)	98.44	(0)	98.44	(0)	98.44	(0)	98.44	(0)
KDD98-2	**83.61**	**(1)**	83.3	(0)	**83.61**	**(1)**	**83.61**	**(1)**	**83.61**	**(1)**
KDD98-9	31.66	(0)	32.32	**(4)**	31.61	(0)	31.74	(2)	31.74	(2)
Nursery	66.25	(0)	66.25	(0)	66.25	(0)	66.25	(0)	66.25	(0)
Nursery-ext	66.25	(0)	66.25	(0)	66.25	(0)	66.25	(0)	66.25	(0)
Cars	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)
Cars-ext	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)
Contracep	43.39	**(2)**	42.31	(0)	42.23	(0)	43.63	**(2)**	43.63	**(2)**
Contracep-ext	42.58	(1)	**44.29**	**(4)**	42.23	(0)	43.73	(3)	43.38	(2)
CoverType	**51.93**	**(1)**	48.76	(0)	**51.93**	**(1)**	**51.93**	**(1)**	**51.93**	**(1)**
CoverType-ext	51.54	(0)	55.55	(3)	51.93	(2)	**58.64**	**(4)**	51.54	(0)
Poker	49.93	(0)	**49.95**	(0)	**49.95**	(0)	49.93	(0)	49.93	(0)
Shelter-15	**48.44**	**(2)**	35.39	(0)	40.1	(1)	**48.44**	**(2)**	**48.44**	**(2)**
S.F. Crime-15	**21.61**	**(2)**	21.24	(1)	19.98	(0)	21.58	**(2)**	21.58	**(2)**
Phonemes	22.21	(0)	22.22	(0)	22.57	(2)	**23.92**	**(3)**	23.91	**(3)**
Average (Sum)	55.81	(9)	55.09	(12)	55.14	(7)	56.45	(20)	55.99	(15)

Table 5.3: Average accuracy and statistical tests for decision trees with depth at most 1 using only nominal attributes. The best accuracy for each dataset is bold-faced.

for this depth is that some random forests methods such as boosting work with very shallow trees. Each accuracy is the average of 20 stratified 3-fold cross-validations, each generated with a different seed. The entry associated with $(\mathcal{D}, I)$ has two pieces of information: the average accuracy of criterion I on dataset $\mathcal{D}$ and the number of criteria with accuracy statistically lower than that of I on dataset $\mathcal{D}$. The statistical test used for criteria comparison is a one-tailed paired t-student test with a 95% confidence level.

In general, there was a clear advantage towards PC-ext, followed by Hypercube Cover and GL Squared Gini. Twoing and GLχ^2had, in general, the worse performance. As expected, the criteria created the same tree for many datasets, and this happened more frequently when the number of classes was smaller.

Table 5.4 presents the comparison between our GL methods, Twoing, Hypercube Cover and PC-ext in another scenario, where we use c_{quad}, one of the bias-free criterion proposed in Hothorn et al. (2006), to select the attribute at each node of the tree. Then, both Twoing, Hypercube Cover, PC-ext and our methods are used only for splitting the chosen attribute, which allows for a more direct comparison of their splitting ability. Since the maximum depth is set to 1, all the criteria are splitting the same attribute. This makes the analysis of their splitting ability more direct than with larger depths. This experiment is very similar to the one from the previous chapter, except now

Dataset	CI-Twoing		CI-GLSG		CI-GLχ^2		CI-PC-ext		CI-HcC	
Adult	**75.13**	**(2)**	75.11	(0)	75.11	(0)	**75.13**	**(2)**	**75.13**	**(2)**
Mushroom	63.88	(1)	**68.03**	**(4)**	66.5	(3)	63.72	(0)	63.88	(1)
KDD98-2	65.29	(0)	65.29	(0)	65.29	(0)	65.29	(0)	65.29	(0)
KDD98-9	25.48	(0)	25.48	(0)	25.48	(0)	25.48	(0)	25.48	(0)
Nursery	**41.9**	**(1)**	41.05	(0)	**41.9**	**(1)**	**41.9**	**(1)**	**41.9**	**(1)**
Nursery-Ext	**41.9**	**(1)**	41.05	(0)	**41.9**	**(1)**	**41.9**	**(1)**	**41.9**	**(1)**
Cars	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)
Cars-Ext	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)	70.02	(0)
Contracep	43.39	**(2)**	42.66	(1)	42.23	(0)	**43.63**	**(2)**	**43.63**	**(2)**
Contracep-Ext	43.39	**(2)**	42.66	(1)	42.23	(0)	**43.63**	**(2)**	**43.63**	**(2)**
CoverType	48.76	(0)	48.76	(0)	48.76	(0)	48.76	(0)	48.76	(0)
CoverType-Ext	48.76	(0)	48.76	(0)	48.76	(0)	48.76	(0)	48.76	(0)
Poker	49.93	(0)	49.95	(0)	49.95	(0)	49.93	(0)	49.93	(0)
Shelter-15	35.93	(0)	35.93	(0)	35.93	(0)	35.93	(0)	35.93	(0)
S.F. Crime-15	19.92	(0)	19.92	(0)	19.92	(0)	19.92	(0)	19.92	(0)
Phonemes	**15.7**	**(4)**	15.34	(0)	15.42	(0)	15.46	(1)	15.46	(1)
Average (Sum)	47.46	(13)	47.5	(6)	47.46	(5)	47.47	(9)	47.48	(10)

Table 5.4: Average accuracy and statistical tests for Conditional Inference trees with depth at most 1 using only nominal attributes. The best accuracy for each dataset is bold-faced.

we are using real datasets.

As can be seen, they all obtained similar results. In terms of beating other criteria, Twoing obtained the best results, followed by Hypercube Cover. Ignoring the extended datasets, since no extended attribute was chosen by the conditional inference tree and thus its results are a repetition of the datasets without extended attributes, we note that PC-ext and the GL criteria obtained similar results. This suggests that Twoing and Hypercube Cover are better at choosing the best values partition, once the attribute is fixed. Nonetheless, since GLSG had significantly better results in the Mushroom dataset, it had the best average accuracy.

Table 5.5 shows experiments similar to those presented at Table 5.3, but now using also the numeric attributes. This time, in terms of beating the accuracy of other criteria, the GL methods obtained the best results, while Twoing had the worst. This suggests that criteria from our framework are better in terms of comparing numerical and nominal attributes among themselves. Considering only the average accuracy, PC-ext and Hypercube Cover had the best results, mainly due to the accuracies found for both CoverType datasets.

Dataset	Twoing		GLSG		GLχ^2		PC-ext		HcC	
`Adult`	75.11	(0)	75.11	(0)	**79.55**	**(4)**	75.11	(0)	75.11	(0)
`KDD98-2`	83.61	(0)	**83.93**	**(4)**	83.67	(1)	83.62	(0)	83.62	(0)
`KDD98-9`	31.66	(0)	**32.52**	**(3)**	32.46	**(3)**	31.44	(0)	31.44	(0)
`Contracep`	42.84	(1)	42.21	(0)	**45.19**	**(4)**	42.7	(1)	42.7	(1)
`Contracep-Ext`	42.39	(0)	44.29	(3)	**45.19**	**(4)**	42.68	(0)	42.59	(0)
`CoverType`	51.93	(0)	62	(2)	52.38	(1)	**63.36**	**(3)**	**63.36**	**(3)**
`CoverType-Ext`	51.54	(0)	62	(2)	52.38	(1)	**63.36**	**(3)**	**63.36**	**(3)**
`Shelter-15`	**48.44**	**(2)**	35.4	(0)	43.29	(1)	**48.44**	**(2)**	**48.44**	**(2)**
`S.F. Crime-15`	**21.61**	**(2)**	21.24	(1)	20.61	(0)	21.58	**(2)**	21.58	**(2)**
Average (Sum)	49.9	(5)	50.97	(15)	50.52	(19)	52.48	(11)	52.47	(11)

Table 5.5: Average accuracy and statistical test results for Decision Trees using both nominal and numeric attributes.

5.3.2 Maximum Depth 5

Table 5.6 presents the results of an experiment to compare the accuracy of Decision Trees built by our methods with those built by Twoing, Hypercube Cover and PC-ext. In this experiment, the maximum depth was set to 5 and we considered just the nominal attributes of the datasets. The motivation for this depth is to produce trees that are relatively easy to interpret and, in addition, some random forests methods such as boosting work with shallow trees. Each accuracy is the average of 20 stratified 3-fold cross-validations, each generated with a different seed. The entry associated with $(\mathcal{D}, I)$ has two pieces of information: the average accuracy of criterion I on dataset $\mathcal{D}$ and the number of criteria with accuracy statistically lower than that of I on dataset $\mathcal{D}$. The statistical test used for criteria comparison is a one-tailed paired t-student test with a 95% confidence level.

In general, there was a clear advantage towards PC-ext and a clear disadvantage towards the GW-based criteria. Twoing, Hypercube Cover and the GL criteria had somewhat similar results. The advantage of the GL-based criteria over the GW-based ones is likely related with the fact that the weights of the cuts computed by the GL approach in this experiment are, in general, larger than those obtained by the GW algorithm.

The results of Table 5.6 also provide evidence of the potential of considering aggregated attributes. The accuracy obtained for the extended versions of datasets `Nursery`, `Cars` and `CoverType` are considerably higher than those obtained for the original versions. For `Contracep`, the effect is not clear.

Another key aspect to discuss is the computational cost of the proposed criteria. Table 5.7 shows the running time of each criterion in the experiment of Table 5.6. Twoing is the fastest method when the number of classes is small

Dataset	Twoing		GWSG		GWχ^2		GLSG		GLχ^2		PC-ext		HcC	
Adult	82.21	(2)	81.91	(1)	82.24	(2)	81.83	(0)	82.24	(2)	**82.31**	**(6)**	82.21	(2)
Mushroom	**100**	**(2)**	99.99	(0)	99.98	(0)	100	**(2)**	99.99	(0)	**100**	**(2)**	**100**	**(2)**
KDD98-2	80.47	(0)	81	(3)	80.74	(1)	81.16	(4)	80.51	(0)	**81.25**	**(5)**	80.47	(0)
KDD98-9	40.35	(2)	38.13	(0)	40.93	**(5)**	38.15	(0)	**41**	**(5)**	40.27	(2)	40.14	(2)
Nursery	88.25	**(3)**	88.03	(0)	88.2	(0)	**88.33**	**(3)**	88.2	(0)	88.25	**(3)**	88.25	**(3)**
Nursery-Ext	**93.82**	**(4)**	90.95	(0)	93.02	(2)	90.75	(0)	93.13	(2)	93.81	**(4)**	93.81	**(4)**
Cars	86.53	(1)	85.39	(0)	86.37	(1)	**87.93**	**(6)**	86.42	(1)	86.5	(1)	86.5	(1)
Cars-Ext	90.3	(0)	90.82	(3)	91.57	(5)	90.84	(3)	**91.9**	**(6)**	90.32	(0)	90.32	(0)
Contracep	43.77	(0)	43.96	(0)	**44.04**	**(2)**	43.89	(0)	44	(0)	43.59	(0)	43.62	(0)
Contracep-Ext	43.17	(0)	44.32	**(5)**	43.44	(0)	**44.35**	**(5)**	43.7	(0)	43.77	(2)	43.36	(0)
CoverType	52.97	(2)	55.05	(3)	51.07	(0)	55.05	(3)	51.07	(0)	**58.12**	**(5)**	**58.12**	**(5)**
CoverType-Ext	64.48	(4)	64.12	(2)	57.73	(0)	64.23	(3)	59.95	(1)	**64.71**	**(6)**	64.54	(5)
Poker	51.9	(4)	50.28	(1)	51.77	(2)	49.94	(0)	**51.92**	**(5)**	51.7	(2)	51.69	(2)
Shelter-15	48.01	(3)	—	(0)	—	(0)	45.31	(2)	**48.13**	**(4)**	48.07	(3)	48.05	(3)
S.F. Crime-15	**22.1**	**(2)**	21.32	(1)	22.09	**(2)**	21.23	(0)	22.09	**(2)**	22.09	**(2)**	22.09	**(2)**
Phonemes	**30.92**	**(6)**	—	(0)	—	(0)	30.29	(4)	29.47	(2)	30.59	(5)	29.92	(3)
Average (Sum)	63.7	(35)	66.8*	(19)	66.66*	(22)	63.33	(35)	63.36	(30)	64.08	(48)	63.94	(34)

Table 5.6: Average accuracy and statistical tests for decision trees with depth at most 5 using only nominal attributes. The best accuracy for each dataset is bold-faced. Experiments that did not finish in reasonable time are considered statistically worse than the others. These criteria have a * mark besides their average accuracies, since they are calculated only on the experiments that finished.

and the GL-based methods and PC-ext become competitive and eventually the fastest ones when the number of classes gets larger. As the number of classes increases, the GL-based methods and PC-ext become much faster than Twoing, with the turning point being around $k = 7$. For datasets with 15 classes our GL criteria are 15-300 times faster. The same can be seen for PC-ext, which is 15-600 times faster. We also ran experiments using all the classes available in both the `S.F. Crime` and `Shelter` datasets (39 and 22, respectively). Twoing and the GW criteria can not be executed in a reasonable time with that many classes, while GLSG and GLχ^2 ran in approximately 100 seconds on the `S.F. Crime` dataset and 300 seconds on the `Shelter` dataset (PC-ext ran in 75 and 32 seconds, respectively). This behavior for the Twoing criterion is not surprising, since its running time has an exponential dependence of the number of classes k. Nonetheless, since the execution time for our GL criteria in this experiment grew in an approximately linear fashion with k, it suggests that they can also be used with datasets that have a much larger number of classes. It is also interesting to note that the aggregated attributes usually appeared at or near the root of the decision trees. Lastly, the running time for the GW-based criteria were usually one or two orders of magnitude larger than the others. The only clear exception was in the CoverType dataset, where the number of samples is very large while the attributes' number of values is much smaller. Hypercube Cover behaves very similarly to the Twoing criterion, but it is a bit slower because the impurity calculation with k classes is slower than with 2 classes.

Since the GW-criteria performe much worse in terms of execution time and yield comparable or worse accuracy than the other criteria, they were not used in any other experiments.

Table 5.8 presents the comparison between our GL methods, Twoing, Hypercube Cover and PC-ext in another scenario, where we use c_{quad}, one of the bias-free criterion proposed in Hothorn et al. (2006), to select the attribute at each node of the tree. Then, both Twoing, Hypercube Cover, PC-ext and our methods are used only for splitting the chosen attribute, which allows for a more direct comparison of their splitting ability. Once again we observed a balance between most of the criteria, with a significant advantage towards GL-χ^2 and, perhaps surprisingly, a very poor performance by PC-ext. the bias free approach had significantly worse results for the datasets with extended attributes. This suggests that its advantage lies in comparing different attributes, and not necessarily finding the best split. This experiment also showed that it is not possible to run c_{quad} in reasonable time for the `Shelter-15` dataset. This happens because it calculates a pseudoinverse of a

Dataset	k	Twoing	GWSG	GWχ^2	GLSG	GLχ^2	PC-ext	HcC
Adult	2	**2.7**	88.2	41.3	3	4	2.8	3.8
Mushroom	2	**0.6**	8.6	6.9	0.9	1	0.7	0.9
KDD98-2	2	**4**	3579.3	2162	43.5	44	5.2	5.8
Contracep	3	**0.1**	1	0.6	0.1	0.1	0.1	0.1
Contracep-Ext	3	**0.1**	12.7	3.3	0.2	0.2	0.2	0.2
Cars	4	**0.1**	2.4	2.5	0.1	0.1	0.1	0.2
Cars-Ext	4	**0.2**	11.3	7.7	0.3	0.3	0.2	0.4
Nursery	5	**0.8**	5	4.7	1	0.9	0.9	1.2
Nursery-Ext	5	**1.1**	147.9	75.8	3.3	2.6	1.4	1.7
CoverType	7	348.5	**179.2**	265.4	245.2	307.8	271.4	338.5
CoverType-Ext	7	213	212.5	340.8	**182.5**	295.7	196.5	258.9
KDD98-9	9	132	5898.8	3410.1	97.2	73.9	**14.7**	297.5
Poker	10	7.4	11.2	6.4	2.2	**2.1**	2.5	13.3
Shelter-15	15	3599	—	—	149.9	166.3	**14.1**	7968
S.F. Crime-15	15	638.2	605.2	823.7	40.7	**40.2**	41.7	1223.9
Phonemes	15	1343.3	—	—	4.4	5.5	**2.2**	2187.4

Table 5.7: Average time in seconds of a 3-fold cross validation for building decision trees with depth at most 5. The fastest method for each dataset is bold faced.

matrix whose dimension grows with the number of values and classes, which is infeasible for large n and k.

Table 5.9 shows experiments similar to those presented at Table 5.6, but now using also the numeric attributes. We observed a significant gain in terms of accuracy for all datasets except for KDD98-2. Also note that GLSG was much inferior to the other criteria. Other than that, the comparison results were very similar to the ones found in Table 5.6, with a slight better performance by Hypercube Cover and GL-χ^2 and slightly worse by Twoing.

5.3.3 Maximum Depth 16

In this subsection we explore the same experiments studied in the previous one, except now the maximum depth allowed is 16. This larger depth is common in most random forest methods, thus the importance of this analysis.

Table 5.10 presents the results of an experiment to compare the accuracy of Decision Trees built by our GL-methods with those built by Twoing, Hypercube Cover and PC-ext, using just the nominal attributes of the datasets. Once again PC-ext had the best results, but this time by a smaller margin. This suggests the possibility that, for a larger depth, it might not have the best results. Moreover, there was a balance between Twoing, Hypercube Cover and GLχ^2, with GLSG being significantly worse. This suggests that GLSG loses competitiveness as the tree depth increases.

Dataset	CI-Twoing		CI-GLSG		CI-GLχ^2		CI-PC-ext		CI-HcC	
Adult	**81.96**	**(3)**	81.61	(1)	81.77	(2)	79.98	(0)	**81.96**	**(3)**
Mushroom	86.97	(1)	**94.79**	**(4)**	90.15	(3)	86.77	(0)	86.97	(1)
KDD98-2	81.29	(0)	**82.34**	**(4)**	81.68	(3)	81.35	(0)	81.29	(0)
KDD98-9	41.84	(0)	42	(0)	**42.26**	**(3)**	41.73	(0)	41.95	(1)
Nursery	**88.48**	**(1)**	88.3	(0)	88.47	**(1)**	**88.48**	**(1)**	**88.48**	**(1)**
Nursery-Ext	**88.48**	**(1)**	88.26	(0)	88.47	**(1)**	**88.48**	**(1)**	**88.48**	**(1)**
Cars	86.51	**(1)**	85.02	(0)	**86.57**	**(1)**	86.48	**(1)**	86.48	**(1)**
Cars-Ext	88.27	**(1)**	85.51	(0)	**88.32**	**(1)**	88.26	**(1)**	88.26	**(1)**
Contracep	43.83	(0)	**44.12**	**(3)**	43.87	(0)	43.63	(0)	43.69	(0)
Contracep-Ext	43.39	(0)	**43.81**	**(3)**	43.76	**(3)**	43.21	(0)	43.32	(0)
CoverType	54.1	(0)	54.1	(0)	54.1	(0)	54.1	(0)	54.1	(0)
CoverType-Ext	54.1	(0)	54.1	(0)	54.1	(0)	54.1	(0)	54.1	(0)
Poker	**51.05**	**(3)**	50.56	(0)	50.91	(1)	50.8	(1)	50.79	(1)
Shelter-15	48.03	(0)	48.03	(0)	**48.22**	**(2)**	48.22	**(2)**	48.2	**(2)**
S.F. Crime-15	**21.59**	**(4)**	20.53	(0)	21.49	(1)	21.52	(1)	21.52	(1)
Phonemes	23.12	(3)	22.52	(2)	**23.8**	**(4)**	21.3	(0)	22.11	(1)
Average (Sum)	61.44	(18)	61.6	(17)	61.75	(26)	61.15	(8)	61.36	(14)

Table 5.8: Average accuracy and statistical tests for Conditional Inference trees with depth at most 5 using only nominal attributes. The best accuracy for each dataset is bold-faced.

Dataset	Twoing		GLSG		GLχ^2		PC-ext		HcC	
Adult	84.19	(1)	82.36	(0)	84.15	(1)	84.4	**(3)**	**84.4**	**(3)**
KDD98-2	80.93	(0)	81.92	**(1)**	**81.93**	**(1)**	81.9	**(1)**	81.89	**(1)**
KDD98-9	45.68	(1)	45.32	(0)	**48.73**	**(4)**	47.38	(3)	47.04	(2)
Contracep	54.13	(1)	52.85	(0)	53.99	(1)	**55.11**	**(3)**	55.04	**(3)**
Contracep-Ext	51.66	(0)	52.27	(0)	53.26	(2)	**53.31**	**(3)**	52.75	(2)
CoverType	**70.25**	**(2)**	68.52	(0)	69.23	(1)	70.24	**(2)**	70.24	**(2)**
CoverType-Ext	70.88	(2)	70.3	(1)	69.23	(0)	**71.66**	**(4)**	71.25	(3)
Shelter-15	53.72	(1)	51.91	(0)	54.57	(2)	**54.83**	**(4)**	54.64	(2)
S.F. Crime-15	23.53	(3)	23.18	(0)	**23.58**	**(4)**	23.49	(1)	23.49	(1)
Average (Sum)	59.44	(11)	58.74	(2)	59.85	(16)	60.26	(24)	60.08	(19)

Table 5.9: Average accuracy and statistical test results for Decision Trees using both nominal and numeric attributes.

Dataset	Twoing		GLSG		GLχ^2		PC-ext		HcC	
Adult	**82.52**	**(2)**	82.38	(0)	82.43	(0)	82.51	**(2)**	**82.52**	**(2)**
Mushroom	**100**	**(1)**	100	(0)	99.99	(0)	**100**	**(1)**	**100**	**(1)**
KDD98-2	79.41	(0)	**80.65**	**(4)**	79.73	(0)	79.91	(2)	79.41	(0)
KDD98-9	38.54	(2)	37.97	(0)	**39.68**	**(4)**	38.55	(2)	37.95	(0)
Nursery	93.51	(2)	92.39	(0)	**93.74**	**(4)**	93.5	(1)	93.5	(1)
Nursery-ext	95.83	(1)	94.79	(0)	**96.02**	**(4)**	95.83	(1)	95.83	(1)
Cars	92.82	**(1)**	90.51	(0)	**92.88**	**(1)**	92.69	**(1)**	92.69	**(1)**
Cars-ext	96.49	**(2)**	90.89	(0)	94.44	(1)	96.46	**(2)**	**96.5**	**(2)**
Contracep	43.5	(0)	43.83	(0)	**43.85**	**(1)**	43.53	(0)	43.53	(0)
Contracep-ext	43.15	(0)	**44.32**	**(3)**	43.72	(0)	43.75	(2)	43.37	(0)
CoverType	64.09	(1)	64.59	(2)	63.61	(0)	**64.66**	**(3)**	**64.66**	**(3)**
CoverType-ext	65.08	**(1)**	**65.08**	**(1)**	65.05	(0)	65.08	**(1)**	65.08	**(1)**
Poker	**52.24**	**(2)**	49.88	(0)	51.83	(1)	52.09	**(2)**	52.09	**(2)**
Shelter-15	47.64	(2)	46.52	(0)	**48.09**	**(4)**	47.71	(2)	47.26	(1)
S.F. Crime-15	22.08	**(1)**	22.06	(0)	**22.08**	**(1)**	22.08	**(1)**	22.08	**(1)**
Phonemes	37.13	(2)	35.9	(0)	35.75	(0)	**37.95**	**(3)**	37.89	**(3)**
Average (Sum)	65.88	(20)	65.11	(10)	65.81	(21)	66.02	(26)	65.9	(19)

Table 5.10: Average accuracy and statistical tests for decision trees with depth at most 16 using only nominal attributes. The best accuracy for each dataset is bold-faced, even when multiple criteria have the same accuracy in the table because of rounding.

Table 5.11 presents the comparison between the different methods in the scenario where we use c_{quad} to select the attribute at each node of the tree, and the criteria are used only for splitting the chosen attribute. We observed a very strong advantage for GLχ^2, followed by Twoing. Both Hypercube Cover and GLSG are competitive among themselves, while PC-ext had the worse results.

Table 5.12 shows the running time of each criteria when used for both selecting and splitting purposes (the experiment of Table 5.10). When the number of classes is small all the criteria have very similar execution time. After running this experiments a few times, we have realized there is a variation of about 2-3 seconds in the execution time. This explains why Hypercube Cover is sometimes faster than Twoing.

As the number of classes increases, both PC-ext and the GL-based methods become much faster than Twoing and Hypercube Cover, with the turning point being once again around $k = 7$. For datasets with 15 classes our criteria are 30-400 times faster than Twoing, while PC-ext is 50-600 times faster. We also ran experiments using all the classes available in both the S.F. Crime and Shelter datasets (39 and 22, respectively). Twoing cannot be executed in a reasonable time with that many classes, while GLSG and GLχ^2 ran in approximately 100 seconds on the S.F. Crime dataset and 300 seconds on the Shelter dataset (PC-ext ran in 110 and 33 seconds, respectively). Since the execution time for our criteria in this experiment grew onde again in an

Dataset	CI-Twoing		CI-GLSG		CI-GLχ^2		CI-PC-ext		CI-HcC	
Adult	82.33	**(2)**	82.18	(0)	**82.35**	**(2)**	82.28	(1)	82.33	**(2)**
Mushroom	99.55	(1)	99.6	(1)	**99.64**	**(3)**	99.4	(0)	99.55	(1)
KDD98-2	79.92	(1)	**81.53**	**(4)**	80.65	(3)	79.72	(0)	79.92	(1)
KDD98-9	39.44	(2)	**40.65**	**(4)**	40.01	(3)	38.85	(0)	39.07	(0)
Nursery	93.55	**(1)**	92.22	(0)	**93.56**	**(1)**	93.55	**(1)**	93.55	**(1)**
Nursery-ext	93.64	**(1)**	92.32	(0)	**93.65**	**(1)**	93.64	**(1)**	93.64	**(1)**
Cars	92.74	(1)	87.19	(0)	**92.92**	**(4)**	92.67	(1)	92.67	(1)
Cars-ext	93.12	(1)	87.38	(0)	**93.34**	**(4)**	93.08	(1)	93.08	(1)
Contracep	43.82	**(2)**	**44.05**	**(2)**	43.81	(0)	43.59	(0)	43.6	(1)
Contracep-ext	43.21	(0)	**43.78**	**(3)**	43.72	**(3)**	43.07	(0)	43.12	(0)
CoverType	61.88	(0)	61.88	(0)	61.88	(0)	61.88	(0)	61.88	(0)
CoverType-ext	61.88	(0)	61.88	(0)	61.88	(0)	61.88	(0)	61.88	(0)
Poker	**51.4**	**(4)**	50.67	(0)	50.84	(1)	51.04	(1)	51.03	(1)
Shelter-15	47.66		47.32		**48.14**		47.82			
S.F. Crime-15	22.07	(0)	22.07	(0)	**22.07**	**(1)**	22.07	(0)	22.07	(0)
Phonemes	33.91	(2)	31.26	(0)	33.32	(1)	34.08	(2)	**34.77**	**(4)**
Average (Sum)	65.01	( )	64.12	( )	65.11	( )	64.91	( )		

Table 5.11: Average accuracy and statistical tests for conditional inference trees with depth at most 16 using only nominal attributes. The best accuracy for each dataset is bold-faced, even when multiple criteria have the same accuracy in the table because of rounding.

approximately linear fashion with k, it suggests that they can also be used with datasets that have a much larger number of classes. It is also interesting to note that the aggregated attributes usually appeared at or near the root of the decision trees.

Table 5.13 shows experiments similar to those presented at Table 5.10, except now it also uses the numeric attributes. We observed a significant gain in terms of accuracy for almost all datasets. This time Hypercube Cover has the best results, followed by PC-ext. The other 3 criteria have similar, but inferior, results.

Dataset	k	Twoing	GLSG	GLχ^2	PC-ext	HcC
Adult	2	4.1	4.6	5.7	**4**	5.8
Mushroom	2	1.7	1.5	2.3	**0.6**	1
KDD98-2	2	9.1	50.9	53.4	**8.3**	10.4
Contracep	3	0.2	0.2	0.2	**0.1**	0.1
Contracep-Ext	3	0.2	0.3	0.4	**0.2**	0.3
Cars	4	0.3	0.3	0.2	**0.2**	0.3
Cars-Ext	4	0.3	0.5	0.4	**0.2**	0.3
Nursery	5	4.1	3.9	3.7	**1.3**	1.9
Nursery-Ext	5	4.6	9.5	8.6	**1.4**	2.3
CoverType	7	612.7	**294.7**	700	472.4	668.9
CoverType-Ext	7	251.8	209.9	462.6	**200.4**	296.2
KDD98-9	9	434.5	256.2	243.6	**23.8**	442.9
Poker	10	25.2	10.8	9.2	**3.8**	18
Shelter-15	15	5397	181.9	191.2	**18.6**	7311.6
S.F. Crime-15	15	2731.2	95.9	81.8	**50.5**	3188.9
Phonemes	15	3894.6	9.4	11	**6.3**	5804.8

Table 5.12: Average time in seconds of a 3-fold cross validation for building decision trees with depth at most 16. The fastest method for each dataset is bold-faced.

Dataset	Twoing		GLSG		GLχ^2		PC-ext		HcC	
Adult	83	(1)	77.34	(0)	83.21	**(2)**	**83.28**	**(2)**	83.25	**(2)**
KDD98-2	76.67	(2)	76.36	(1)	76.04	(0)	**77.87**	**(4)**	77.14	(3)
KDD98-9	37.73	(1)	37.49	(0)	**43.45**	**(4)**	39.88	(3)	38.96	(2)
Contracep	48.78	**(1)**	48.01	(0)	48.66	**(1)**	**48.93**	**(1)**	48.86	**(1)**
Contracep-Ext	48.53	(0)	48.15	(0)	48.6	(0)	48.52	(0)	**49.31**	**(4)**
CoverType	85.14	(1)	**90.32**	**(4)**	81.38	(0)	86.23	(2)	86.23	(2)
CoverType-Ext	89.1	(2)	**92.03**	**(4)**	82.46	(0)	88.32	(1)	89.39	(3)
Shelter-15	53.79	(2)	52	(0)	**54.4**	**(4)**	53.82	(2)	53.59	(1)
S.F. Crime-15	26.66		26.71		**27.13**		27.13			
Average (Sum)	61.05	( )	60.93	( )	60.59	( )	61.55	( )		( )

Table 5.13: Average accuracy and statistical test results for Decision Trees using both nominal and numeric attributes with depth at most 16.

6
Conclusions

In this dissertation we proposed a framework for designing splitting criteria for handling multi-valued nominal attributes. Criteria derived from our framework can be implemented to run in polynomial time in n and k, with theoretical guarantee of producing a split that is close to the optimal one. This is the only known criteria that have all these characteristics simultaneously.

Experiments over 11 datasets suggest that the GLχ^2 criterion, obtained from our framework, is competitive with the well-established Twoing criterion in terms of both accuracy and speed for datasets with a small number of classes ($k \leq 7$). It is also much faster than Twoing when the number of classes is greater than 10, while keeping a comparable accuracy. Even though the PC-ext criterion does not have a theoretical guarantee, the experiments also show that it has some advantage in terms of accuracy and speed over the other methods, except when used in the conditional inference tree framework. This suggests that PC-ext is very good in terms of comparing different attributes among themselves, but not as good in finding the best split for a given attribute. In these bias-free experiments, the GLχ^2 criterion had the best results.

Therefore, our methods are an interesting alternative to deal with datasets with a large number of classes that contain nominal attributes with a large number of different values, since those cannot be properly handled by Twoing due to its exponential running time dependence on the number of classes. In practice, one should also consider using PC-ext and comparing their results in a cross-validation.

Furthermore, our experiments also reinforce the potential of aggregating attributes as a tool for improving the accuracy of decision trees. An interesting topic for future research is evaluating the behavior of our criteria in boosted tree methods. Another direction for future work is developing new methods for automatic aggregating attributes, or improving the available ones.

Bibliography

Angel, O., Bubeck, S., Peres, Y. & Wei, F. (2017), Local max-cut in smoothed polynomial time, *in* 'Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing', STOC 2017, ACM, New York, NY, USA, pp. 429–437. DOI 10.1145/3055399.3055402. URL http://doi.acm.org/10.1145/3055399.3055402.

Asuncion, A. & Newman, D. (2007), 'Uci machine learning repository'.

Austin Animal Center (2016), 'Shelter animal outcomes dataset', *URL: kaggle.com/c/shelter-animal-outcomes*.

Barros, R. C., de Carvalho, A. C. & Freitas, A. A. (2015), *Automatic design of decision-tree induction algorithms*, Springer.

Breiman, L., Friedman, J., Stone, C. J. & Olshen, R. A. (1984), *Classification and regression trees*, CRC press.

Burshtein, D., Della Pietra, V., Kanevsky, D. & Nadas, A. (1992), 'Minimum impurity partitions', *The Annals of Statistics* pp. 1637–1646.

Chou, P. A. (1991), 'Optimal partitioning for classification and regression trees', *IEEE Transactions on Pattern Analysis & Machine Intelligence* (4), 340–354.

Coppersmith, D., Hong, S. J. & Hosking, J. R. (1999), 'Partitioning nominal attributes in decision trees', *Data Mining and Knowledge Discovery* **3**(2), 197–217.

Dobra, A. & Gehrke, J. (2001), Bias correction in classification tree construction, *in* 'Proceedings of the Eighteenth International Conference on Machine Learning', Morgan Kaufmann Publishers Inc., pp. 90–97.

Goemans, M. X. & Williamson, D. P. (1995), 'Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming', *Journal of the ACM (JACM)* **42**(6), 1115–1145.

Goethals, O. M. & Rokach, L. (2005), 'The data mining and knowledge discovery handbook'.

Hothorn, T., Hornik, K. & Zeileis, A. (2006), 'Unbiased recursive partitioning: A conditional inference framework', *Journal of Computational and Graphical statistics* **15**(3), 651–674.

Kaggle (2012), 'Titanic survival', *URL: kaggle.com/c/titanic-survival*.

Laber, E. S., Molinaro, M. & de A. Mello Pereira, F. (2018), On the approximation of binary partitions with minimum impurity.

Loh, W.-Y. (2009), 'Improving the precision of classification trees', *The Annals of Applied Statistics* pp. 1710–1737.

Loh, W.-Y. (2014), 'Fifty years of classification and regression trees', *International Statistical Review* **82**(3), 329–348.

Mehta, M., Agrawal, R. & Rissanen, J. (1996), 'Sliq: A fast scalable classifier for data mining', *Advances in Database Technology—EDBT'96* pp. 18–32.

Mingers, J. (1987), 'Expert systems—rule induction with statistical data', *Journal of the operational research society* **38**(1), 39–47.

Mola, F. & Siciliano, R. (1997), 'A fast splitting procedure for classification trees', *Statistics and Computing* **7**(3), 209–216.

Nádas, A., Nahamoo, D., Picheny, M. A. & Powell, J. (1991), An iterative'flip-flop'approximation of the most informative split in the construction of decision trees, *in* 'Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on', IEEE, pp. 565–568.

Navascues, M., Owari, M. & Plenio, M. B. (2009), 'Power of symmetric extensions for entanglement detection', *Physical Review A* **80**(5), 052306.

Navratil, J. (2006), Recent advances in phonotactic language recognition using binary-decision trees., *in* 'Interspeech'.

Quinlan, J. R. (2014), *C4.5: programs for machine learning*, Elsevier.

Sahni, S. & Gonzalez, T. (1976), 'P-complete approximation problems', *Journal of the ACM (JACM)* **23**(3), 555–565.

SF-OpenData (2015), 'San francisco crime dataset', *URL: kaggle.com/c/sf-crime*.

Shih, Y.-S. (2001), 'Selecting the best splits for classification trees with categorical variables', *Statistics & probability letters* **54**(4), 341–345.

Shih, Y.-S. (2004), 'A note on split selection bias in classification trees', *Computational statistics & data analysis* **45**(3), 457–466.

Weide, R. L. (1998), 'The cmu pronouncing dictionary', *URL: http://www. speech. cs. cmu. edu/cgibin/cmudict*.