UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Random Intersection Trees

for Genomic Data Analysis

Dissertation presented by
Robin BALLARINI

for obtaining the Master's degree in
Computer Science and Engineering
Option(s): Artificial Intelligence, Software Engineering and Programming Systems

Supervisor(s)
Pierre DUPONT

Reader(s)
Roberto D'AMBROSIO, Jérôme PAUL

Academic year 2015-2016

Abstract

In Machine Learning classification, searching for informative interactions in large high-dimensional datasets is computationally intensive. Most algorithms that attempt this usually start with an empty set of variables and greedily add to that set. The drawback is that those approaches tend to miss some informative interactions due to their greedy behaviour. On the other hand, the brute force approach does not exhibit this greedy behaviour but has a high computational cost that renders problems with even moderate numbers of variables infeasible.

In 2014, Rajen Dinesh Shah and Nicolai Meinshausen published an article [1] on an alternative approach called *Random Intersection Trees*. This new approach starts from the full set of variables and removes them by taking intersections with randomly chosen instances. This algorithm boasts a reduced computational cost compared to the brute force method while avoiding the drawbacks of greedy behaviour.

The algorithm, as described in [1], remains limited to datasets with binary variables. In this thesis, we will propose modifications to the algorithm in order to generalise it to problems with both continuous and categorical variables. We will also propose classification rules inspired by Random Ferns (see [2] and [3]) and Naive Bayes that make use of the interactions from the *Random Intersection Trees* algorithms. Each variant of *Random Intersection Trees* proposed in this thesis will be analysed from a classification and feature selection perspective. The focus will be placed on how these algorithms perform on high-dimensional (genomic) datasets and why some of them perform better or worse than others.

Keywords: machine learning, classification, interactions, random intersection trees, high-dimensional data

Acknowledgments

I would like to express my deepest appreciation to all those who helped me with this thesis. In particular, I give a special gratitude to Prof. Pierre Dupont and Dr. Roberto D'Ambrosio from the ICTEAM institute (INGI dept., UCL) for the suggestions and feedback that they provided during the development of this thesis.

In addition, I would like to thank my family for their support and encouragement throughout my study.

Contents

Introduction

In Machine Learning classification, searching for informative interactions in large high-dimensional datasets is computationally intensive. Most algorithms that attempt this usually start with an empty set of variables and greedily add to that set. The drawback is that those approaches tend to miss some informative interactions due to their greedy behaviour. On the other hand, the brute force approach does not exhibit this greedy behaviour but has a high computational cost that renders problems with even moderate numbers of variables infeasible.

In 2014, Rajen Dinesh Shah and Nicolai Meinshausen published an article [1] on an alternative approach called *Random Intersection Trees*. This new approach starts from the full set of variables and removes them by taking intersections with randomly chosen instances. This algorithm boasts a reduced computational cost compared to the brute force method while avoiding the drawbacks of greedy behaviour.

The algorithm, as described in [1], remains limited to datasets with binary variables. In this thesis, we will propose modifications to the algorithm in order to generalise it to problems with both continuous and categorical variables. We will also propose classification rules inspired by Random Ferns (see [2] and [3]) and Naive Bayes that make use of the interactions from the *Random Intersection Trees* algorithms. Each variant of *Random Intersection Trees* proposed in this thesis will be analysed from a classification and feature selection perspective. The focus will be placed on how these algorithms perform on high-dimensional (genomic) datasets and why some of them perform better or worse than others.

Contribution

The main contribution of this work is a first study of the *Random Intersection Trees* algorithm. This thesis proposes three variants of the original *Random Intersection Trees* algorithm, all of them applicable to continuous and categorical features as well as n class problems. Each variant is described and its performance on various datasets documented. These variants, like the original algorithm, have the potential to be used in many ways to build a classifier. For instance, it is possible to feed the resulting interactions from the algorithms into a tree-based classifier such as Random Forests [4] or CART [5] although this particular use is not documented in this work. In addition to classification, the algorithms may also be used for the sake of the returned interactions themselves as they might be useful for knowledge discovery.

The algorithms described in this thesis are still prototypes as the associated classifiers sometimes have lacklustre performance on genomic datasets. Nevertheless, this thesis reports those results and proposes theories to explain why the classifiers sometimes fail on such datasets.

Structure

The remainder of this document is structured as follows: the first chapter describes the principles of *Random Intersection Trees* as well as the original algorithm published in [1]. The second through fourth chapters describe the three proposed variants of *Random Intersection Trees*. In addition to a description, these chapters contain an analysis of how the algorithms perform on standard and genomic datasets and a subsequent discussion on various issues. Each analysis discusses the performance with a basic classification rule as well as the performance when *Random Intersection Trees* is used for feature selection. A brief analysis of the time complexity is also included in each chapter. Finally, the fifth chapter is very brief and only contains an overview of the experimental results.

Chapter 1

Random Intersection Trees

Contents

1.1 Description and concepts

Random Intersection Trees is a relatively new machine learning algorithm (published in [1] in 2014). The algorithm searches for informative interactions between variables. It does so by intersecting randomly chosen instances of the training set. As intersections between instances are chained together, more and more variables are removed from the interactions. The sampling of instances is then repeated enough times to give appropriate coverage of the dataset's interactions.

Definition 1.1. Given the full set of variables/features in the dataset $F = \{f_1, f_2, \cdots f_n\}$, an interaction is defined as a set of pairs (f_i, v_i) where each f_i is a feature and each v_i is a corresponding value for f_i. Note that a full instance from a dataset is also an interaction.

In practice, the v_i can be any kind of value representation. For instance, they could be intervals or sets of values.

Definition 1.2. Given an interaction $I_1 = \{(f_i, v_i)\}$ and an instance from the dataset $I_2 = \{(f_j, v_j)\}$, their intersection is defined as interaction I_3, the set of pairs (f_k, v_k) such that

$$\forall k \; \exists f_i \in I_1, f_j \in I_2 : f_i = f_j = f_k \; and \; dist(v_i, v_j) \leq \epsilon_k$$

where $\epsilon_k \geq 0$. The v_k may be equal to their corresponding v_i but this is not a necessity. Indeed, new values of v_k may also be computed from the v_i and v_j.

In other words, this is similar to an intersection between two sets: only features that belong to both I_1 and I_2 are kept but some extra features might be removed if their values vary beyond some threshold ϵ_k.

It follows from this definition that $|I_3| \leq |I_1| \leq |I_2|$.

Definition 1.3. We say that an instance I_2 *matches* or *has* an interaction I_1 if and only if, for all features $f_i \in I_1$, the value associated to f_i in I_2 is equivalent to v_i.

1.1.1 Informative interactions

An important aspect of *Random Intersection Trees* is that not all returned interaction are informative.

Definition 1.4. An interaction of variables is said to be *informative* if its presence and/or absence within an instance gives enough information about its class label.

We therefore need some criterion to determine if an interaction is informative or not. Later in this chapter, we will see that the algorithm from [1] uses prevalence probabilities to establish the criterion. Other variants of *Random Intersection Trees* might use other criteria, such as similarity measures based on distances. Yet another possibility would be the use of information-theoretical measures such as Symmetrical uncertainty (see [6] and [7]) to measure the relevance of the interaction with respect to the class label as if it were a feature.

Definition 1.5. The Symmetrical Uncertainty $SU(X, Y)$ between variables X and Y is an information-theoretical measure defined as follows:

$$SU(X, Y) = 2\left[\frac{IG(X|Y)}{H(X) + H(Y)}\right]$$
$$IG(X|Y) = H(X) - H(X|Y)$$

where $H(X)$ is the entropy of X and $H(X|Y)$ is the entropy of X after observing Y.

1.1.2 Tree structure

Definition 1.6. A tree consists of a set of nodes N and set of edges $E \in N^2$ that form a connected and acyclic graph. The tree is said to be rooted if a node is chosen as the root and all edges are directed away from it.

Another important aspect of the algorithm is that the search is organised as a rooted tree. This is done for performance reasons. Indeed, when computing an intersection between an interaction I_1 and an instance I_2, the main computational cost comes from I_1's length. This is because $|I_1| \leq |I_2|$ and searching I_2 can be done in logarithmic time using sorted collections or in constant time (on average) using hash tables. Consequently, we would like to perform as few intersections as possible with a large I_1. On the other hand, computing intersections with a smaller I_1 are computationally cheaper.

A tree structure reflects this perfectly as there are fewer intersections (edges) near the root of the tree and therefore fewer expensive intersections. This use of a tree structure is illustrated

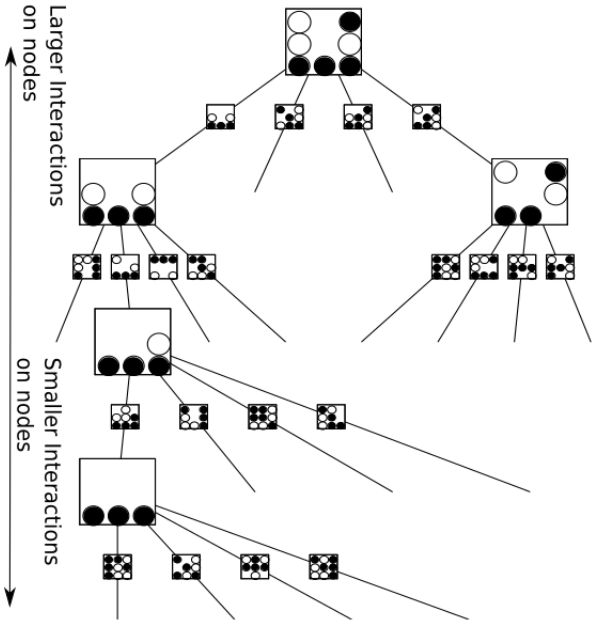


Figure 1.1: The tree structure in *Random Intersection Trees*. Image taken from [1] and edited.

in figure 1.1.

In high-dimensional datasets, intersections with the root node might become particularly expensive if the branching factor is high. under those circumstances, implementations of the algorithm might chose to split the search only when the depth of the current node is 1 or more. In other words, the root node is restricted to having a single child.

1.1.3 Uses of the algorithm

The first and most obvious use of this algorithm is to find interactions within the studied dataset. Indeed, interactions returned by *Random Intersection Trees* are essentially a set of rules that can be used for classification. Regardless of how those rules are actually used in a classifier, they remain easily interpretable and can therefore be used in knowledge discovery.

It follows that the second use of the algorithm is to insert the rules in some sort of classifier in order to predict the class labels of new instances. In theory, any classifier can be used and interactions can be viewed as new, transformed features that are fed into the classifier.

A third possible use of this algorithm is as a makeshift feature selection algorithm. This use stems from the idea that, if a feature never or rarely appears in an *informative* interaction, then it is likely irrelevant with respect to the class label. However, it should be noted that using the algorithm in this way does not remove redundant features and that the removal of an irrelevant feature is not guaranteed.

1.2 Binary Random Intersection Trees

In this section, we will go over the *Random Intersection Trees* algorithm as it is described in [1]. That particular variant of the algorithm will be referred to as the *Binary Intersection Trees* algorithm throughout this report. This section can also be viewed as an introduction to the other variants of the algorithm as they use many similar concepts.

1.2.1 Restrictions and assumptions

The *Binary Random Intersection Trees* algorithm was initially designed to be used in high-dimensional classification with binary features only. In practice, this is very inconvenient because few datasets consist solely of binary features.

Consequently, each dataset must be preprocessed adequately before the algorithm can be applied. The use of discretization algorithms is an obvious choice as a preprocessing step and is further explored in chapter 2.

Furthermore, the algorithm was designed for classification problems with two class labels. To apply it, as is, to a problem with more than two classes, one would need to make use of strategies such as *one-vs-one* or *one-vs-all*.

1.2.2 Data encoding

Since all features are binary, it follows that the various instances and interactions can be encoded in such a way that the computational cost of the algorithm is reduced.

The interactions are encoded as a collection of features (that may be represented by integers) $I_{enc} = \{f_1, f_2, f_3, \cdots f_I\}$ such that all of the features $f \in I_{enc}$ are those with a value of 1. On the other hand, the features $f \notin I_{enc}$ are those with a value of 0 or those that were removed in a previous intersection.

It was previously discussed that the algorithm's tree structure was chosen for computational reasons (cfr. subsection 1.1.2). This choice was made because, when an intersection in computed, the bulk of the computational cost comes from the parent interaction. The choice of such an encoding reduces the size of interactions and, consequently, the overall computational cost.

Example 1.1. *Consider an instance from a dataset that has the following values for the various features:*

$$[1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1]$$

This particular instance would be encoded as $[1, 3, 5, 7, 9, 10, 11, 12, 14, 15]$. *With such an encoding, the instance is stored as a list of length 10 instead of length 15.*

The encoding also simplifies the way intersections are computed from a purely conceptual point of view. Indeed, since we are now allowed to forego explicit values for the various features, computing an intersection between interaction and instance is essentially reduced to an intersection between two sets of integers.

1.2.3 Prevalence probabilities

As mentioned previously, the goal of *Random Intersection Trees* is the discovery of informative interactions. As such, a criterion is needed in order to determine if an interaction is informative of not. The criterion used in [1] makes use of prevalence probabilities.

Definition 1.7. The prevalence probability (or simply prevalence) of an interaction I with respect to a class label C is the probability of having an instance that matches I given that the instance has class label C. The prevalence is noted $\mathbb{P}(I|C)$ and can be computed

empirically as

$$\mathbb{P}_n(I|C) = \frac{\#Instances\ of\ class\ C\ matching\ I}{\#Instances\ of\ class\ C}$$

Assuming that there are two class labels C_1 and C_0, the criterion itself makes use of two user-defined prevalence thresholds θ_1 and θ_0 and states that an interaction I is considered informative if and only if at least one of the following equations is true.

$$\mathbb{P}(I|C_1) \geq \theta_1 \text{ and } \mathbb{P}(I|C_0) \leq \theta_0 \tag{1.1}$$
$$\mathbb{P}(I|C_0) \geq \theta_0 \text{ and } \mathbb{P}(I|C_1) \leq \theta_1 \tag{1.2}$$

Intuitively, this criterion marks an interaction I as informative if many instances of one class match I while few instances of the other class match it. Interactions that are not informative are typically filtered out as they do not enough provide information on the class label.

Classifiers Although, there is no standard classifier in which interactions are fed in [1], all examples described in the article use prevalences as weights of some sort in order to classify new instances.

1.2.4 Interaction search

Algorithm 1 describes the basic version of *Binary Random Intersection Trees*. It can be observed in the inner-most for loop that the algorithm indeed follows a tree structure if the nodes are indexed chronologically[1]. In this algorithm, each iteration m results in a set of candidate interactions $L_{D,m}$. In the end, the result is the union L_D of all sets of interactions.

It should be noted that the use of random variables as branching factors B_j was a theoretical choice introduced in [1]. In practice, all B_j would be equal to a fixed user-defined value.

Once algorithm 1 has run its course, one could test each of the returned interactions against the criterion described in subsection 1.2.3 and filter out those that violate it. The algorithm can also be repeated using instances of the other class if necessary.

Algorithm 1 A basic version of *Binary Random Intersection Trees*, taken from [1] and modified.

for tree $m = 1$ **to** n_{tree} **do**

Let m be a rooted tree of depth D, with each node j in levels $0, \ldots, D-1$ having B_j children, where the B_j are i.i.d. with a pre-specified distribution. Denote by J the total number of nodes in the tree, and index the nodes chronologically. For each of the nodes $j = 1, \ldots, J$, let $i(j)$ be an independently and uniformly chosen index in the set of class 1 observations $\{i : Y_i = C_1\}$.

Set $I_1 = X_{i(1)}$.
for node $j = 2$ **to** J **do**
Set $I_j = X_{i(j)} \cap I_{\text{pa}(j)}$.
end for

Denote the collection of resulting sets from all nodes at depth d, for $d = 1, \ldots, D$, by $L_{d,m} = \{I_j : \text{depth}(j) = d\}$.
end for
return candidate set of interactions $L_D := \bigcup_{m=1}^{n_{tree}} L_{D,m}$.

[1] i.e. Parents have smaller indexes j than their children and are visited before them

1.2.5 Early stopping

The basic version of *Binary Random Intersection Trees* is already much more attractive from a computational point of view than a brute-force approach to interaction search. Early stopping was proposed as a way to further reduce the computational cost of the algorithm.

Early stopping can be seen as a form of branch pruning on the algorithm's tree structure. In other words, if the algorithm can infer that a particular branch of the intersection tree will never produce any informative interaction, the branch can be removed altogether. To make this inference, the algorithm takes advantage of the following theorem.

Theorem 1.1. *Consider two interactions I_1 and I_2 such that I_2 is a descendant of I_1 in the intersection tree. We have that*

$$\forall C \in \{C_0, C_1\},\ \mathbb{P}_n(I_2|C) \geq \mathbb{P}_n(I_1|C)$$

As a direct consequence of this, it can be stated that if $\mathbb{P}_n(I_1|C) \geq \theta_C$ for a given C then $\mathbb{P}_n(I_2|C) \geq \theta_C$ for the same C. Concretely, this means that if an interaction I_1 violates $\mathbb{P}_n(I_1|C) \leq \theta_C$, then all its descendants I_{desc} violate $\mathbb{P}_n(I_{desc}|C) \leq \theta_C$.

Using this, one option would be to compute $\mathbb{P}_n(I_1|C_0)$ if the instances being intersected are from class C_1 and to compute $\mathbb{P}_n(I_1|C_1)$ if they are from class C_0. In the case where the computed prevalence exceeds the corresponding θ, the associated interaction can be discarded immediately and the algorithm needs not compute the intersections for its descendants. The revised algorithm (with early stopping) is depicted as algorithm 2.

Furthermore, concrete examples of how prevalences may vary on a branch of an intersection tree are depicted in figures 1.2 and 1.3. Notice that, while symmetrical uncertainty can be used to evaluate if an interaction is informative or not, it cannot be used to implement early stopping (i.e. because it is not monotonic).

Algorithm 2 *Binary Random Intersection Trees* with early stopping, taken from [1] and modified.

for tree $m = 1$ **to** n_{tree} **do**

Let m be a rooted tree of depth D, with each node j in levels $0, \ldots, D-1$ having B_j children, where the B_j are i.i.d. with a pre-specified distribution. Denote by J the total number of nodes in the tree, and index the nodes chronologically. For each of the nodes $j = 1, \ldots, J$, let $i(j)$ be an independently and uniformly chosen index in the set of class 1 observations $\{i : Y_i = C_1\}$.

Set $I_1 = X_{i(1)}$.
for node $j = 2$ **to** J **do**
  if $\mathbb{P}_n(I_{\mathrm{pa}(j)}|C_0) \leq \theta_0$ **then**
    Set $I_j = X_{i(j)} \cap I_{\mathrm{pa}(j)}$.
  end if
end for

Denote the collection of resulting sets of all nodes at depth d, for $d = 1, \ldots, D$, by $L_{d,m} = \{I_j :\ \mathrm{depth}(j) = d\}$.
end for
return $L_D := \bigcup_{m=1}^{n_{tree}} L_{D,m}$.

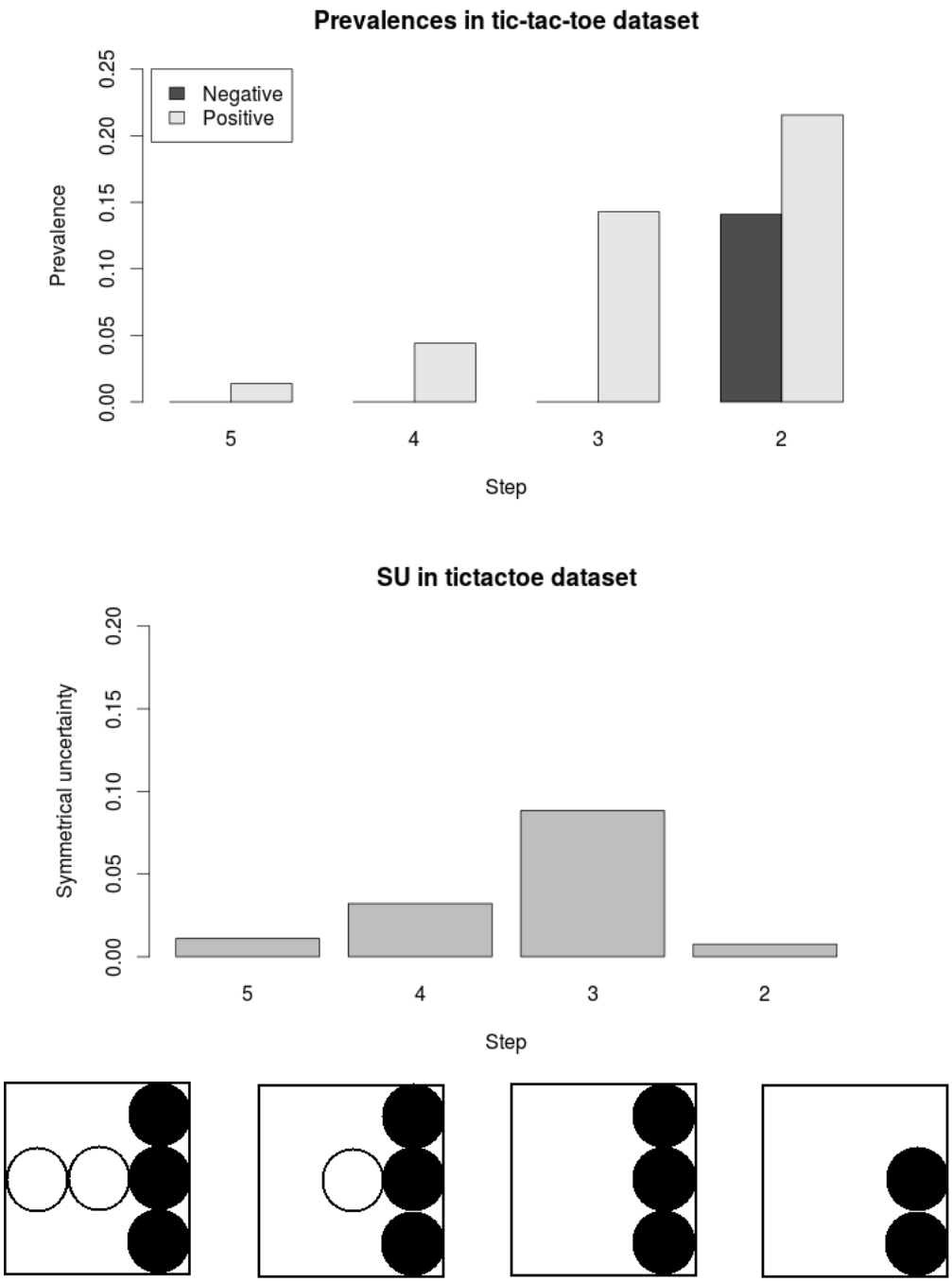


Figure 1.2: An example of a branch in an intersection tree (top → down depicted left → right) in the tic-tac-toe dataset. Early stopping should stop the search after the first three candidate interactions.

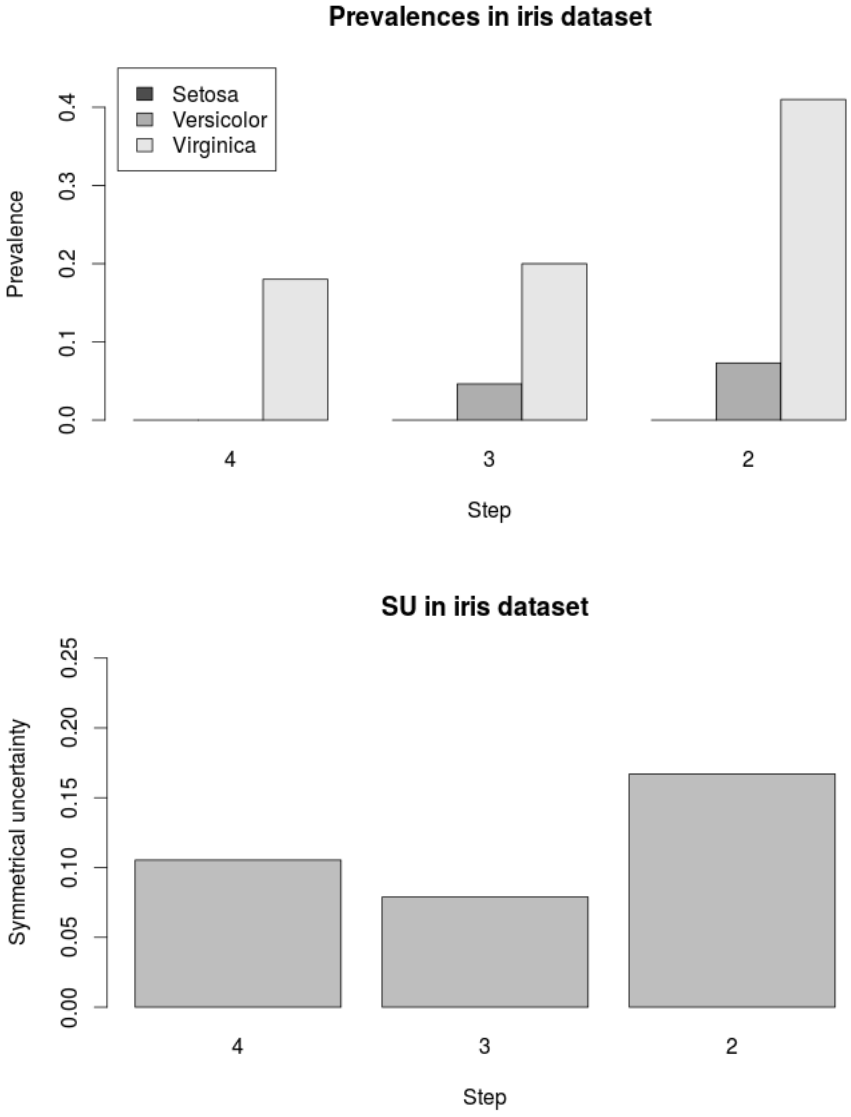


Figure 1.3: An example of a branch in an intersection tree (top → down depicted left → right) in the iris dataset.

Min-Wise Hashing

As a follow-up to the introduction of early stopping, it is stated in [1] that computing the required prevalence probabilities could be prohibitively expensive as it would require a pass over all instances of a given class in the training set. Furthermore, this computation would need to be repeated for every node[2].

A fast approximation method based on min-wise hashing was proposed. This method takes advantage of the data encoding described earlier (cfr subsection 1.2.2) as applying ideas from min-wise hashing would not be as straightforward otherwise.

The first step in this method is the computation of a $L \times p$ min-wise hash matrix H. Building this matrix involves L permutations of instances from a given class. Each entry H_{lk} is computed as follows:

1. Generate L random permutations of instances from a given class. Permutations are noted $\{\sigma_1, \sigma_2 \cdots \sigma_L\}$.
2. Assuming that there is a total of p features, we define

$$h_{\sigma_l}(k) = \min\{i' : \text{feature } k \text{ is 1 in instance } \sigma_l(i')\}$$

   with $l \in \{1, 2, \cdots L\}$ and $k \in \{1, 2, \cdots p\}$.
3. Entries are then defined as $H_{lk} = h_{\sigma_l}(k)$.

Once the matrix H has been generated using instances of class C_1. The prevalences of interactions with respect to C_1 can be computed using the following equations. Note that n is the number of instances with class C_1 and I_{enc} refers to the encoded form of interaction I.

$$\begin{aligned}
\hat{\mathbb{P}}_n(I|C_1) &= \hat{\pi}_1(L, I, H) \cdot \hat{\pi}_2(L, I, H) \\
\hat{\pi}_1(L, I, H) &= \frac{1}{L} \sum_{l=1}^{L} \mathbb{1}_{\{H_{lk} = H_{lk'} \text{ for all } k,k' \in I_{enc}\}} \\
\hat{\pi}_2(L, I, H) &= \frac{n+1}{n} \left\{ \frac{1}{\frac{1}{L} \sum_{l=1}^{L} \min_{k \in I_{enc}} H_{lk}} - \frac{1}{n+1} \right\}
\end{aligned}$$

Note that a given matrix H only needs to be computed once at the start of the algorithm. Additionally, it might be useful to build a matrix for both classes, especially if $\mathbb{P}(I|C_0)$ and $\mathbb{P}(I|C_1)$ are both needed for a subsequent classification task.

[2] with the exception of those that get dropped by the early stopping

Chapter 2

Random Intersection Trees with Discretization

Contents

2.1 Motivation

Since the *Binary Random Intersection Trees* algorithm is restricted to binary features and two classes, we will propose and describe an alternative algorithm called *Random Intersection Trees with Discretization*. For the sake of brevity, we might refer to this algorithm as *discRIT* or *baseRIT*.

The goal was to produce a modified version of *Binary Random Intersection Trees* with the following characteristics:

- Able to deal with both continuous and categorical features.
- Able to deal with more than two classes. Ideally, this should be done in a smarter way than simply applying a *one-vs-one* or *one-vs-all* strategy.

Definition 2.1. A feature is *continuous* if and only if its values have a total order. In other words, for all pairs of values v_1 and v_2 from feature f, we have $v_1 < v_2$, $v_2 < v_1$ or $v_1 = v_2$.

Definition 2.2. A feature is *categorical* if and only if there is no relation of order between any of its values. In other words, for all pairs of values v_1 and v_2 from feature f, we can only state $v_1 = v_2$ or $v_1 \neq v_2$.

The rough idea behind *discRIT* is to add a pre-processing step in order to discretize the dataset. Once the dataset has been discretized, it can effectively be expressed as a dataset with binary features only (see property and example below). We can then apply a slightly modified version of *Binary Random Intersection Trees* for the actual interaction search.

Property 2.1. *Any discrete feature f_d may be transformed into a set of binary features. Each of those binary features corresponds to one of the discrete labels in f_d. The value of a binary feature is set to 1 for a given instance if and only if that instance had the corresponding label in f_d.*

Example 2.1. *An example of a single discrete feature being transformed. The feature before transformation is on the left. The result of the transformation is on the right. In this case, the feature is categorical but, through the use of a discretization method, this type of transformation can also be applied to continuous features as well.*

	Color
Instance 1	*red*
Instance 2	*yellow*
Instance 3	*blue*

$\Rightarrow$

	Color = red	*Color = blue*	*Color = yellow*
Instance 1	*1*	*0*	*0*
Instance 2	*0*	*0*	*1*
Instance 3	*0*	*1*	*0*

2.2 Feature discretization

Definition 2.3. *Feature discretization* is a process through which a continuous feature f is transformed into a discrete feature f_d. This is done by determining intervals (or cut-points). Each interval is then assigned a different discrete label. The *granularity* of the discretized feature f_d is the number of distinct labels in f_d.

According to [8], there are three different axes by which discretization methods may be classified: global vs. local, supervised vs. unsupervised and static vs. dynamic.

- Local methods form partitions that are applied to localized portions of the instance space. In other words, the discretization is applied to a subset of the instances in the dataset. Since discretization is to be applied as a preprocessing step in *discRIT*, local methods will not be considered further. Global methods, which can be applied to an entire dataset, are more pertinent in this case.
- Unsupervised methods divide continuous attributes independently from the class labels. On the other hand, supervised methods do use the class labels in order to determine where an attribute's various cutpoints should be placed. For *discRIT*, we mainly focus on supervised methods as the algorithm is meant to be used in a classification context. However, unsupervised methods are not excluded from our research.

- Static (univariate) methods discretize features independently from each other while dynamic (multivariate) methods analyse all features simultaneously. By doing so, dynamic methods are able to capture interdependencies in feature discretization. While the use of multivariate methods in *discRIT* might be a promising avenue of research, the fact that they discretize according to interdependencies is somewhat redundant[1] with the objective of the *Random Intersection Trees* algorithms. Therefore, the focus will be placed on univariate methods.

Due to the focus on univariate methods, we can predict that *discRIT* may have some difficulties as a classification method if the discretization turns out to be poor. A typical pattern that could result in such a poor discretization is the (continuous) XOR pattern. Indeed, assuming a supervised and univariate method is used, it will be unable to discover any correlation between discretized values and class labels.

2.2.1 Methods

In our research and experiments on *discRIT*, we have considered various discretization methods, which are described below. All of these methods are univariate and global.

- *Equal-width binning* is an unsupervised discretization method which consists of binning a feature's values into k intervals of equal length. The parameter k is user-defined. The time complexity (for a single feature) is $\mathcal{O}(n)$, with n being the number of instances.
- *CAIM* (cfr. [9]) is a top-down supervised discretization method that uses a heuristic measure called the *Class-Attribute Interdependency Maximization* criterion. This particular method aims to produce a discretization with as few intervals as possible (low granularity). Given M the number of distinct values in the original feature, the expected running time of the algorithm is in $\mathcal{O}(M \log(M))$.
- *Ameva* (cfr. [10]) is a top-down supervised discretization method that uses a criterion based on χ^2 statistics. Like *CAIM*, *Ameva* is known to produce few intervals. Unlike *CAIM*, *Ameva* executes faster on datasets that have many different classes.
- *ChiMerge* (cfr. [11]) is a bottom-up supervised discretization method. It works by merging pairs of adjacent intervals if they do not exceed some χ^2 threshold. The user must provide a significance level α. The time complexity is in $\mathcal{O}(n \log(n))$ with n being the number of instances.

2.2.2 Mixed datasets

In order to ensure that *discRIT* can be applied to as many datasets as possible, mixed datasets must be taken into account. A mixed dataset is one that contains both continuous and categorical features. To transform continuous features into binary features, we can apply one of the discretization methods discussed previously. For categorical features, we can split them into a set of new features: one for each value.

Example 2.2. *Consider a categorical feature f_{cat} from the tic-tac-toe dataset. The feature has three possible values: x, o and b. It's possible to split this feature into three binary features: $f_{cat} = x$, $f_{cat} = o$ and $f_{cat} = b$.*

[1] Finding interdependencies in feature discretization is essentially the same problem as finding feature interactions

A brief pseudo-code for mixed dataset discretization is displayed in algorithm 3. In particular, it should be noted that a mapping between original and binary features should be kept in order to use *discRIT* as a classifier or as a feature selection algorithm.

Algorithm 3 Discretization of a mixed dataset.

Let A be the number of attributes in the dataset D. Denote by $D[i]$ the i^{th} attribute of D. Use method M for continuous features (e.g. $Ameva$). Let $split(f)$ be a function that returns a binary dataset for categorical feature f. Let $apply(f, M)$ be a function that returns a binary dataset for continuous feature f and uses discretization method M. Both functions also return information that maps each binary feature to their corresponding values in the original feature.

Initialize D_{disc} and Map_{disc}.
$c = 1$
for $a = 1$ **to** A **do**
  if $D[a]$ is categorical **then**
    $d = split(D[a])$
    for $i = 1$ **to** $length(d.data)$ **do**
      $D_{disc}[c] = d.data[i]$
      $value = d.values[i]$
      $Map_{disc}[c] = (a, value)$
      $c = c + 1$
    end for
  else
    $d = apply(D[a], M)$
    for $i = 1$ **to** $length(d.data)$ **do**
      $D_{disc}[c] = d.data[i]$
      $interval = (d.cutpoints[i][1], d.cutpoints[i][2])$
      $Map_{disc}[c] = (a, interval)$
      $c = c + 1$
    end for
  end if
end for
return D_{disc} the binary dataset. Map_{disc} a mapping between original and binary features.

2.2.3 Discretization and sparsity

The discretization of a dataset exposes an interesting property: by discretizing and binarizing the features, the dataset becomes sparse. This is because the intervals generated by the various discretization methods considered here are mutually exclusive. To see this, we have to re-examine what discretization is.

Property 2.2. *The discretization of a feature f is equivalent to establishing a function $D : dom(f) \rightarrow L$. This means that each value of f is mapped to a* ***single*** *discrete label l such that $l \in L$.*

Let $|F|$ be the number of features in the dataset. As a direct consequence of the above property, we can establish that each instance will have exactly $|F|$ binary features for which its value is 1, each corresponding to a different feature from the original dataset.

Plots of the sparsity and increase in the number of features after the discretization and binary transformation for various datasets are displayed in figures 2.1 and 2.2 respectively.

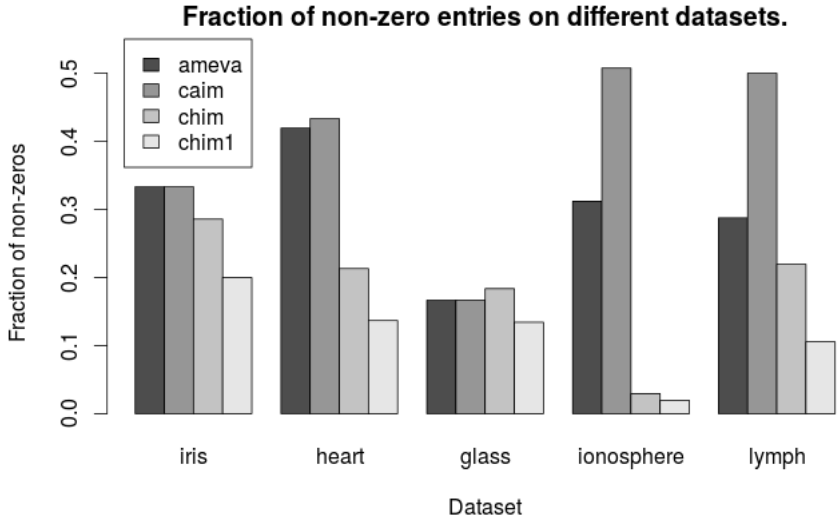


Figure 2.1: Sparsity after discretization. "chim" and "chim1" correspond to the ChiMerge method with $\alpha = 0.05$ and $\alpha = 0.1$ respectively.

This sparsity can be heavily exploited by using the same data encoding strategy as in subsection 1.2.2. This guarantees that candidate interactions within the algorithm will be of size $|F|$ at most. The time complexity is therefore reduced in this case.

Discretization methods and granularity It can be shown that sparsity and granularity are related. Indeed, a high sparsity in the binarized datasets suggests that the original features were split into many small intervals. This means that the granularity is high. In particular, the high sparsity of ChiMerge on the Ionosphere dataset should be noted.

2.3 Prevalence probabilities

Like in *Binary Random Intersection Trees*, informative interactions should be returned while other interactions should be filtered out. Since *discRIT* is essentially the same as the binary algorithm but with an extra preprocessing step, prevalence probabilities may still be used.

Unlike the binary algorithm, *discRIT* should be able to manage datasets with more than two classes. This requires that a few changes be made to the criterion used previously. The most simple way to adapt the criterion is to use as many prevalence thresholds θ_C as there are classes. We can then refine the criterion. An interaction I is informative if and only if the following are both true.

$$\exists C : \mathbb{P}(I|C) > \theta_C \tag{2.1}$$
$$\exists C : \mathbb{P}(I|C) < \theta_C \tag{2.2}$$

Another possibility is to use double the number of thresholds: $\theta_{upper,C}$ and $\theta_{lower,C}$. Since the thresholds are user-defined, this grants more control over the selection of interactions. This new criterion states that an interaction I is informative if and only if the following are both true.

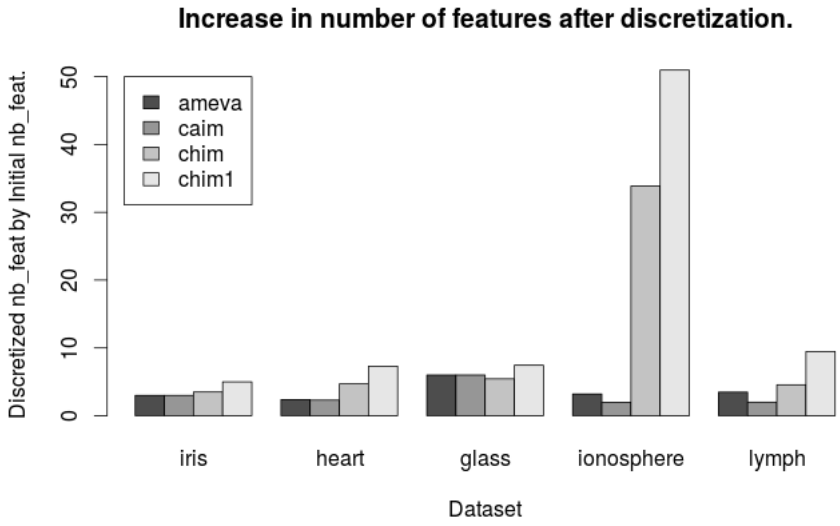


Figure 2.2: Increase in the number of features after discretization: $\frac{\#features_{binary}}{\#features_{orig}}$. "chim" and "chim1" correspond to the ChiMerge method with $\alpha = 0.05$ and $\alpha = 0.1$ respectively.

$$\exists C : \mathbb{P}(I|C) > \theta_{lower,C} \tag{2.3}$$

$$\exists C : \mathbb{P}(I|C) < \theta_{upper,C} \tag{2.4}$$

Note that this improved criterion was unused in the simulations and experiments. It is however described here for future research.

2.4 Early Stopping

Early stopping can still be used in the same way as in *Binary Random Intersection Trees*. In fact the only difference is the criterion used to determine if a branch needs to be cut off.

Despite the similarities however, we propose the following idea: that it is possible to use early stopping to force a given search branch to return an informative interaction if it has encountered one before. While this proposition on the use of early stopping does not guarantee that the classification performance will increase, it should not degrade it.

Consider an informative interaction I_1 and its children in the intersection tree. Intuitively, we would like this branch to yield some informative interaction rather than a set of non-informative interactions which will be filtered out. In view of this, we can guarantee that this branch will yield some informative interaction if it has encountered one before by using the following strategy:

1. Check that, for each child, $\exists C : \mathbb{P}(I|C) < \theta_C$ or $\exists C : \mathbb{P}(I|C) < \theta_{upper,C}$ depending on the criterion used.

2. If at least one child satisfies the above, continue the algorithm. Branches should be cut or grown as appropriate.

3. Otherwise, I_1 should be returned by the algorithm since none of its descendants will be informative.

arg max rule	Ameva+es			Ameva			equal-width+es			equal-width		
Dataset	BCR	ASM	Matches	BCR	ASM	Matches	BCR	ASM	Matches	BCR	ASM	Matches
Iris	0.94	NA	0.176	0.92	NA	0.17	0.91	NA	0.1	0.93	NA	0.1
Heart	0.8	NA	0.2	0.82	NA	0.2	0.83	NA	0.2	0.83	NA	0.2
Tic-tac-toe	0.96	NA	0.009	0.96	NA	0.012	0.96	NA	0.009	0.96	NA	0.012
Ionosphere	0.88	NA	0.4	0.89	NA	0.4	0.77	NA	0.24	0.79	NA	0.25
Glass	0.61	NA	0.194	0.64	NA	0.19	0.49	NA	0.25	0.45	NA	0.25

Table 2.1: Experimental results: BCR, ASM and the fraction of interactions that match. The algorithms are *discRIT* with Ameva and equal-width binning, with and without early stopping (es). These results are based on the arg max rule from section 2.6. The number of bins for equal-width binning is set to $\sqrt[3]{N}$ where N is the number of instances in the training set.

However, according to our experimental results, this strategy neither improves nor degrades classification performance significantly in practice. Those results can be viewed in table 2.1.

2.5 Interaction search

The interaction search used in *discRIT* is depicted in algorithm 4. The following points should be noted:

- Algorithm 4 displays the pseudo-code for a single tree on a given class. In the full *discRIT* algorithm, this would be repeated n_{tree} times for each class. Furthermore, algorithm 3 would be executed before this.
- Parameter Z was explicitly added. It is used to remove interactions that are unwanted because of their low size (e.g. empty interactions).
- It includes early stopping along with the modification discussed in section 2.4.
- Instances and interaction should be encoded as in subsection 1.2.2.
- Matrices for the min-wise hashing estimator should also be built before this pseudo-code. There should be a matrix per class label.

Depth-first search One of the changes displayed in algorithm 4 when compared with algorithm 2 is the use of DFS. Indeed, in the binary algorithm published in [1], the pseudo-code suggests all the interactions must be kept in memory regardless of their depth. This is also the way the algorithm was implemented in the *FSInteract* R package[2]. This is a waste of memory as only leaf nodes need to be returned. From there, we chose DFS over BFS in order to alleviate the memory requirements.

2.6 Classifier

There are several ways interactions can be used in order to build a classifier. In this section, we will go over several strategies that may be used.

As a general rule, an interaction returned by *discRIT* may be viewed as a binary feature whose value is 1 for a given instance I if and only if I matches the interaction. From that angle, the algorithm can be described as a transformation of the feature space. The "transformed

[2]Version 0.1.1 in particular.

Algorithm 4 Interaction search for a single tree on class C_1.

Consider class labels $C_1, C_2, C_3, ...C_n$. Let *data* contain the instances of the dataset that corresponds to class C_1. Let *select_instance* be a function that selects a random instance from a dataset. Let B be the branching factor, D the maximum depth and Z the minimum interaction size allowed.

Initialize empty stack of interactions $frontier$.
Initialize empty set of interactions $results$.
$root = select\_instance(data)$
$root.depth = 0$
$frontier.push(root)$
while $frontier$ is not empty **do**
  $parent = frontier.pop()$
  $depth = parent.depth$
  $nb_{valid} = 0$
  for $i = 1$ **to** B **do**
    $child = parent \cap select\_instance(data)$
    $child.depth = depth + 1$
    if $child.size() \geq Z$ and $\exists C : \mathbb{P}_n(child|C) < \theta_C$ **then**
      if $child.size() = Z$ or $child.depth = D$ **then**
        $results.add(child)$
      else
        $frontier.push(child)$
      end if
      $nb_{valid} = nb_{valid} + 1$
    end if
  end for
  if $nb_{valid} = 0$ **then**
    $results.add(parent)$
  end if
end while
return $results$ a set of interactions.

dataset" can then be used in conjunction with any model such as *Random Forest* or *Naive Bayes*.

Another possible classifier (and the one we used in our experiments) uses the following rule:

$$Class(a) = \arg\max_{C} \log(\mathbb{P}(C)) + \sum_{I:\ a \text{ matches } I} \log(\mathbb{P}_n(I|C))$$

in which a is the instance whose class label needs to be predicted. This particular rule was inspired by the random ferns classifier as used in [2]. As the rule sums log-probabilities, it is necessary to smooth the prevalences to avoid summing terms equal to $-\infty$. The simplest smoothing strategy (which we used in our experiments) is Laplace smoothing.

It can be noted that the above classifier does not make use of the absence $\mathbb{P}(\bar{I}|C) = 1-\mathbb{P}(I|C)$ of an interaction from an instance, which is what *Naive Bayes* would have done. This is a deliberate choice as, while the use of absence might be useful for some datasets, it can also serve as an obstacle to effective classification. This is particularly visible when there are very few matches between returned interactions and test instances as the "absences" will heavily influence the classification. Furthermore, an instance not matching an interaction might be caused by some quirk of the discretization method.

2.7 Algorithm Interface

Like most other models in machine learning, *discRIT* can be controlled by various hyper-parameters. This section summarizes what those hyper-parameters are.

- θ a vector whose length is equal to the number of classes. This hyper-parameter contains the prevalence thresholds that are used to determine if an interaction is informative or not. If the improved criterion is used, there are two vectors (θ_{lower} and θ_{upper}) instead.
- n_{tree} is the number of intersection trees per class.
- B is the branching factor used in the trees.
- D is the maximum depth for the trees.
- M is the name of the discretization method used for continuous features.
- Z is the minimum interaction size allowed.
- L is number of permutations used to compute the prevalences with the min-wise hashing estimator (cfr. 1.2.5).

2.8 Time Complexity

Since the original goal of the *Random Intersection Trees* family of algorithms was to improve on the computational cost of the brute force search of interactions, it is pertinent to analyse the time complexity to see how it performs. Note that, like the brute force search, the complexity bound is exponential. However, unlike the brute force search, the parameters involved in the exponential are user-controlled and may be assigned modest values while still retrieving interesting interactions.

Brute-force complexity For reference, a brute-force search of interactions of size s on a dataset with p features after discretization is in $\mathcal{O}(p^s)$ according to [1].

To study the time complexity of *discRIT*, the following notations will be used in addition to the hyper-parameters:

- $\#C$ the number of classes.
- $|F|$ the number of features before discretization.
- p the number of binary features after discretization.
- N the number of instances in the dataset.

In a single intersection tree, there are at most $\left(\sum_{i=0}^{D-1} B^i\right) \cdot B$ intersections to compute and each of those may be computed in $\mathcal{O}(|F|)$[3]. Assuming that early stopping is used, the algorithm needs to compute the prevalences for the classes after each intersection. This results in an additional cost in $\mathcal{O}(\#C \cdot L \cdot |F|)$ per intersection if the min-wise hashing estimator is used or in $\mathcal{O}(N \cdot |F|)$ per intersection if it is not. The process of building an intersection tree is repeated $n_{tree} \cdot \#C$ times.

The discretization step is executed in $\mathcal{O}(|F| \cdot Meth)$ where $Meth$ depends on the discretization method chosen. The min-wise hashing estimator requires matrices to be built at the start of the algorithm: this can be done in $\mathcal{O}(\#C \cdot (L \cdot (N + p)))$.

Note on the use of the min-wise hashing estimator with genomic datasets In general, a genomic dataset is characterized by its low number of instances N and its high number of features $|F|$. The low number of instances N renders the use of the estimator somewhat questionable for such datasets. Indeed, N usually remains low enough that computing the prevalences by iterating over all instances is an acceptable solution from the perspective of the computational cost. Furthermore, not using the estimator eliminates any imprecision that may come from its use.

2.8.1 Experimental study

In order to verify the theoretical expressions for the time complexity, several simulations were run.

First of all, in order to study the effect of various hyper-parameters, the algorithm was run repeatedly on the iris dataset. The resulting boxplots can be viewed in figure 2.3. From those plots, it can be observed that the execution time seems to increase exponentially with the depth, linearly with the number of trees and polynomially with the branching factor. These results confirm the theoretical expression obtained previously.

Secondly, in order to study the effects of the early stopping mechanism, the algorithm was run repeatedly on various datasets with and without early stopping. The resulting boxplots can be seen in figures 2.4 and 2.5. From these, it can be observed that early stopping may improve or degrade the temporal performance depending on the situation. This is consistent with the theory as using early stopping adds some overhead to compute the prevalences on every node but may reduce the overall execution time by cutting of branches. In particular, early stopping

[3]The interactions and instance are assumed to be sorted.

displays an improvement when equal-width binning is used on genomic datasets such as Alon and Lymphoma. This is most likely caused by the fact that less informative interactions are being returned as the method is unsupervised. The interactions will thus fail the early stopping check more often and more branches will be cut off. The plots also confirm that datasets with larger $|F|$ are more costly computationally. Moreover, it should be noted that early stopping often visibly increases the variance in execution time: low times occur when many useless branches are cut off, a situation which may or may not happen depending on the random instance selection process within the algorithm.

Notes on the experiments In order to plot figure 2.5, the genomic datasets were reduced in size using a feature selection method. This was done because of time constraints on the experiments. The feature selection method used computes the Golub weights (cfr. [12]) for all features and keeps the n features with the highest weights. In the experiments below, the value of n was fixed to 250. Additionally, the number of bins for *Equal-width binning* was fixed to $k = \sqrt[3]{N}$.

2.9 Model analysis and Classification performance

In this section, we will analyse the models produced by the *discRIT* algorithm. Various experimental measures will be reported:

- The balanced classification rate (BCR): to measure classification performance.
- Lustgarten's adjusted stability measure [13] (ASM): to measure the stability of the algorithm with respect to which features are selected. Note that this measure may be undefined, in which case NA will be reported instead of a value between -1 and 1.
- For some models, we will also report the average fraction of matches between interactions and instances.

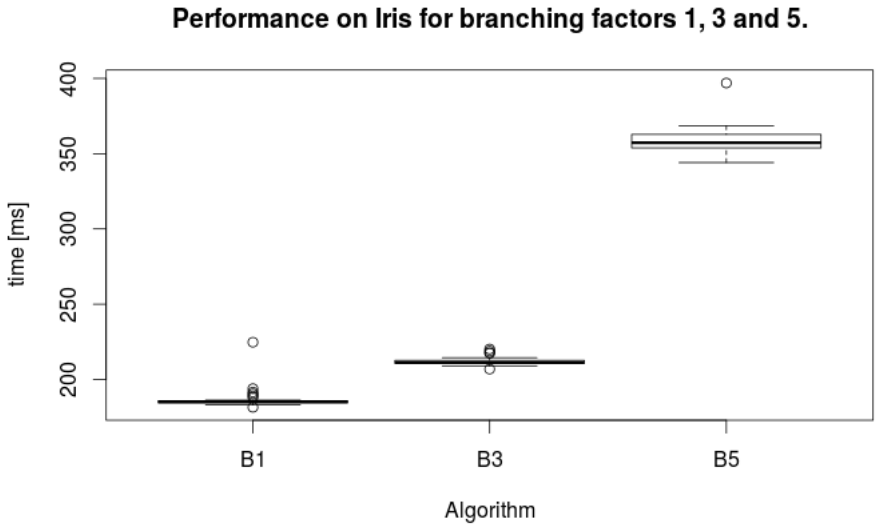
In addition to these measures, some information will be reported about the models' returned interactions. This additional information aims to give more insight into the algorithm's peculiarities as well as unearth various issues which may be the cause of poor classification performance.

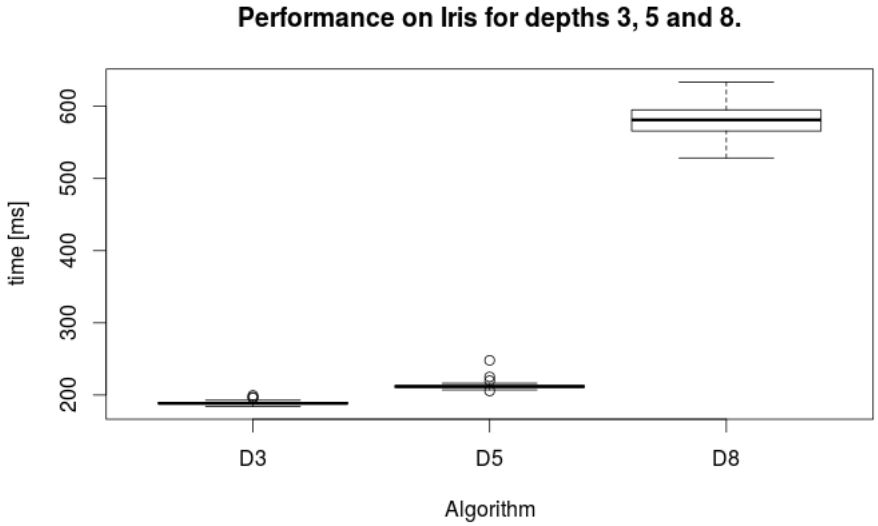
Experimental setup For more information about the experimental setup and measures used, refer to the appendix titled "Experimental setup". Also note that the improved criterion for interaction information (θ_{lower} and θ_{upper}), was not used in the experiments.

2.9.1 Standard datasets with the $\arg\max$ rule

The performance of *discRIT* on standard (non-genomic) datasets with the previously defined $\arg\max$ rule is displayed in table 2.2. From this table, we can observe that:

- The BCR values are comparable to the performance of a naive-bayes classifier (cfr. table 5.2). The only exception is the Tic-tac-toe dataset, for which there is an improvement of 0.42.
- The ASM values are mostly undefined. This indicates that the algorithm ended up using all features. An exception occurs with ChiMerge discretization, which has been shown to cause higher granularity on the Heart and Ionosphere datasets. This is particularly visible on the latter, where the feature selection is unstable.





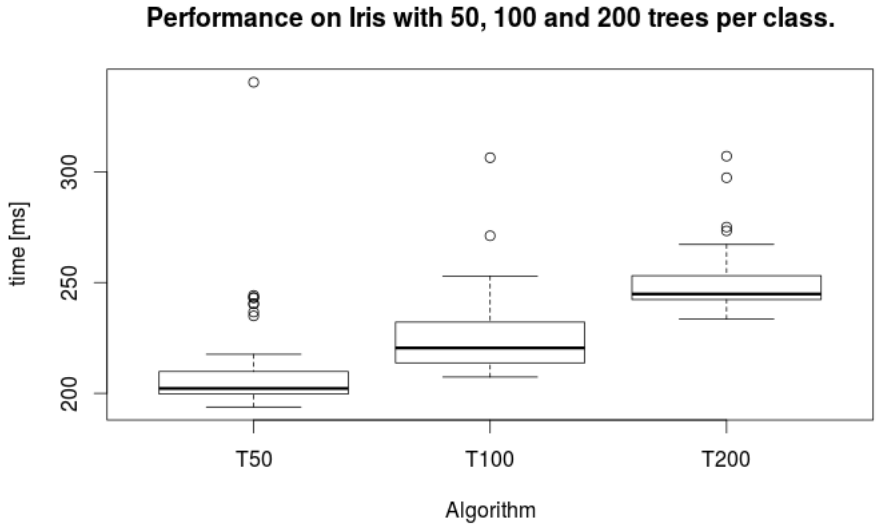


Figure 2.3: Execution time on the iris dataset for various hyper-parameter values without early stopping. Default values are 3, 5 and 100 for the branching factor, the depth and the number of trees respectively.

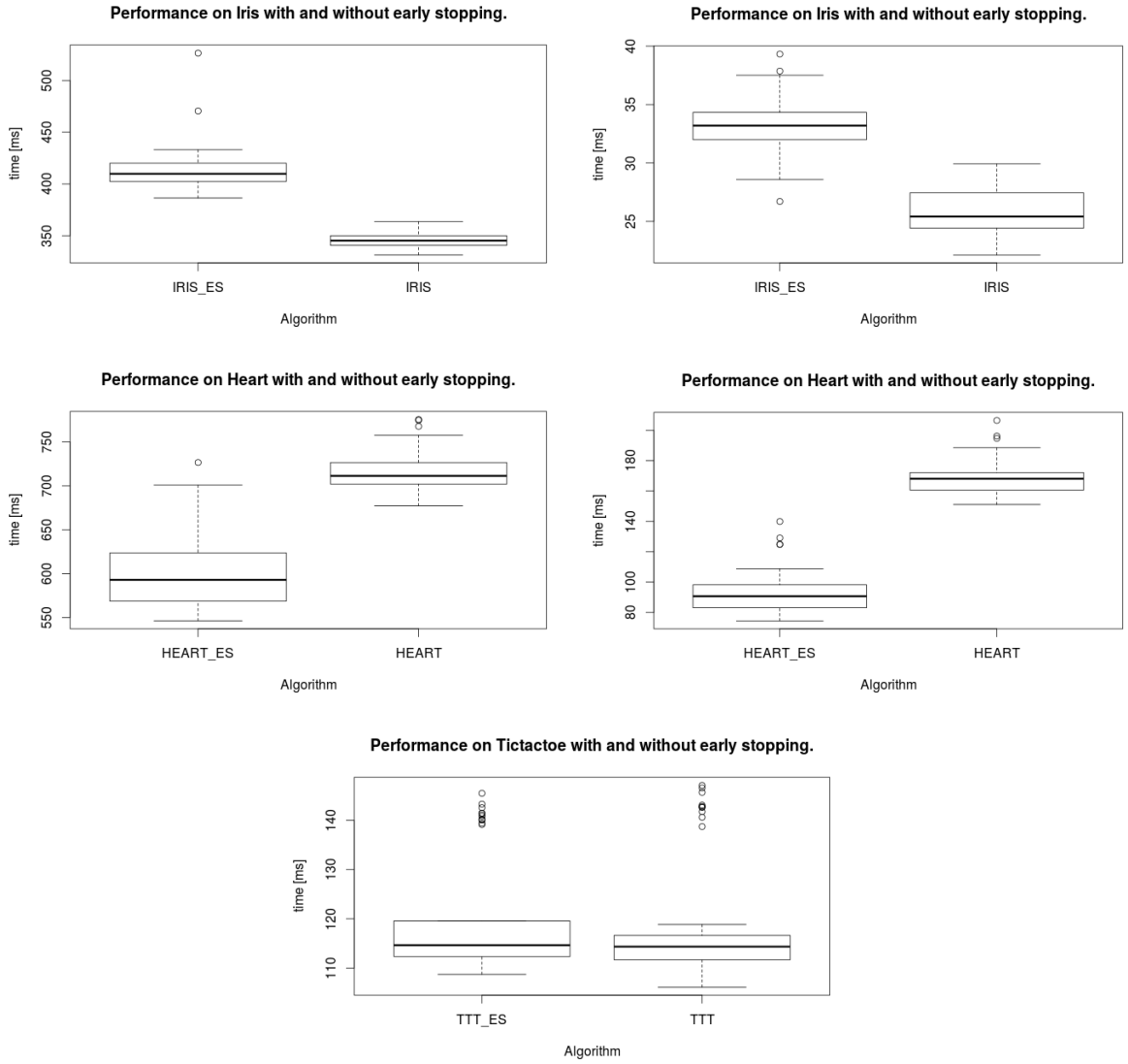


Figure 2.4: Execution time on various datasets with and without early stopping. The boxplots all use the *Ameva* method with the except the ones on the right, which use equal-width binning.

arg max rule	Ameva			CAIM			ChiMerge			equal-width		
Dataset	BCR	ASM	Matches	BCR	ASM	Matches	BCR	ASM	Matches	BCR	ASM	Matches
Iris	0.94	NA	0.176	0.93	NA	0.176	0.94	NA	0.15	0.91	NA	0.1
Heart	0.8	NA	0.2	0.81	NA	0.21	0.81	0.92	0.2	0.83	NA	0.2
Tic-tac-toe	0.96	NA	0.009	0.96	NA	0.009	0.96	NA	0.009	0.96	NA	0.009
Ionosphere	0.88	NA	0.4	0.91	NA	0.36	0.85	0.126	0.13	0.77	NA	0.24

Table 2.2: Experimental results: BCR, ASM and the fraction of interactions that match. The algorithms are *discRIT* with various discretization methods and using the arg max rule. The number of bins for equal-width binning was set to $\sqrt[3]{N}$ where N is the number of instances.

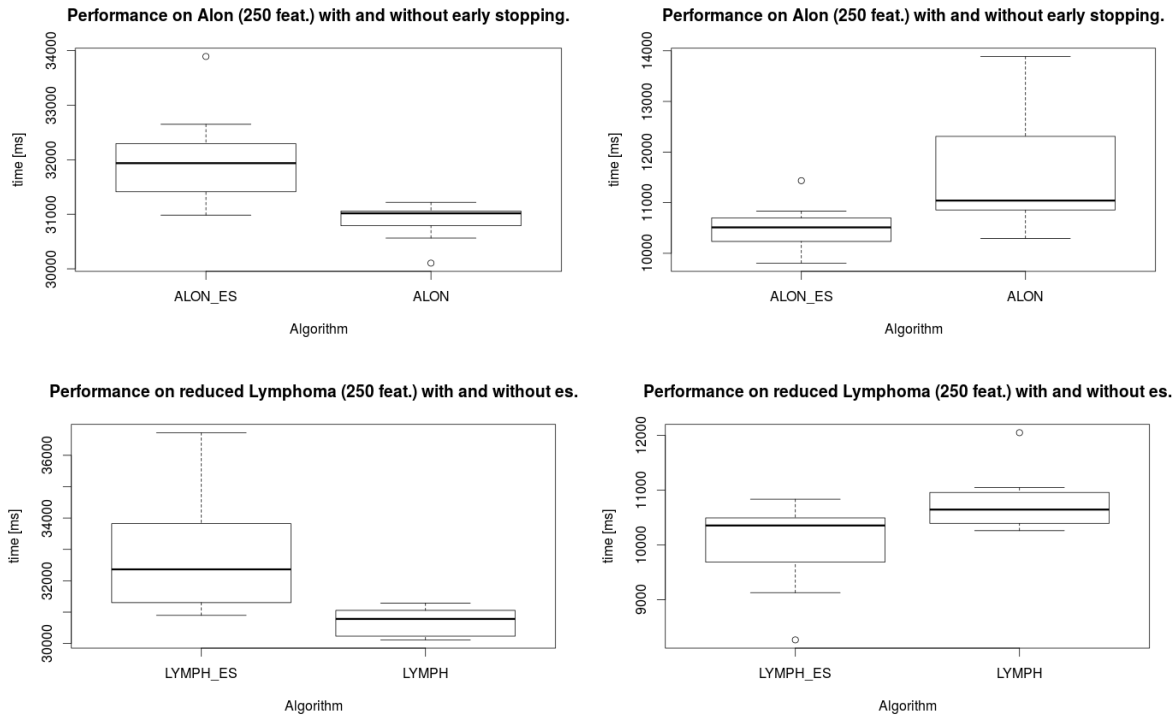


Figure 2.5: Execution time on various datasets with and without early stopping. The boxplots all use the *Ameva* method with the except the ones on the right, which use equal-width binning.

2.9.2 Genomic datasets with the arg max rule

The performance of *discRIT* on genomic datasets with the previously defined arg max rule is displayed in table 2.3. Note that the focus was placed on the difference between supervised and unsupervised discretization, which is why only Ameva and equal-width binning are included. From this table, we can observe that:

- The BCR values are much lower for Ameva. This is the result of low granularity combined with the high-dimensional character of genomic datasets. Low granularity and high dimensionality result in few instances being able to match interactions, which results in poor classification performance.
- The ASM values imply a certain level of stability in the feature selection. For the most part, equal-width binning boasts the greater stability. This is most likely due to less features being used as more of them are removed from the interactions thanks to the higher granularity.

Note on Golub and West datasets Due to a limited computational budget, both aforementioned datasets were reduced to 4000 features using golub weights [12].

Effect of granularity on interaction matches.

As previously stated, a discretization scheme's granularity is an indication of how many intervals the continuous features are divided into. High granularity means many small intervals and causes high sparsity in the binarized datasets. Low granularity means a few larger intervals and causes low sparsity.

arg max rule	Ameva			equal-width		
Dataset	BCR	ASM	Matches	BCR	ASM	Matches
Alon	0.51	0.39	0.04	0.75	0.76	0.47
Lymph	0.5	0.38	0	0.74	0.38	0.37
Golub	0.5	0.46	0	0.73	0.85	0.33
West	0.5	0.36	0	0.51	0.77	0.02

Table 2.3: Experimental results: BCR, ASM and the fraction of interactions that match. The algorithms are *discRIT* with various discretization methods and using the arg max rule. The number of bins for equal-width binning was set to $\sqrt[3]{N}$ where N is the number of instances.

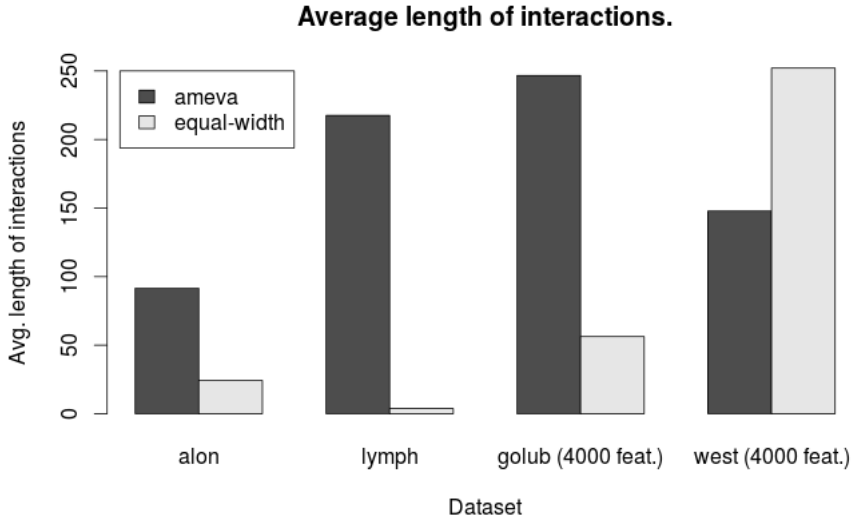


Figure 2.6: The average length of interactions with Ameva and equal-width binning. The number of bins for equal-width binning was set to $\sqrt[3]{N}$ where N is the number of instances.

The granularity of the chosen discretization method has a visible impact the interaction search, the resulting model and, consequently, the classification performance. When the classifier checks if an instance matches an interaction, the instance is in fact compared to a conjunction of constraints.

- High granularity tends to reduce the number of constraints in the conjunction. This is because features with high granularity will be easier to remove during the interaction search. On the other hand, the intervals created by the discretization will be smaller.
- Low granularity tends to increase the number of constraints in the conjunction. This results in larger interactions. The intervals created by the discretization will be larger.

Choosing between high and low granularity is balancing act. Indeed, in the first case, interactions might be easier to match because of their reduced length but they might also become more difficult to match because of the smaller intervals associated to the individual features. In genomic datasets, it is usually preferable to have high granularity because of the high dimensionality and the consequently larger interactions.

Figure 2.6 shows the variation in interaction length for Ameva and equal-width binning. Note how there is a direct correlation with the number of matches and the BCR in table 2.3.

FS rule	Ameva		CAIM		ChiMerge		equal-width	
Dataset	BCR	ASM	BCR	ASM	BCR	ASM	BCR	ASM
Iris	0.95	NA	0.95	NA	0.95	NA	0.95	NA
Heart	0.84	NA	0.84	NA	0.84	0.92	0.84	NA
Tic-tac-toe	0.64	NA	0.64	NA	0.64	NA	0.64	NA
Ionosphere	0.84	NA	0.84	NA	0.83	0.3	0.84	NA

Table 2.4: Experimental results: BCR, ASM. The algorithms are *discRIT* with various discretization methods and using the feature selection rule. The number of bins for equal-width binning was set to $\sqrt[3]{N}$ where N is the number of instances.

Interaction information

In addition to the granularity issue, it is pertinent to analyse the amount of information that the interactions carry. On datasets that only have two distinct class label, a simple and slightly naive way to measure this is to compute the difference between the prevalences of the two classes for each interaction: $\theta_1 - \theta_0$.

From a theoretical standpoint, supervised discretization schemes should result in more informative interactions being discovered than unsupervised methods. This is confirmed by the histograms in figure 2.7. Ameva results in high frequencies near 1 and -1 and equal-width binning boasts high frequencies closer to the center of the histograms.

2.9.3 Feature selection

In order to use *discRIT* as a pure feature selection algorithm, we make use of the following idea: if a feature never or rarely appears in any returned interaction, it likely is irrelevant and can be removed. We therefore count the number of times each feature appears in an interaction and remove the features that never appear in any. A Naive-Bayes classifier can then be applied on the filtered dataset.

ASM computation Note that the method used for the feature selection is identical to the one used to compute the stability indexes $S_A(s_i, s_j)$ and the ASM. For more details on the ASM computation, refer to the appendix titled "Experimental setup".

As previously mentioned in subsection 1.1.3, there are strong theoretical arguments against the use of *Random Intersection Trees* in this fashion. In particular,

- Highly correlated features are likely to be processed similarly during the interaction search. This means that both of them will be either selected or rejected. In the case where they are both selected, the algorithm fails to eliminate redundancy between features.
- Near-zero variance features are typically irrelevant with respect to the class label. Since the algorithm searches interactions on a class by class basis, near-zero variance features will most likely be kept. The algorithm thus fails to eliminate some irrelevant features.

Despite this however, experiments were performed and the results can be viewed in tables 2.4 and 2.5. The results are mostly comparable to a pure Naive-Bayes classifier (cfr. table 5.2) with some slight variations in BCR for some genomic datasets.

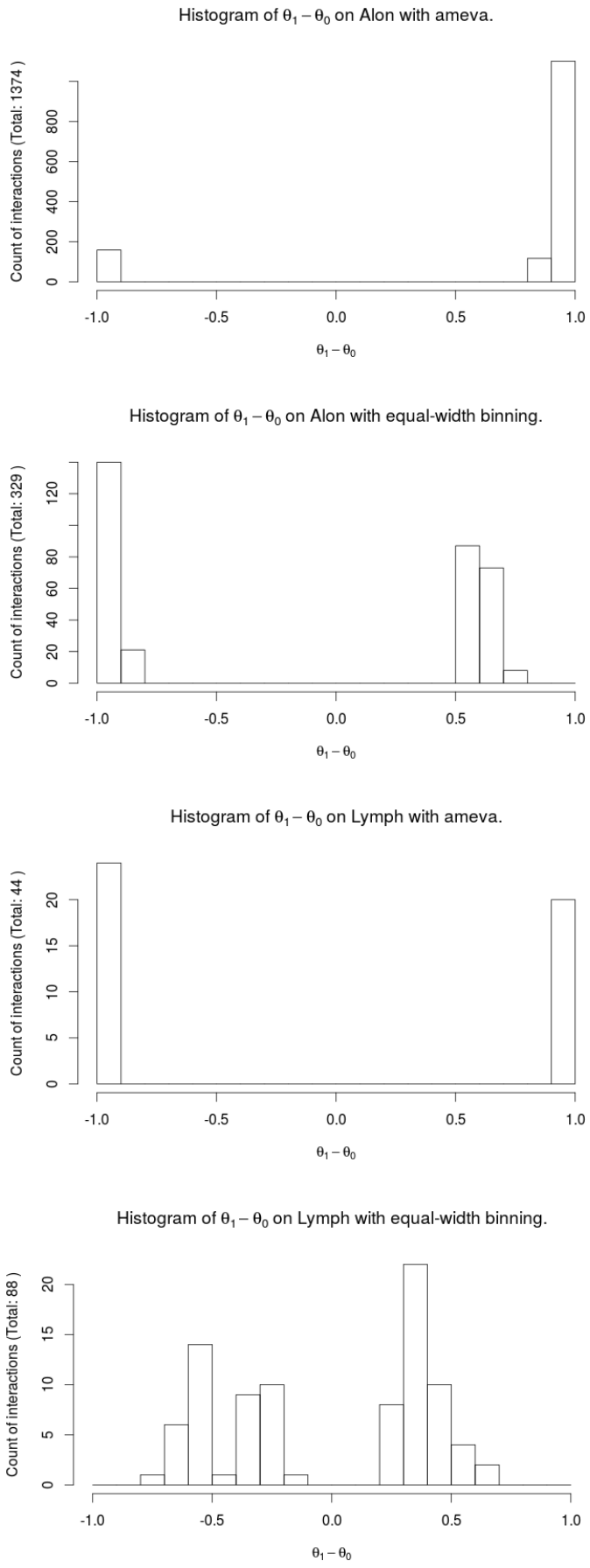


Figure 2.7: Frequency histograms for $\theta_1 - \theta_0$ on Alon and Lymphoma datasets with Ameva and equal-width binning.

FS rule	Ameva		equal-width	
Dataset	BCR	ASM	BCR	ASM
Alon	0.63	0.38	0.71	0.76
Lymph	0.91	0.38	0.75	0.36
Golub	0.99	0.46	0.97	0.85
West	0.42	0.36	0.55	0.76

Table 2.5: Experimental results: BCR, ASM. The algorithms are *discRIT* with various discretization methods and using the feature selection rule. The number of bins for equal-width binning was set to $\sqrt[3]{N}$ where N is the number of instances.

Chapter 3

Coverage-based Random Intersection Trees

Contents

3.1 Motivation

In this chapter, we introduce a new variant of *Random Intersection Trees*. The motivation behind this variant is to remove the necessity for a discretization step such as the one in *discRIT*. Throughout the remainder of this report, this new algorithm will be referred to as *covRIT*.

3.2 Interaction search

covRIT works by dynamically growing intervals or sets of values during the interaction search instead of fixing them during a preprocessing step. Sets of values are used for categorical features while intervals are used for continuous features. From a local point of view in the search, an intersection operation adjusts the interval or set of values for the features of the parent interaction. An intersection may also remove features under certain circumstances. A typical situation in which a feature might be removed is when its interval or set covers many of that feature's values. Feature removal is controlled by two hyper-parameters: ϵ_{cont} for continuous features and ϵ_{cat} for categorical features. These two hyper-parameters play a role analogous to discretization granularity in *discRIT*.

The interaction search used in *covRIT* is exactly the same as the one shown in algorithm 4. The difference lies in how the intersections are computed. The pseudo-code for the new intersection can be viewed in algorithm 5.

Algorithm 5 New intersection for *covRIT* algorithm.

Consider I_1 the parent interaction and I_2 the instance with which I_1 will be intersected. Note that each element of I_1 should contain information on which feature of I_2 it corresponds to in addition to the interval or set of values that it uses. Let ϵ_{cat} and ϵ_{cont} be real numbers in $[0, 1]$ that are used to determine when categorical or continuous features should be removed. Finally, let $span[f]$ be the span of feature f: $max(f) - min(f)$ if f is continuous, the number of different values if f is categorical. The span should be computed on the training set.

procedure $update(f, v, span)$
  if f is continuous **then**
    if $v < f.interval.min$ **then**
      $f.interval.min = v$
    end if
    if $v > f.interval.max$ **then**
      $f.interval.max = v$
    end if
    if $f.interval.max - f.interval.min > span \cdot \epsilon_{cont}$ **then**
      Mark that f should be removed.
    end if
  else
    if v not in $f.set$ **then**
      $f.set = f.set \cup v$
    end if
    if $f.set.size() > (span - 1) \cdot \epsilon_{cat} + 1$ **then**
      Mark that f should be removed.
    end if
  end if
end procedure

function $intersect(I_1, I_2)$
  Clone I_1 into I_3.
  for $f = 1$ **to** $length(I_1)$ **do**
    $i = I_1[f].idx$
    $value = I_2[i]$
    $update(I_3[f], value, span[f])$
  end for
  Remove from I_3 the elements that have been marked for removal.
  return I_3 the intersection of I_1 and I_2.
end function

arg max rule	Early Stopping			No Early Stopping		
Dataset	BCR	ASM	Matches	BCR	ASM	Matches
Iris	0.95	NA	0.1	0.92	NA	0.1
Heart	0.81	NA	0.17	0.81	NA	0.18
Tic-tac-toe	0.97	NA	0.004	0.96	NA	0.005
Ionosphere	0.75	NA	0.23	0.75	NA	0.23

Table 3.1: Experimental results: BCR, ASM and the fraction of interactions that match. The algorithms are *covRIT* with and without early stopping. These results are based on the arg max rule (cfr subsection 3.3.2).

3.3 Similarities with *discRIT*

3.3.1 Prevalences and Early Stopping

Despite the differences with *discRIT*, the use of prevalences and early stopping is still pertinent. Indeed, the interactions returned by the algorithm follow the same logic as in *discRIT* and behave as binary features would[1]. It follows that the same criterion on the prevalences may be used to determine if an interaction is informative. Another consequence of this is that classifiers that may be used with *discRIT* may also be used with *covRIT*.

Furthermore, it can be observed that each intersection either removes a feature, extends its associated interval (or set) or does not modify it in any way. From this, it can be deduced that theorem 1.1 is valid for *covRIT* and that early stopping can still be used.

Experimental results with and without early stopping are available in table 3.1.

3.3.2 Classifier

As stated previously, the interactions returned by *covRIT* follow the same logic as in *discRIT*. We can therefore use the same classification methods. The main classification rule that we used in our experiments remains identical:

$$Class(a) = \arg\max_{C} \log(\mathbb{P}(C)) + \sum_{I:\ a \text{ matches } I} \log(\mathbb{P}_n(I|C))$$

in which a is the instance whose class label needs to be predicted.

3.3.3 Algorithm Interface

covRIT can be controlled by various hyper-parameters, most of which are inherited from *discRIT*:

- θ a vector whose length is equal to the number of classes. This hyper-parameter contains the prevalence thresholds that are used to determine if an interaction is informative or not. If the improved criterion is used, there are two vectors (θ_{lower} and θ_{upper}) instead.
- n_{tree} is the number of intersection trees per class.
- B is the branching factor used in the trees.
- D is the maximum depth for the trees.
- M is the name of the discretization method used for continuous features.

[1] Match or no match

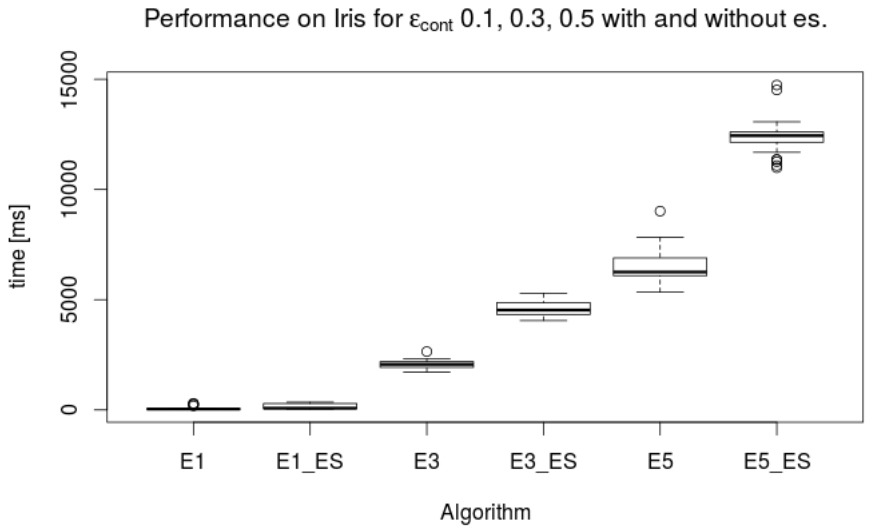


Figure 3.1: Execution time for *covRIT* on Iris with different values of ϵ_{cont} with and without early stopping.

- Z is the minimum interaction size allowed.
- ϵ_{cont} and ϵ_{cat} to control feature removal in continuous and categorical features respectively.

3.3.4 Time Complexity

Under some assumptions, the time complexity bound computed for *discRIT* remains valid for *covRIT* with the following exceptions:

- There is no discretization as preprocessing.
- Since the encoding described in subsection 1.2.2 cannot not be used, the min-wise hashing estimator for prevalences cannot be used either. Indeed, in *discRIT*, the feature encoding could be used because the intervals and sets of values were determined during the preprocessing step and remained constant afterwards.

The assumptions necessary to make this conclusion are that:

- The operations on the interval of values used by continuous features should be in $\mathcal{O}(1)$.
- The operations on the set of values used by categorical features to find or add an element should also be in $\mathcal{O}(1)$.

Influence of ϵ_{cont} and ϵ_{cat}

Although it is not shown in the worst-case time complexity bounds, the algorithm will be slower to terminate if the interactions are longer. Indeed, the intersection between an interaction and a selected instance from the training set can be computed in linear time w.r.t the interaction's length. Additionally, since ϵ_{cont} and ϵ_{cat} control feature removal, it is expected that execution time will increase with them as they increase. Figure 3.1 confirms this but makes no statement about the difference caused by early stopping. Another (less glaring) example can be seen in figure 3.2.

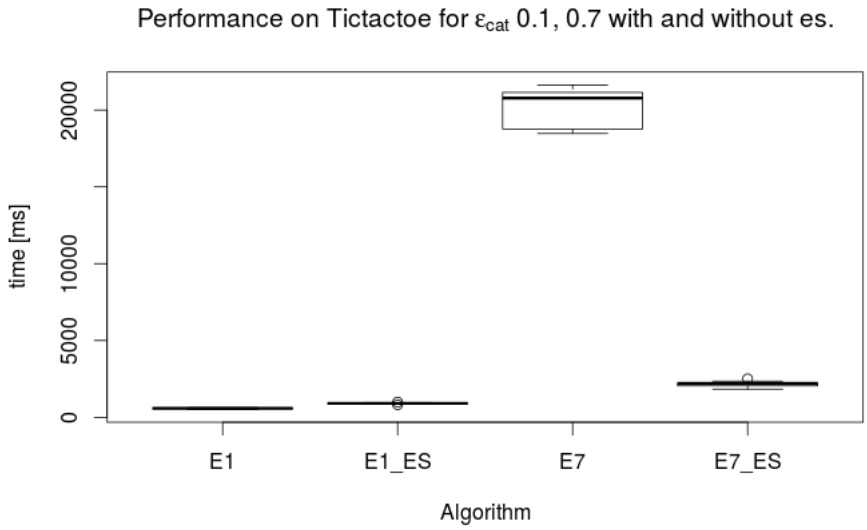


Figure 3.2: Execution time for *covRIT* on Tic-tac-toe with different values of ϵ_{cat} with and without early stopping.

	arg max			Feature selection	
Dataset	BCR	ASM	Matches	BCR	ASM
Iris	0.95	NA	0.1	0.95	NA
Heart	0.81	NA	0.17	0.84	NA
Tic-tac-toe	0.97	NA	0.004	0.64	NA
Ionosphere	0.75	NA	0.23	0.84	NA
Alon	0.65	0.5	0.16	0.66	0.5
Lymph	0.43	0.48	0.15	0.88	0.48
Golub	0.5	0.73	0	0.98	0.73
West	0.5	0.63	0	0.63	0.63

Table 3.2: Experimental results: BCR, ASM and the fraction of interactions that match. The algorithms are *covRIT* with the arg max rule and with the feature selection rule.

3.4 Model analysis and Classification performance

In this section, we will analyse the models produced by the *covRIT* algorithm. Various experimental measures will be reported: BCR, ASM and the average fraction of matches between interactions and instances. All results are available in table 3.2. Refer to the appendix titled "Experimental setup" for more details.

Note on Golub and West dataset Due to memory constraints and the large number of returned interactions, the hyper-parameter corresponding to the depth of the interaction search had to be restricted to lower values. This restriction increased the overall length of the interaction and decreased the size of the grown intervals. The consequence of this is a zero match count and poor classification performance. The various problems caused by the high numbers of returned interactions are discussed later in this chapter.

3.4.1 Standard datasets with the arg max rule

The performance on standard datasets is comparable to that of *discRIT* regardless of the discretization method used. Similarly to the aforementioned algorithm, the reported BCR values

match what would be expected of a Naive-Bayes classifier with the exception of the Tic-tac-toe dataset. Furthermore, the reported ASM values are all undefined, meaning that *covRIT* relies on all the features in its returned interactions.

3.4.2 Genomic datasets with the $\arg\max$ rule

When compared to *discRIT* with equal-width binning, a sharp drop in BCR can be observed on datasets Alon, Lymphoma and Golub. Our theory is that this discrepancy is due to an uneven dependency distribution between the various interactions that are matched to each instance.

Uneven dependency distribution

According to [14], the performance of Naive-Bayes classifiers heavily depends on whether or not the distribution of dependencies between features is even or uneven across classes. At the most basic level, Naive-Bayes assumes that all features are independent and classifies instances with an $\arg\max$ rule based on that assumption. On most datasets, this assumption is almost always violated but Naive-Bayes often manages to retain decent if not excellent classification performance regardless.

When two features are highly dependent, Naive-Bayes essentially counts the same information twice in its $\arg\max$ rule. This may result in skewed classification. But if the dependency between those two features is distributed evenly across all classes, the information is duplicated similarly for all class labels in the $\arg\max$ rule and the classification remains unaffected by the dependency.

While the classifier used with *covRIT* is not strictly a Naive-Bayes classifier, it is similar enough that the same reasoning about uneven dependency distributions can be applied. The following observations can be made:

- The interaction space for *covRIT* is potentially very rich. A lot of interactions that are all dependent, redundant or even approximately identical but with slight variations may be returned. This is because discretization intervals are grown during the interaction search and are thus subject to the algorithm's random instance selection.
- Even if we were to assume the even distribution of dependencies amongst all returned interactions, there would be no guarantee of it being preserved in the classification rule. Indeed, the slight variations in the grown intervals would further compound the issue by counting some interactions but not others.
- This issue can be guessed at by observing the number of returned interactions. Comparing those numbers (see figure 3.3) with the ones obtained with *discRIT* is particularly telling.

Uneven dependency distribution in *discRIT* While *discRIT* might also suffer from this issue to some extent, it is nowhere as susceptible to it. Indeed, since intervals are fixed in a preprocessing step instead of grown dynamically, the interaction space is usually much more restricted. More of the dependent interactions will end up being perfectly identical and it is then trivial to remove the duplicates before applying the classification rule.

Interaction information

In order to analyse the information carried by the returned interactions, frequency histograms of the discrepancy between prevalences were plotted. Those histograms can be viewed in figure 3.3. There are two main observations to be made:

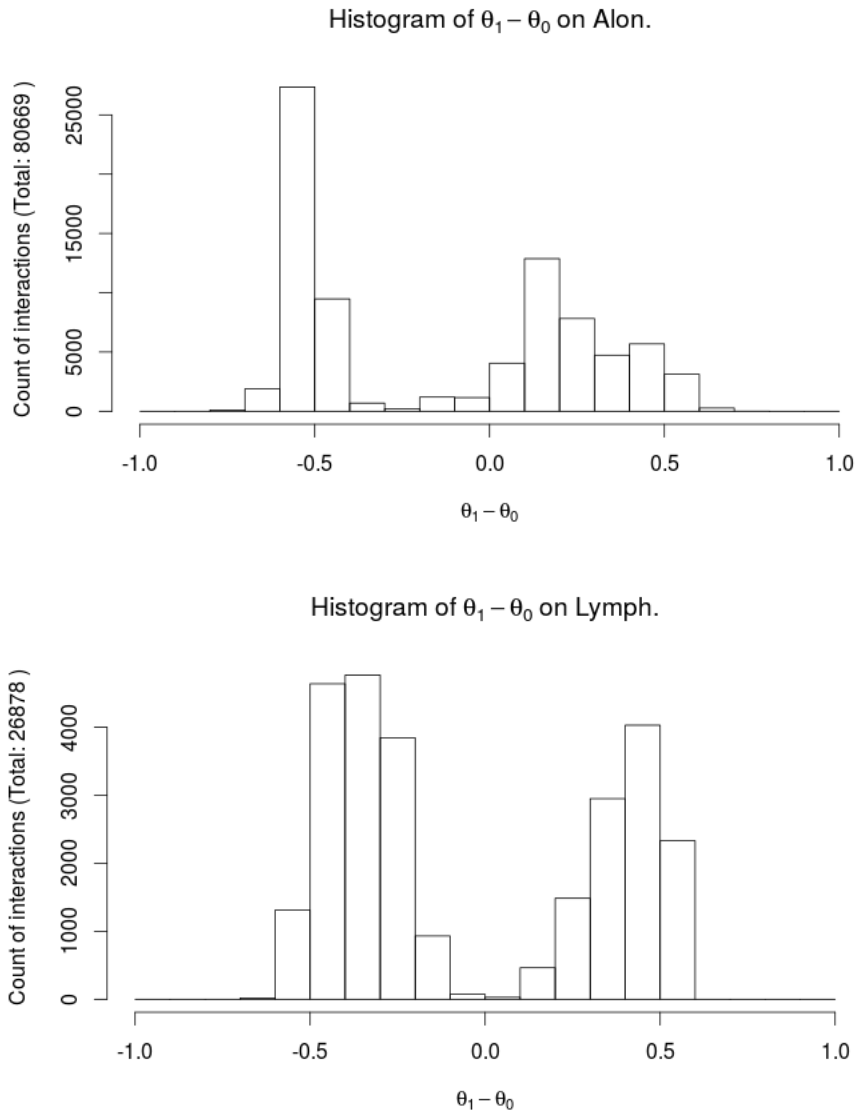


Figure 3.3: Frequency histograms for $\theta_1 - \theta_0$ on Alon and Lymphoma datasets.

- When compared to *discRIT*, the distribution of frequencies is closer to what was obtained with equal-width binning rather than Ameva. This is logical as *covRIT* does not refer to the various class labels to determine when an interval should be grown or when the corresponding feature should be removed.
- The numbers of returned interactions displayed in figures 3.3 and 2.7 are vastly different. The much larger numbers for *covRIT* are a consequence of the richer interaction space. This is an important drawback of the *covRIT* algorithm as some models on larger datasets might consist of prohibitively large numbers of interactions unless an additional filter is used. Those large numbers also result in the classification rule taking more time to execute as its time complexity is linear with respect to the number of interactions in the model. Furthermore, taken to its logical extreme, this phenomenon might result in memory problems.

Notes on stability

The ASM values reported in table 3.2 suggest that there is some measure of stability in the feature selection process. At first glance, this may seem somewhat surprising as the intervals and sets grown during the interaction search depend on the random instance selection process. However, it can be observed that, as an interval or set grows, the probability that it will be grown again later in the search decreases. The bigger the growth in the interval, the bigger the loss of probability. With appropriate hyper-parameter values [2], an interval often stabilizes as it approaches the full span of values for the corresponding feature in the currently explored class.

3.4.3 Feature selection

Using *covRIT* as a pure feature selection method can be done in the exact same way as with *discRIT*. The theoretical arguments that stand against this use of the algorithm are also still applicable. The results are mostly comparable to a pure Naive-Bayes classifier (cfr. table 5.2) with some slight variations in BCR for some genomic datasets.

[2] in particular large enough ϵ_{cont} and ϵ_{cat} values

Chapter 4

Relaxed Random Intersection Trees

Contents

4.1 Motivation

The motivation behind the use of *Relaxed Random Intersection Trees* is the same as *covRIT*: the removal of the discretization as a preprocessing step. Unlike *covRIT* however, *Relaxed Random Intersection Trees*, which will also be called *relRIT*, does not discretize the dataset implicitly during the interaction search.

The algorithm *relRIT* uses a new type of intersection in which the successive differences between feature values are cumulated and the features themselves are removed when the cumulated differences cross a given threshold.

Instead of being patterns that can be matched by instances, the returned interactions are more similar to activation functions in RBFN.

4.2 Interaction search

The interaction search used in *relRIT* is exactly the same as the one shown in algorithm 4. The difference lies in how the intersections are computed. To do so, we define a new type of intersection.

The rough idea is to allow some difference in the feature values when intersecting a parent interaction with an instance. The successive errors for each feature are cumulated while growing a branch of an intersection tree and, once their total exceeds a given threshold, the feature is removed. Additionally, the thresholds for the various features should be scaled. This new type of intersection is depicted in algorithm 6.

Another way to look at this new intersection is that, when growing a branch of the intersection tree, the features that are removed are the ones that vary too much according to a random sampling of instances.

As for the interactions, they are still represented by feature-value pairs (f_i, v_i). Unlike *discRIT* where the v_i are intervals or single categorical values and *covRIT* where they are intervals or sets of values, the values v_i in *relRIT* are always single values. Those values are fixed when the intersection tree's root node is selected among the instances of the training set.

Algorithm 6 Relaxed intersection for *relRIT* algorithm.

Consider I_1 the parent interaction and I_2 the instance with which I_1 will be intersected. Furthermore, let ϵ_f be the scaled threshold value for feature f. Note that I_1 contains the state of the cumulative error counters. Also, each element of I_1 should contain information on which feature of I_2 it corresponds to.

```
function diff(f, v)
    if f is continuous then
      return (f.value − v)^2
    else
      if f.value = v then
            return 1
      else
            return 0
      end if
    end if
end function

function intersect(I1, I2)
    Clone I1 into I3.
    Initialize empty list of integers removed.
    for f = 1 to length(I1) do
      i = I1[f].idx
      I3[f].count = I1[f].count + diff(I1[f], I2[i])
      if I3[f].count > ε_f then
            removed.add(f)
      end if
    end for
    Remove from I3 the elements at indexes specified in removed.
    return I3 the intersection of I1 and I2.
end function
```

4.2.1 Threshold scaling

The idea behind threshold scaling is that a given difference d between two values may be significant for a given feature f_1 while it may be insignificant for another feature f_2. Indeed, a possible feature f_1 could have values in $[0, d]$ while another feature f_2 could have values in $[0, 1000 \cdot d]$.

The difference d is much more significant for the former feature and less so for the latter.

Concretely, if ϵ_{cat} and ϵ_{cont} depict two user-defined values for the error thresholds and v_{root} depicts the corresponding feature value chosen by the root interaction, it is possible to scale the thresholds as follows:

$$\epsilon_{cat,f} = \epsilon_{cat} \cdot var_f$$
$$\epsilon_{cont,f} = \epsilon_{cont} \cdot var_f$$

where var_f is a measure of variance with respect to v_{root} for feature f and limited to the instances of the current class in the interaction search. For continuous features, this could be the actual sample variance. For categorical features, this could be the fraction of instances where $f \neq v_{root}$.

4.3 Similarity measures

Since, in *relRIT*, the interactions can no longer be matched by instances, prevalence probabilities can no longer be used and need to be replaced. Indeed, prevalence probabilities can only be computed if the interactions behave as binary features would (i.e. an instance either matches or does not match an interaction). In *relRIT* however, interactions can be said to act as continuous features whose values depend on the distance between instance and interaction. In order to keep the algorithm logic as close to *discRIT* and *covRIT* as possible, a similarity measure bound between 0 and 1 was chosen to replace the prevalence probabilities. The similarity measure chosen is computed as follows:

- As a preprocessing step, the continuous features are scaled using min-max scaling. This results in features bound between 0 and 1.
- If I is an interaction and Obs an instance, $Dist(I, Obs) = \sum_{f \in I} (I_f - Obs_f)^2$ for continuous features f.
- For categorical f where $I_f = Obs_f$, we replace the corresponding term by 0.
- For categorical f where $I_f \neq Obs_f$, we replace the corresponding term by 1.
- The similarity measure is defined as $Sim(I, Obs) = \exp(\frac{-Dist(I,Obs)}{r})$ where $r > 0$.
- As a replacement for the prevalence probabilities, we use

  $$Pres(I, C) = \frac{1}{n_C} \sum_{Obs:\ class(Obs)=C} Sim(I, Obs)$$

  where n_C is the number of instances in class C.

Apart from the replacement of prevalences by similarity measures, the criterion used to determine whether an interaction is informative or not remains the same:

$$\exists C : Pres(I, C) > \theta_C \tag{4.1}$$
$$\exists C : Pres(I, C) < \theta_C \tag{4.2}$$

or, if the improved criterion[1] is used,

$$\exists C : Pres(I, C) > \theta_{lower,C} \tag{4.3}$$
$$\exists C : Pres(I, C) < \theta_{upper,C} \tag{4.4}$$

[1]Note that this improved criterion was unused in the simulations and experiments. It is described here for future research.

arg max rule	Early Stopping		No Early Stopping	
Dataset	BCR	ASM	BCR	ASM
Iris	0.95	NA	0.95	NA
Heart	0.8	NA	0.8	0.47
Tic-tac-toe	0.65	NA	0.65	NA
Ionosphere	0.77	NA	0.76	0.45

Table 4.1: Experimental results: BCR, ASM and the fraction of interactions that match. The algorithms are *relRIT* with and without early stopping. These results are based on the arg max rule (cfr section 4.5).

Mixed distances Due to the absence of any discretization, a special distance measure that is compatible with mixed datasets needs to be used. In our experiments, we limit ourselves to the strategy described above: using min-max scaling on continuous attributes and setting the cost of differences for categorical attributes to 1. This hardly is the only solution however and we believe that it might be useful to evaluate the performance of the algorithm with dedicated mixed distances such as the HVDM distance (cfr. [15]) in future research.

Drawbacks The drawback in using a similarity measure and in the absence of any discretization is that, for any interaction I, there will be some level of similarity and dissimilarity with all instances in the dataset even if, from a logical point of view, some instances are not related to I at all. This results in $Pres(I, C)$ measures that are not as easily interpretable. Reducing r may help mitigate this effect but should not be abused lest all the $Pres(I, C)$ measures approach zero.

4.4 Early Stopping

Since the $Pres(I, C)$ measures boast many similar properties to the prevalence probabilities used previously, early stopping may still be used. Indeed, as the interaction search explores a branch of an intersection tree, the features present in the successive interactions may only be removed. Theorem 1.1 remains relevant and early stopping may still be used.

Experimental results with and without early stopping are available in table 4.1. The discrepancy in ASM for the Heart and Ionosphere datasets is most likely due to the fact that early stopping forces some search branches to return extra interactions. Since those extra interactions are taken from a lower depth, the length is higher. Consequently, more features are covered and thus selected when early stopping is used.

4.5 Classifier

Since interactions play a role similar to RBFN activation functions rather than the role of binary rules, using the same classification rule as in *discRIT* would be a poor choice. Indeed, that rule requires that the algorithm be able to determine whether or not an instance matches an interaction as a binary fact[2]. For *relRIT*, it is therefore necessary to include a notion of similarity

[2]While it is technically possible to apply such binary logic here, interactions would almost never match with new instances.

in the rule. Using the concepts introduced in section 4.3, a possible rule is the following:

$$Class(a) = \arg\max_{C} \log(\mathbb{P}(C)) + \sum_{I} (Sim(I,a) \cdot \log(Pres(I,C))) + \sum_{I} ((1 - Sim(I,a)) \cdot \log(1 - Pres(I,C)))$$

in which a is the instance whose class label needs to be predicted.

In this new rule, we make use of the similarity between instance and interaction as well as the dissimilarity. Indeed, since all instances have some similarity to a given interaction to some extent, there is no reason to exclude dissimilarity the same way the absence/mismatch of interactions was excluded in *discRIT* and *covRIT*.

The attentive reader will also note that, since a similarity can only be zero if the corresponding distance is infinite, $Pres(I,C)$ is never equal to zero in practice. It follows that, unlike the other algorithms, smoothing is not required here unless very small values of r are used.

4.6 Algorithm Interface

Like most other models in machine learning, *relRIT* can be controlled by various hyper-parameters. This section summarizes what those hyper-parameters are.

- θ a vector whose length is equal to the number of classes. This hyper-parameter contains the similarity thresholds that are used to determine if an interaction is informative or not. If the improved criterion is used, there are two vectors (θ_{lower} and θ_{upper}) instead.
- n_{tree} is the number of intersection trees per class.
- B is the branching factor used in the trees.
- D is the maximum depth for the trees.
- Z is the minimum interaction size allowed.
- ϵ_{cat} is the unscaled error threshold for categorical attributes.
- ϵ_{cont} is the unscaled error threshold for continuous attributes.
- r is the radius used to compute the $Pres(I,C)$ values. Additionally, choosing a different r value to compute the weights in the classification rule is also a possibility.

4.7 Time Complexity

Overall, the time complexity analysis of *relRIT* is very similar to *discRIT*'s (cfr. section 2.8).

In a single intersection tree, there are at most $\left(\sum_{i=0}^{D-1} B^i\right) \cdot B$ intersections to compute and each of those may be computed in $\mathcal{O}(|F|)$[3]. Assuming that early stopping is used, the algorithm needs to compute the similarity measures for the classes after each intersection. This results in an additional cost $\mathcal{O}(N \cdot |F|)$ per intersection. The process of building an intersection tree is then repeated $n_{tree} \cdot \#C$ times.

Unlike *discRIT* however, there is no discretization step and using the min-wise hashing estimator is not pertinent since prevalence probabilities have been replaced.

[3]The interactions and instance are assumed to be sorted.

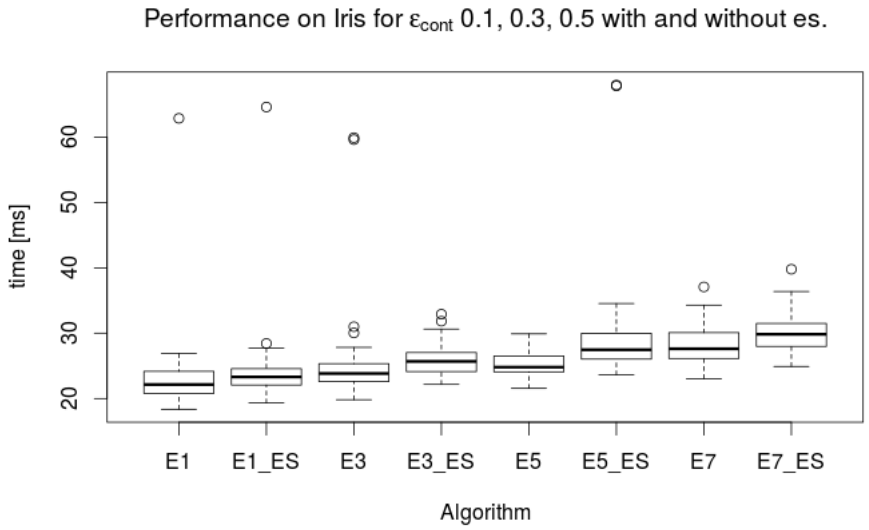


Figure 4.1: Execution time for *relRIT* on Iris with different values of ϵ_{cont} with and without early stopping.

	arg max		Feature selection	
Dataset	BCR	ASM	BCR	ASM
Iris	0.95	NA	0.95	NA
Heart	0.8	NA	0.84	NA
Tic-tac-toe	0.65	NA	0.64	NA
Ionosphere	0.77	NA	0.84	NA
Alon	0.6	0.15	0.61	0.15
Lymph	0.8	0.08	0.92	0.08
Golub	0.81	0.16	1	0.16
West	0.53	0.58	0.57	0.58

Table 4.2: Experimental results: BCR, ASM. The algorithms are *relRIT* with the arg max rule and with the feature selection rule.

4.7.1 Influence of ϵ_{cont} and ϵ_{cat}

Like in *covRIT*, the algorithm will be slower to terminate if the interactions are longer. Indeed, the intersection between an interaction and a selected instance from the training set can be computed in linear time w.r.t the interaction's length. Additionally, since ϵ_{cont} and ϵ_{cat} control feature removal, it is expected that execution time will increase with them. Figure 4.1 somewhat confirms this although the differences in execution time are not as glaring as with *covRIT*.

4.8 Model analysis and Classification performance

In this section, we will analyse the models produced by the *relRIT* algorithm. Two experimental measures will be reported: the BCR and the ASM measures. All results are available in table 4.2. Refer to the appendix titled "Experimental setup" for more details.

4.8.1 Standard datasets with the arg max rule

The performance on standard datasets is comparable to that of *discRIT* and *covRIT*. Similarly to the aforementioned algorithms, the reported BCR match what would be expected of a Naive-

Bayes classifier. The only difference lies in the Tic-tac-toe dataset, where performance has decreased back to the same level as the Naive-Bayes classifier. This is most likely due to the distance metric used in *relRIT* as it does not give enough weight to the differences for categorical features. Furthermore, the reported ASM values are all undefined, meaning that *relRIT* relies on all the features in its returned interactions.

4.8.2 Genomic datasets with the $\arg\max$ rule

By comparing the results for the genomic datasets in tables 4.2 and 5.2, there seems to be a clear correlation between the two sets of BCR values: datasets where Naive-Bayes performs better are the same as those where *relRIT* performs better. Similarly to *covRIT*, the poor performance on some datasets is caused by the uneven distribution of the dependencies between the various interactions (cfr. [14]). Unlike *covRIT* however, the distributions tend to be less distorted by the $\arg\max$ rule.

Uneven dependency distribution

Since *relRIT*'s classification rule is similar to a Naive-Bayes classifier, it is pertinent to link the dependency distributions to the classification performance. Through the various reported measures, it can be observed that there seems to be a correlation between the BCR values for *relRIT* and Naive-Bayes. The former's performance remains slightly inferior to the latter's however. This decrease is likely caused by the distortion that *relRIT* creates in the dependency distributions. In other words, a perfectly even distribution of dependencies between features does not translate to a perfectly even distribution of dependencies between interactions. Nevertheless, the correlated BCR and the fact that highly correlated features are likely to be treated similarly during *relRIT*'s interaction search does indicate that there is some sort of link between the two.

There is also a noticeable difference between the BCR values for *covRIT* and *relRIT*. As mentioned in the previous chapter, *covRIT*'s classification rule tends to distort the dependency distributions further due to the minute variations in the grown intervals. The classification rule for *relRIT* functions differently and does not distort the distributions to the same extent. The reasons for this are that:

- The interaction space is not as rich as with *covRIT*. Indeed, in an interaction, each feature is characterized by a single value taken from the training set. Assuming N different values for a given feature, this means that the number of possible feature values in *relRIT*'s interactions is in $\mathcal{O}(N)$ while the number of possible intervals in *covRIT* is in $\mathcal{O}(N^2)$.
- The classification rule used by *relRIT* does not depend on the minute variations between interactions. Indeed, each interaction is weighted instead of being accepted or rejected. It follows that all the returned interactions play a role in the classification and that, if a high enough radius r is used to compute the weights, dependency distributions tend to be preserved.

Interaction information

In order to analyse the information carried by the returned interactions, frequency histograms of the difference between the $Pres(I, C)$ values were plotted. Those histograms can be viewed in figure 4.2. The following observation is made:

- Most interactions occur near the center of the histogram, unlike *discRIT* and *covRIT*. This is a consequence of using a distance metric and an associated similarity measure to replace the original prevalence probabilities. Indeed, regardless of the class labels the instances belong to, there will necessarily be some similarity between an interaction and an instance.

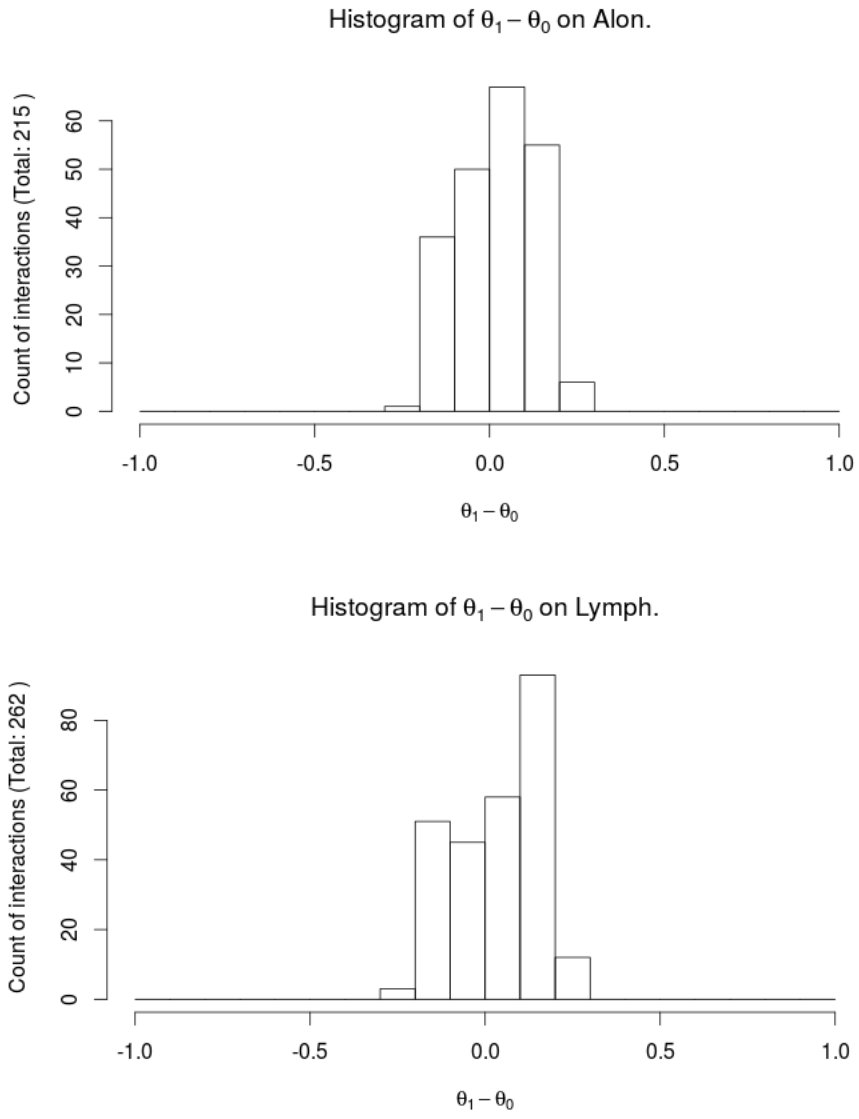


Figure 4.2: Frequency histograms for $\theta_1 - \theta_0$ on Alon and Lymphoma datasets.

This can be inferred from the similarity formula itself: to have a similarity equal to or near 0, it is necessary to have a distance equal to or near $+\infty$ which is a rare occurrence.

Influence of radius Note that reducing the radius used to compute the $Pres(I, C)$ values might affect the above histograms and help make the interactions more informative in general. However, such an improvement also depends on the dataset used and the distributions of its features' values.

Notes on stability

The ASM values reported in table 4.2 suggest that there is very little stability in the feature selection process with the exception of the west dataset. Of the three variants of *Random Intersection Trees* proposed in this report, this is the most unstable one. This is because feature removal during the interaction search is much more dependent on the random instance selection than in *discRIT*. Even compared to *covRIT*, *relRIT* is less stable as the former relies on interval

growth as one of the main reasons for its stability.

To illustrate this, consider two consecutive intersections in a branch of the interaction search. Assume that the same instance is picked both times. Assuming that the first intersection does not remove the feature, the three algorithms will react as follows for the second intersection:

- *discRIT*: the feature is not removed.
- *covRIT*: the feature is not removed.
- *relRIT*: the feature might be removed depending on whether or not the cumulative error for that feature exceeds its threshold.

4.8.3 Feature selection

Using *relRIT* as a pure feature selection method can be done in the exact same way as with the other two algorithms. The theoretical arguments that stand against this use of the algorithm are also still applicable. The results are mostly comparable to a pure Naive-Bayes classifier (cfr. table 5.2) with some slight variations in BCR for some genomic datasets.

Chapter 5

Results Overview

This last chapter contains an overview of the BCR values obtained experimentally for the various algorithms. The overview of BCR for the *Random Intersection Trees* algorithms is displayed in table 5.1. Additionally, the performance of the pure Naive-Bayes classifier is displayed in table 5.2.

	arg max rule			Feature selection		
Dataset	*discRIT*	*covRIT*	*relRIT*	*discRIT*	*covRIT*	*relRIT*
Iris	0.94	0.95	0.95	0.95	0.95	0.95
Heart	0.83	0.81	0.8	0.84	0.84	0.84
Tic-tac-toe	0.96	0.97	0.65	0.64	0.64	0.64
Ionosphere	0.91	0.75	0.77	0.84	0.84	0.84
Alon	0.75	0.65	0.6	0.71	0.66	0.61
Lymph	0.74	0.43	0.8	0.91	0.88	0.92
Golub	0.73	0.5	0.81	0.99	0.98	1
West	0.51	0.5	0.53	0.55	0.63	0.57

Table 5.1: Overview of the BCR values for the *Random Intersection Trees* algorithms.

Dataset	BCR
Iris	0.95
Heart	0.84
Tic-tac-toe	0.64
Ionosphere	0.84
Glass	0.41
Alon	0.65
Lymph	0.91
Golub	0.97
West	0.62

Table 5.2: Classification performance (BCR) for Naive-Bayes classifier.

Conclusion

In this thesis, we proposed three algorithms based on *Random Intersection Trees*. All three algorithms are able to run on datasets with both continuous and categorical features. For each algorithm, a basic classification rule was proposed.

From our experiments on standard (non-genomic) datasets, we conclude that our three algorithms are all comparable in performance to the typical Naive-Bayes classifier. Furthermore, on those relatively low-dimension datasets, we observe that the algorithms usually end up using all of the available features in their interactions.

On genomic datasets however, the performance with the classification rules degrade with respect to the Naive-Bayes classifier, particularly for *covRIT* and *relRIT*. After analysis, we theorize that this degradation in performance is caused by an uneven distribution of dependencies between interactions. *covRIT* suffers the most from this phenomenon because of minute variations in the returned interactions and the nature of the classification rule used.

Additionally, we have identified that *discRIT* may suffer greatly from granularity problems in its discretization scheme. This is particularly visible for supervised discretization methods such as Ameva as they do not always have a customisable parameter to control the granularity. We experimentally observed that when granularity is low, few interactions end up being matched to new instances and the classification rule fails. This phenomenon occurs most often with high-dimension datasets. Therefore, choosing an appropriate discretization method (with a carefully chosen granularity parameter) is one of the keys to good classification performance.

As for the stability of the feature selection on the genomic datasets, we observe that *discRIT* is often the most stable. *covRIT* comes second while *relRIT* is the most unstable.

All three algorithms were also tested as if they were pure feature selection algorithms along with a Naive-Bayes classifier. There are several strong theoretical arguments against such a use: redundant features are managed similarly[1] and irrelevant features with near-zero variance are rarely removed. The experimental results confirm these arguments as there is no significant difference when compared to a Naive-Bayes classifier without any feature selection.

Finally, one of the most attractive properties of *Random Intersection Trees* is the interactions themselves. From the standpoint of knowledge discovery, interactions are useful if they are interpretable as well as informative with respect to the class labels. *discRIT* and *covRIT* are both superior when it comes to the former. Indeed, both of those algorithms return interactions with clear-cut intervals of values. On the other hand, *relRIT* returns interactions that behave in a centroid-like fashion and whose features each carry a single reference value. Additionally, experiments show that *discRIT* remains the best of the three algorithms when it comes to returning informative interactions, especially when a supervised discretization method is used.

[1] both are kept or both are rejected

covRIT ends up second best for informative interactions and *relRIT* ends up third due to its use of distance and similarity measures.

Future Work

The one major issue that remains at the end of this thesis is the uneven distribution of dependencies between interactions. We propose here several possible avenues of research that may help solve this problem:

- Using a classification rule based on decision trees instead of Naive-Bayes would somewhat sidestep the issue as they do not suffer as much from redundancy. Due to the sometimes large number of interactions returned, it might be necessary to perform "interaction selection" to avoid over-fitting.
- Implementing and using the improved prevalence criterion (θ_{lower}, θ_{upper}) could be used to have more control over which interaction are returned. We might then be able to use these hyper-parameters to affect the distribution of dependencies.
- Applying an additional filter to perform "interaction selection". The goals would be the same as standard feature selection: removing redundant and irrelevant elements (here, interactions). An example of a possible solution would be to keep only the n best interactions obtained from each class. The interactions could be graded using symmetrical uncertainty (cfr. [6] and [7]) w.r.t the class label as well as compared to each other if symmetrical uncertainty is computed on pairs of interactions.
- It would be pertinent to embed the "interaction selection" process in the classification rule as removing some of them might reduce the number of matches between interactions and test instances.

Additionally, more experiments could be performed on additional datasets and with more extensive hyper-parameter tuning. More testing on datasets with a high number of class labels would also be beneficial.

Bibliography

[1] Rajen Dinesh Shah and Nicolai Meinshausen. Random intersection trees. *Journal of Machine Learning Research*, 15:629–654, 2014.

[2] M. Özuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Computer Vision and Pattern Recognition*, 2007.

[3] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Image classification using random forests and ferns. In *IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[4] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* 1984.

[6] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the Twentieth International Conference*, 2003.

[7] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical recipes in C.* Cambridge University Press, second edition, 1988-1992.

[8] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[9] L. A. Kurgan and K. J. Cios. Caim discretization algorithm. *IEEE Transactions on knowledge and data engineering*, 16(2):145–153, February 2004.

[10] L. Gonzalez-Abril, F. J. Cuberos, F. Velasco, and J. A. Ortega. Ameva: An autonomous discretization algorithm. *Expert Systems with Applications*, 36:5327–5332, 2009.

[11] Kerber Randy. Chimerge: Discretization of numeric attributes. In *Proceedings of AAAI-92*, 1992.

[12] T. R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caligiuri, C. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

[13] J. L. Lustgarten, V. Gopalakrishnan, and S. Visweswaran. Measuring stability of feature selection in biomedical datasets. In *Symposium Proceedings.* AMIA, 2009.

[14] Harry Zhang. The optimality of naive bayes. In *Proceedings.* 17th International FLAIRS Conference, 2004.

[15] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[16] L. I. Kuncheva. A stability index for feature selection. In *Artificial Intelligence and Applications*, pages 390–395. 25th IASTED International Multi-Conference, February 2007.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve www.uclouvain.be/epl