Master In Computational and Mathematical Engineering

Final Master Project (FMP)

Influence of imbalanced datasets in the induction of Full Bayesian Classifiers

Daniel Morán Jiménez

Dr. Agusti Solanas (supervisor)

15/07/2018

Signature of the director authorizing the final delivery of the FMP:

INDEX CARD OF THE FINAL MASTER PROJECT

Title of the FMP:	*Influence of imbalanced datasets in the induction of Full Bayesian Classifiers*
Name of the author:	*Daniel Morán Jiménez*
Name of the TUTOR:	*Dr. Agusti Solanas*
Name of the PRA:	*Juan Alberto Rodríguez*
Date of delivery (mm/aaaa):	07/2018
Degree:	*Ingeniería Computacional y Matemática*
Area of the Final Work:	*Smart Health*
Language of the work:	*English*
Keywords	*Machine Learning, Bayesian Networks, Imbalance*

Summary of the Work (maximum 250 words): *With the purpose, context of application, methodology, results and conclusions of the work.*

This project consists in three main tasks: first, an analysis of the current state of the art in technologies for dealing with class imbalance problems in machine learning algorithms. Second, the analysis of how this problem actually affects a particular class of statistical models, the Bayesian Classifiers, proposing solutions to the particular problems found. And third, to implement a Bayesian Classifier and develop a series of experiments that would support the assertions of the analysis, and shed more light on how this problem can be dealt with.

Abstract (in English, 250 words or less):

Machine Learning algorithms have received a growing interests in the last years with the increase in computing power and the availability of new data sources coming from digital services and a growing number of connected devices. Concerning the task of classification, one of the problems the field faces is how to deal with datasets where most of the interesting information (examples of the class or classes of interest that have to be classified) is obscured by a greater number of uninteresting examples: this is called the class imbalance problem. Much effort has been put solving this problem, either by fixing the problem modifying the dataset, modifying the particular algorithm used for the learning, or assembling different algorithms to provide imbalance-invariant assemblies.

Bayesian Classifiers are probabilistic graphical models that can be used to efficiently capture the structure of the joint probability distribution that generated a dataset. This work will analyze the impact that imbalanced datasets have in the induction of Bayesian Network models, and try to offer a way to tackle the problem directly in the algorithm, without resorting to modifying the dataset or combining algorithms.

Contents

1 Introduction

1.1 Context and justification of the Work

Inducing efficient classification models for datasets with imbalanced classes is one of the multiple challenges of machine learning implementations. The problem of imbalance arises when the number of instances of one of the classes to classify (hereafter the Majority class) is clearly larger than the total number of observations of the opposite one (the Minority class). In this situation, the Majority class tends to be overrepresented in the predictions of the induced clasifier. This situation can also arise in multiclass classification problems where one or more classes overrepresented in the input data, may lead to an underrepresentation of the minority classes.

As a quick overview of the literature on Machine Learning will reveal, imbalanced datasets are ubiquitous in real-life situations: examples of this can be found in several fields, from biomedical applications to sensor data analysis (e. g.: [1] and [2] show Smart Health applications where both fields meet). To make the situation worse, in a wide range of situations, the minority class is the class of interest, e.g.: in automated diagnosis of an illness, there are usually hundreds of negative cases for each positive case (as most of the population is usually healthy). This makes the task of identifying and solving the class imbalance an important one, that has been given a lot of effort to solve by reasearches in the field.

As we will see in the state of the art, there is no shortage of methods in the literature to deal with class imbalance. But we will also show how the class imbalance problem is more complex than a single parameter balance, having different complications depending on the particular characteristics of the dataset we are evaluating and the learning algorithm we are using. This will make more difficult the selection of a particular technology, and generate a wide variance in the global results of all the algorithms used to tackle it.

Being a very wide field of study, this project will center its scope in the analysis of the impact of this problem in the induction of a particular kind of probabilistic graphical models: the Bayesian Classifiers. We will see how this graphical models offer some advantages over other models, by providing the analyst with a more powerful tool that could not only be used for classification taks, but also for general inference queries, missing data treatment, or distribution sampling. We will also see that the induction of this kind of complete models will come at a cost of losing some of the performance we could have achieved with other simpler but efficient classification-only models (e.g.: SVMs, Decision Trees or Random Forests).

1.2 Aims of the Work

Two are the main objectives of this project: first, to evaluate and validate experimentally to what extent structure and parameter induction for Bayesian Classifiers is affected by the imbalance rates in reallife datasets; and second, to propose improvements on the basic algorithms for Baysian Classifier induction in order to overcome those induction problems. To this extent, common algorithms to cope with the imbalanced classes problems (like preprocessing and ensembles) will be revisited, and some improvements will be proposed to try and improve over the base performance.

1.3 Approach and method followed

The first part of the project consisted in the documentation of the state of the art for both the problem of imbalanced classes in Machine Learning, and the theory underlying Bayesian Classifiers. The former was achieved by checking a wide set of papers on the subject, both general analysis of the problems and papers on specific solutions. A deeper understanding of the problem was achieved by implementing some of the algorithms as part of the state of the art description. Most of the theory about Bayesian Classifiers and other Probabilistic Graphical Models comes from [3] and [4].

The second part consisted in the implementation of a Bayesian Classifier, and the analysis of the impact of the class imbalance in its performance. To this extent, two possibilities where considered: the implementation

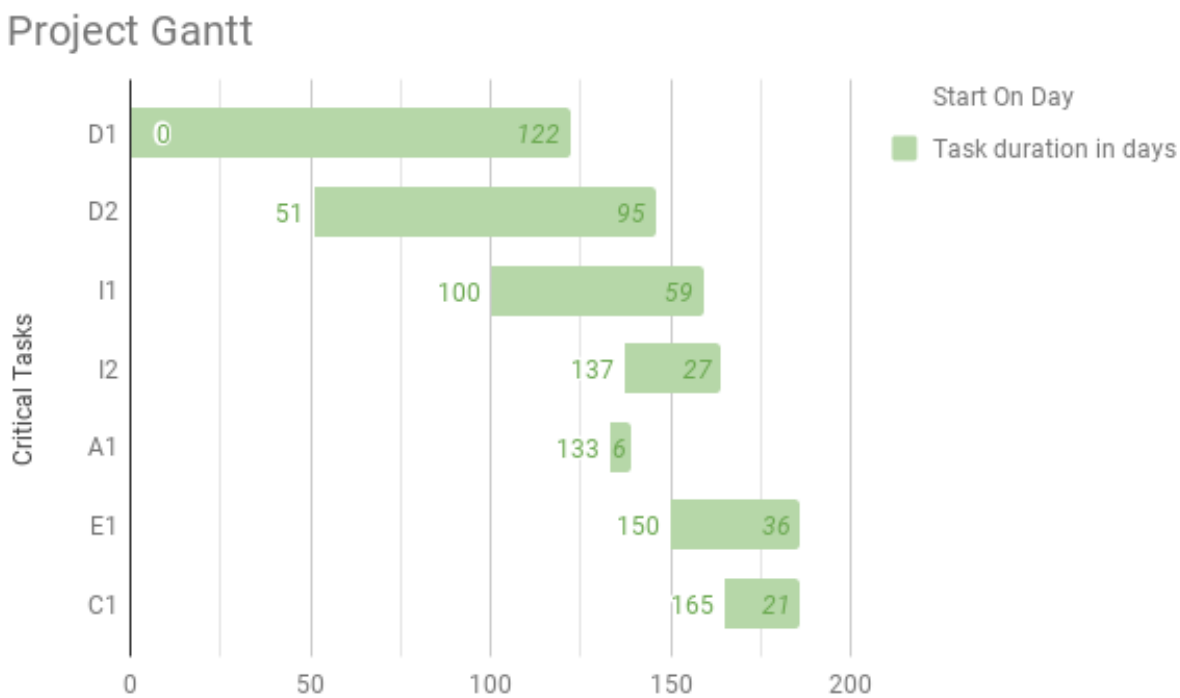

Figure 1: Gantt of the project showing the number of days from start and total length in days of each task

from scratch of a Bayesian Classifier ad-hoc for the project; or the modification of a stablished implementation of BCs to add the instrumentation and options required for the desired experiments. The chosen path was to implement the BC from scratch, since the cost of understanding a reference implementation (usually more complex than the simple classifier required in this case) was judged to be greater than the cost of creating a prototype implementation for the experiments.

1.4 Planning of the Work

The following tasks were required to complete the work:

- D1: Documentation on the problem of class imbalance in Machine Learning. Includes the implementation of some of the algorithms, and experiments with real datasets.
- D2: Documentation on Probabilistic Graphical Models theory.
- I1: Implementation of support libraries (dataset package, näive bayes reference algorithm).
- I2: Implementation of the basic Full Bayesian Classifier.
- A1: Analysis of the impact of the class imbalance problem in BCs (theory).
- E1: Design and execution of experiments.
- C1: Documentation and conclusions.

Figure 1 shows the planification of the project with the total final duration in days.

1.5 Brief summary of products obtained

Two are the main products of the project:

- An analysis and experiment of the impact of the class imbalance problem in the induction of Bayesian Classifiers.
- And the proposal and implementation of new mechanisms to tackle the problem. Those will be added to a Bayesian Classifier that will also be implemented as part of the project.

1.6 Brief description of the others chapters of the memory

The project is structured in the following way: Section 2 shows a brief introduction to the topic of learning in imbalanced datasets; Section 3 shows the most frequent solutions in the literature for the problem of imbalanced datasets; Section 4 gives a brief introduction to Bayesian Classifiers and motivates their selection as the classifiers to study; Section 5 analyzes the expected influence of the class imbalance from a theoretical perspective; Section 6 describes the experiments that were conducted in order to validate some of the assumptions in section 5, and analyzes their results; finally, Section 7 summarizes the conclusions and future work. Extra information about the implementation details, statistical analysis and the datasets used along the document can be found in the apendices.

2 Imbalanced datasets

2.1 Definition of the problem

2.1.1 Definition of Learning

The first step on defining the problem will be to make a brief introduction to the concept of learning, to introduce terms and notation that will be used across the rest of this document (a more formal introduction can be found in [3]).

Informally, the task of learning will consist in generating a model that captures the underlying structure of a problem, given a limited amount of examples; the limited number of examples will make the induced model just an approximation, that we will hope to be accurate enough for our purposes.

Formally, the learning task will consist on:

- given a dataset $D = \{X[1], X[2]..X[M]\}$ consisting of M examples drawn from an underlying, probably unknown distribution, $P^*(X)$,
- where each X consists of a set of N variables, $X_1, X_2, ...X_M$,
- and where each variable has its own domain $X_i \in Val(X_i)$ (that may be discrete, in which case $Val(X_i) = \{r_1, ..., r_k\}$),

inferring a model M that captures the structure of the original distribution, so that:

$$P_D(X) \approx P^*(X)$$

This adopt this definition based on probability theory, as we will see that it generalizes to types of learning where the output may not be a probability.

There are multiple tasks that can be accomplished through learning. This tasks can be divided attending to different criteria. Considering the goal distribution, we can distinguish two kind of tasks:

- **generative learning**: where the objective of the learning is to discover the full structure of the underlying distribution, so the target probability distribution is $P_D(X_1, X_2, ...X_N) \approx P^*(X_1, X_2, ...X_N)$.
- **discriminative learning**: where the objective of learning is to discover the conditional distribution over one of the variables, so the target distribution is $P_D(X_1, X_2, ..X_i|X_{i+1}..X_N) \approx P^*(X_1, X_2, ..X_i|X_{i+1}..X_N)$.

Generative learning models, like Bayesian Classifiers, are more general than discriminative ones, in the sense that, providing a full joint distribution, they also provide a way of getting the conditional one (while the opposite is usually not true).

Learning algorithms can also be divided according to their train objectives in supervised and unsupervised. In the case of **supervised** learning, the dataset $D = \{< X_i, y_i >\}_{i=1..M}$ is a set of example tuples containing a list of values for the predictor variables and a value for the target class (or classes), that will be used to inferr a model of the conditional probability of the target class on the predictions. The problem can then be specified as a search for the model that best captures the true distribution $P^*(Y|X_1, X_2, ...X_M)$.

In **unsupervised learning**, the objective is, once again, to infer a variable (or set of variables) from the given data, but in this case, the dataset contains no information about the target classes. The rest of the paper we will focus mostly on supervised classification problems.

The learning process can be viewed as an optimization process where a target loss function is minimized for the dataset instances. A **loss function** gives an estimate of how well a model fits the data represented by a given dataset D. Commonly used loss functions include the *classification error*:

$$E_r = E_{(x,y)\sim\tilde{P}}[\mathbb{I}\{\overline{\sim}_{\tilde{\mathbb{P}}}(\curvearrowleft) \neq \curvearrowright\}]$$

where $h_{\tilde{P}}(x)$ is our estimation function; i.e.: the probability of selecting the wrong label, given the training data. Another commonly used loss function, that takes into account the uncertainty in the prediction is the *conditional log-likelihood*:

$$L = E_{(x,y)\sim P^*}\left[log\ \tilde{P}(y|x)\right]$$

The selection of the loss function to optimize will influence the behavior of our classifier when dealing with imbalanced datasets, as we will show in subsequent sections.

2.1.2 Model selection

The learning process involves the choice of a representation for the induced model. This choice of representation will determine how rich the hypothesis space will be, and this decision will have an impact in the ability of the model to generalize. As the model gets more complex, and the hypothesis space wider, the capacity for the model to accurately represent the target distribution grows higher. But there is a caveat to this increased complexity. Since the amount of dataset instances is limited, as the hypothesis space grows, the density of examples shrinks, and so the posterior probability of unseen examples tend to zero. These low probabilities and high number of possible models make the task of selecting the most appropriate one (the one that better fits to the target unknown distribution) more difficult. This problem is referred to as the **variance**. A reach space of hypothesis can also lead to a phenomenon called **overfitting**, where the classifier learns to perfectly predict the training set, while losing its ability to generalize to other unseen data.

On the contrary, when the hypothesis space is so small that the target probability distribution can't be adequately fit, it's said that the model has a high **bias**. There is a necessary tradeoff between bias and variance in model designing; this tradeoff will usually be dealt with by introducing special **regularization** terms in the loss function expression.

Since the objective of the learning task is usually to perform classification or inference over new instances, we need a way of estimating its generalization ability. This evaluation involves calculating some kind of metric over a set of instances, but drawing them from the training data will not give a good estimation over unseen instances. In order to get an unbiased estimation of the generaliation ability of a model, a subset of the available dataset instances will be hold out, to create a **test dataset**. The remaining data will be referred to as the **training dataset**. For those cases that imply model selection (whether it is a hyperparameter search or a selection among a set of algorithms), this training dataset can be in turn splitted, to form a third split, called **validation dataset** that will be used to calculate the metrics that will be used in the model selection process.

2.1.3 Data imbalance

The problem of data imbalance is defined as the scenario where, given a dataset $D = \{< X_i, y_i >\}_{i=1..M}$, where the target variable has K different classes, $y_1, ..y_k$ and :

$$\exists i, j \quad s.t. \quad M_D[y_i] << M_D[y_j]$$

where $M_D[y_i]$ is the number of times the value y_i appears in dataset D. Hereafter, for the binary cases, the underrepresented class will be called **minority** or **positive** class, while the overrepresented one will be called **majority** or **negative**. This situation usually leads to an understimation of the probability of y_i, that may be harmful for our objectives in two ways:

- since the available dataset instances are usually very scarce when compared with the complete feature space, it may be the case that the empirical distribution of examples in the dataset may not reflect appropriately the underlying distribution.
- most of the times, accurately predicting the positive class is the target of our modelling, so classification errors will not be as important for the negative class as it would be for the positive (i.e.: underrepresenting the positive class is more harmful than doing the same for the negative one).

As we will see in section about interrelation with other factors, the class imbalance problem is usually mixed with other factors in practice, that may change the extent of its impact on the learning process.

2.2 Metric Selection

The inability of the classical classifiers to cope with imbalance problems is largely due to the loss function they are designed to optimize, as pointed out in [5]. Most classifiers are trained to optimze the 0-1 loss, that can be understood as 1 minus the Accuracy, a global measure that doesn't take into account the per-class classification errors. Both performance metrics have a strong bias towards the majority class, to the point that, for highly imbalance cases, the simplest strategy of always predicting the majority class will offer a really high Accuracy. This is highly problematic in most cases, as it is usually the case that the minority class classification accuracy is more important than the majority one (e.g.: in automated diagnosis systems where patients with an actual disease represent the minority class).

Classification error is usually defined in terms of the *confusion matrix*: a $K \times K$ matrix, C (where K is the number of classes) that indicates, for any element $c_{i,j}$, the number of elements of the class K_i classified as elements of K_j. Two local performance measures are particularly important in this context:

- The *precision* measures, out of the total number of ground truth instances of a particular class, how many of them where correctly classified. It's defined with the following formula for the multiclass case:

$$P_i = \frac{c_{ii}}{\sum_{j=1}^{K} c_{ji}}$$

- The *recall* measures the amount of instances classified as being part of a particular class, from the total number of instances of that class in the ground truth distribution. It's defined with the following formula for the multiclass case:

$$R_i = \frac{c_{ii}}{\sum_{j=1}^{K} c_{ij}}$$

This two local measures serve as building blocks for other global metrics that have different properties when dealing with accuracy. Most of those metrics can be redefined as particular instances of the weighted Hölder means across class recalls. The following list shows the most common ones in the literature:

- The *a-mean* is the arithmetic mean among recalls, i.e.:

$$A = \sum_{i=1}^{K} \frac{1}{K} R_i$$

- The *g-mean* is the geometric mean among recalls, i.e.:

$$A = \sqrt[K]{\prod_{i=1}^{K} R_i}$$

Another common metric in the literature is the F^1 metric, defined as the harmonic mean of precision and recall:

$$F_i^1 = 2\frac{P_i \cdot R_i}{P_i + R_i}$$

This metric, that gives the same weight to both Precision and Recall, is an instance of the more general class of F^β metrics defined as:

$$F_i^\beta = (1+\beta^2)\frac{P_i \cdot R_i}{\beta^2 P_i + R_i}$$

It's important to highlight here that, in most cases, the importance of Precision and Recall for a particular problem may not be equal, as there may be very different misclassification costs for the positive classes (medical diagnosis is a common example). For those cases, the F^β metric offers a tool to select the desired level of significance of both components.

The total error classification Accuracy can also be defined in terms of the elements of the confusion matrix, as the total number of elements correctly classified (the diagonal of the confusion matrix) over the total number of elements:

$$Acc = \frac{\sum_{i=1}^{K} c_{ii}}{\sum_{j,i=1}^{K} c_{ij}}$$

For those cases where the classifier gives as output a continous value, that can be interpreted as the certainty of the classification as the possitive example (in a two-class scenario), another commonly used measure is based on the Receiver Operating Characteristic curve (ROC curve). This is a curve plotting true positive rate against false positive rate, for each value of the threshold (so that a curve that pass through the upper left corner corresponds to a perfect classifier and a curve that goes along the lower-left upper-right corners diagonal behaves as a random classifier). The total Area Under the Curve (AUC) is a metric in the $[0, 1]$ interval that has been proved to be useful in imbalanced scenarios.

It's important to highlight that there is no metric that can be selected as the optimal classification metric for all datasets and classifiers. For instance, in its study of the effect of several hybrid algorithms, [6] shows that metric selection can have an important impact in classifier rankings for the same datasets, making difficult to decide which classifier is optimal for a given problem. To account for that difficulty, the experimental part of this work will show the results for different metrics, trying to capture both their strengths and weaknesses. The set of metrics will usually contain: Accuracy, F1, Precision and Recall. Area Under the Roc Curve (AUC) analysis will also be used in the final algorithm comparisons.

2.3 Interrelation with other factors

The imbalance of datasets alone can't explain the difficulties in finding a good predictive model for the target datasets, and many papers even point out that the fall in performance observed in some imbalanced datasets may not be due to the imbalance itself, but to other factors that are most commonly found when a high degree of imbalance is present. This can be inferred from the fact that datasets with the same level of imbalance may have very different classification complexities [7]. The following subsections describe several factors that influence the difficulties in classification learning with a particular dataset.

2.3.1 Interclass imbalance

Even if the proportion of the two classes is globally balanced, a single class might have most of its observations grouped in a single cluster, while other smaller clusters get infrarrepresented, so becoming harder to classify

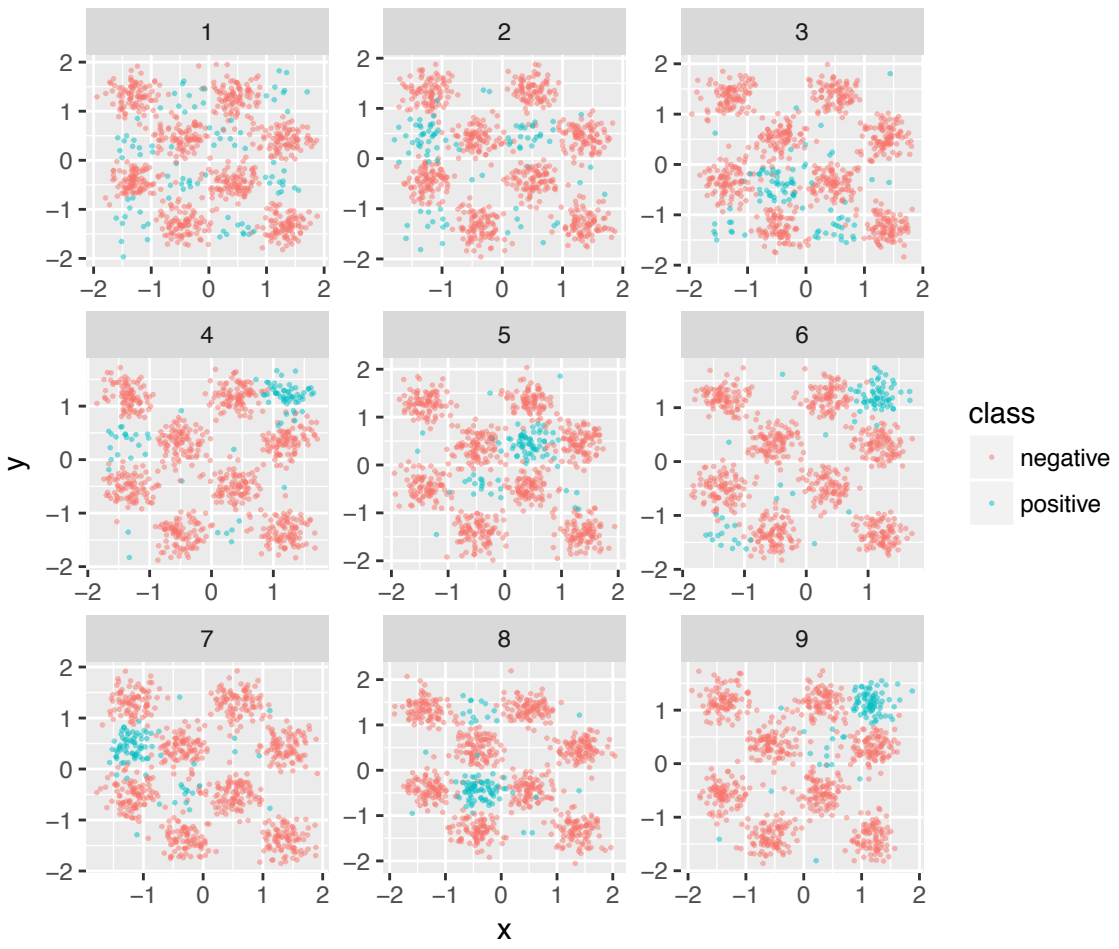

Figure 2: Dataset with growing interclass imbalance

(a problem usually called of the "small (hidden) disjuncts", in the literature). The number and relative sizes of the different class disjunts can be one of the factors of the classification difficulty [8], [9]. Differences in hubness for the classes may also affect the way imbalance affects, even to the point of producing misclassification of the majority class in favor of the minority, when small disjuncts of the majority class appear in a region where there is a bigger density of minority class examples[10], [11].

Figure 2 shows a set of datasets with two classes, each one composed by eight clusters, with an 0.1 imbalance rate, but with a different value of interclass imbalance for the possitive class. Interclass imbalance is marked with a number from 1 to 9. Appendix B explains the formula used to calculate the number of examples in each disjunct based on its disjunct degree.

As we can see, as the interclass imbalance grows, most of the examples concentrate in a small number of disjuncts, to the point where really small disjuncts can be disregarded as noise.

Figure 3 shows the effect of the interclass imbalance level (with all the other parameters fixed) in the Accuracy, F1, Precision and Recall classification metrics for a C5.0 classification tree. As we can see in the graph, the global metrics seem to improve as the global class imbalance grows. This effect comes from the increased density of one of the disjuncts with respect to the rest: as the density on the biggest disjunct grows, so do the classifier performance for that disjunct; and since the positive class examples concentrate in that disjunt as the interclass imbalance grows, the percentage of appropriately classified examples also grows.

Figure 4 shows how the errors committed by the model concentrate in the small disjuncts, making the recall for those disjuncts a lot higher than the one of the biggest possitive class disjunct, that is perfectly captured

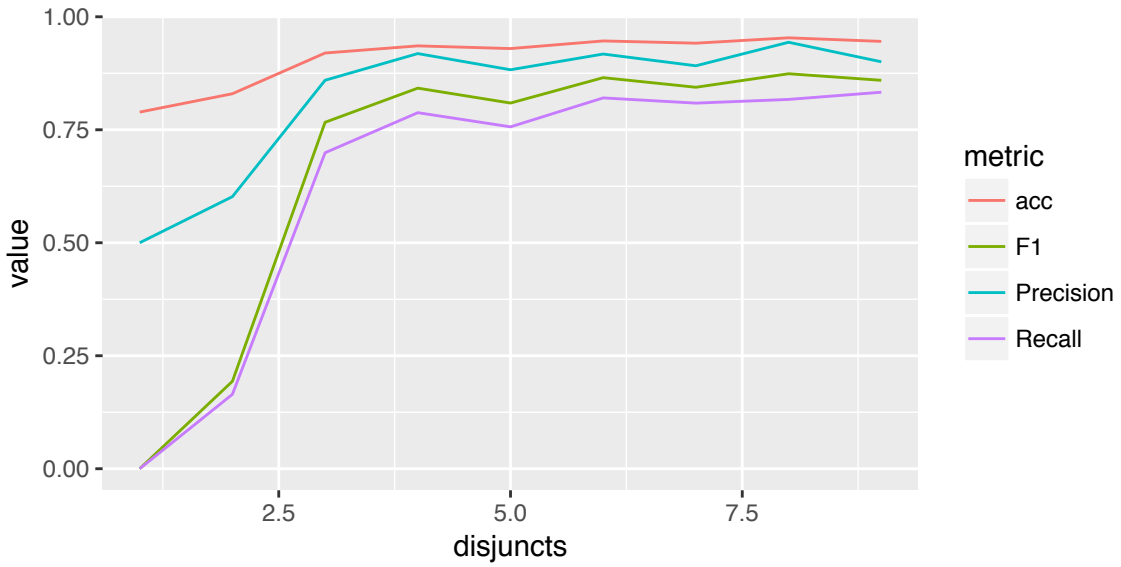

Figure 3: Effect of the imbalance level in the classification for C5.0 classification trees

by the model.

2.3.2 Complexity of the classification borders

With independence of the balance between classes, the complexity of the function that defines the classification border influences the expected learnt accuracy (e.g.: a linear classification border will be very easy to be induced from examples, without any regard to the balance between classes) [12].

Figure 5 shows a group of datasets generated in growing degree of complexity. As it is described in the dataset appendix, complexity will be represented by a different number of clusters for each class.

Figure 6 shows the effect of the growing complexity level (with all the other parameters fixed) in the Accuracy, F1, Precision and Recall metrics for C5.0 classification trees. As we can see, the metrics that depend on the performance of the positive class or both classes, rapidly decay as the complexity of the dataset increases, while those dependant on the global number of errors remain higher (as most of the examples are still correctly classified; the negative ones, mostly).

2.3.3 Size of the training set

Imbalance between classes usually pose a smaller threat on bigger datasets, where even if the Minor class contains relatively less observations, it contains enough in absolute value to clearly define the classification borders [12]. Conversely, when the number of instances in the dataset is very low, the induction algorithm has a lack of information about the boundaries of the problem, so making it more difficult to generalize [11]. This problems may in time be considered as instances of other data intrinsic problems, as the small hidden disjuncts or the presence of noise.

Figure 7 shows a group of datasets generated with exponentially growing size (all other parameters kept equal).

Figure 8 shows the effect of the dataset size with an imbalanced dataset (with all the other parameters fixed) in the Accuracy, F1, Precision and Recall metrics for C5.0 classification trees (size is in log scale). The graphic shows how all metrics usually improve as the size of the dataset grows, achieving high recalls in the

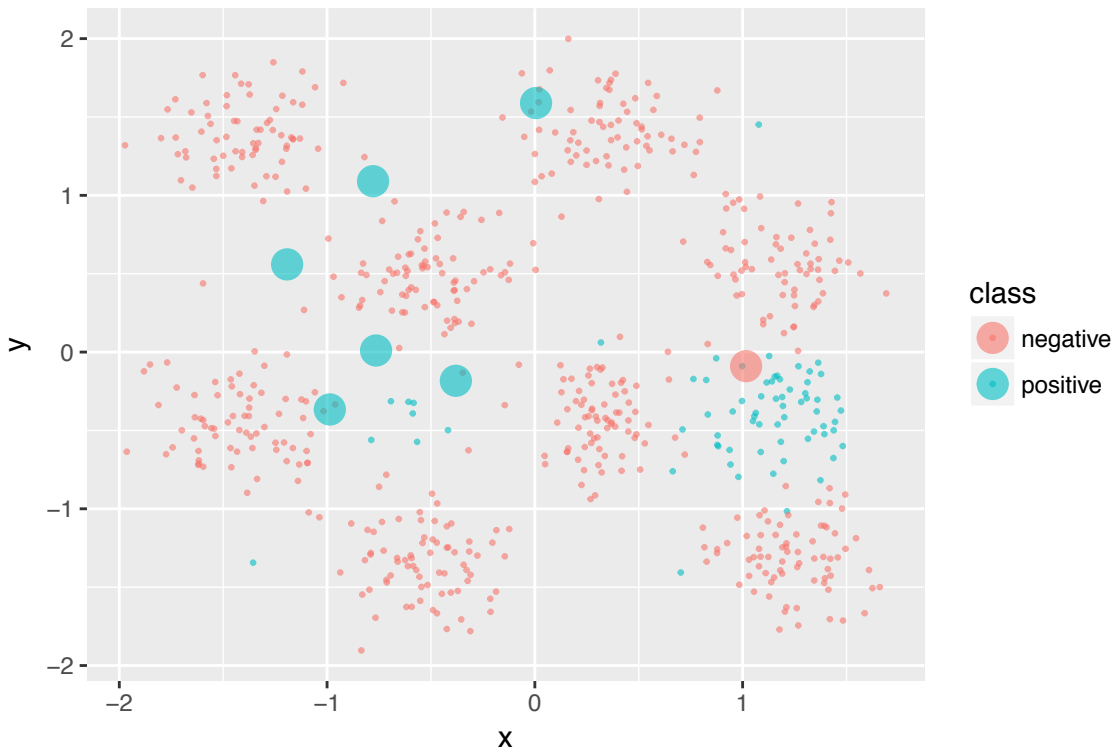

Figure 4: Errors committed in classification with high interclass imbalance

positive class for big sizes, while mantaining the high imbalance rate. These figures show how the imbalance problem is usually confined to the realm of small to medium size datasets.

2.3.4 Level of noise or overlap between classes

Even if the border to be induced has an easy expression, a high level of noise or a clear overlap between classes may make learning the target function a really difficult task. In fact, some authors point out that, for big enough datasets, the effect of imbalance is negligible with respect to the effect of class overlap [13] when considered independently (although the performance loss is higher than the expected combined loss when both are present).

Figure 9 shows a group of datasets generated with exponentially growing level of overlap (all other parameters kept equal). As we can see, as the overlap level increases, the borders between disjuncts become more fuzzy, and it gets progresively more difficult to properly assign a region to each disjunct. This grade of overlap would force a classifier algorithm to return a highly complex model (to account for all the small differences in a very complicated border) or to rely on a statistical approximation (that would give a lower certainty as the level of overlap grows).

Figure 10 shows the effect of the dataset overlap (with all the other parameters fixed) in the Accuracy, F1, Precision and Recall metrics for C5.0 classification trees.

[13] offers one explanation on the concrete mechanism for the classification loss due to the mixture of overlap and imbalance in their study of the complexity of the induced classifier (measured in this case with the number of support vectors in a SVM trained in the dataset) for different datasets. Their results show that increasing levels of overlap, or combined imbalance and overlap, forces the classifier to retain more examples as support vectors, thus increasing the complexity of the classifier. Interestingly enough, this effect is not present in the datasets with high imbalance and low overlap. These results show that an increased complexity may be one

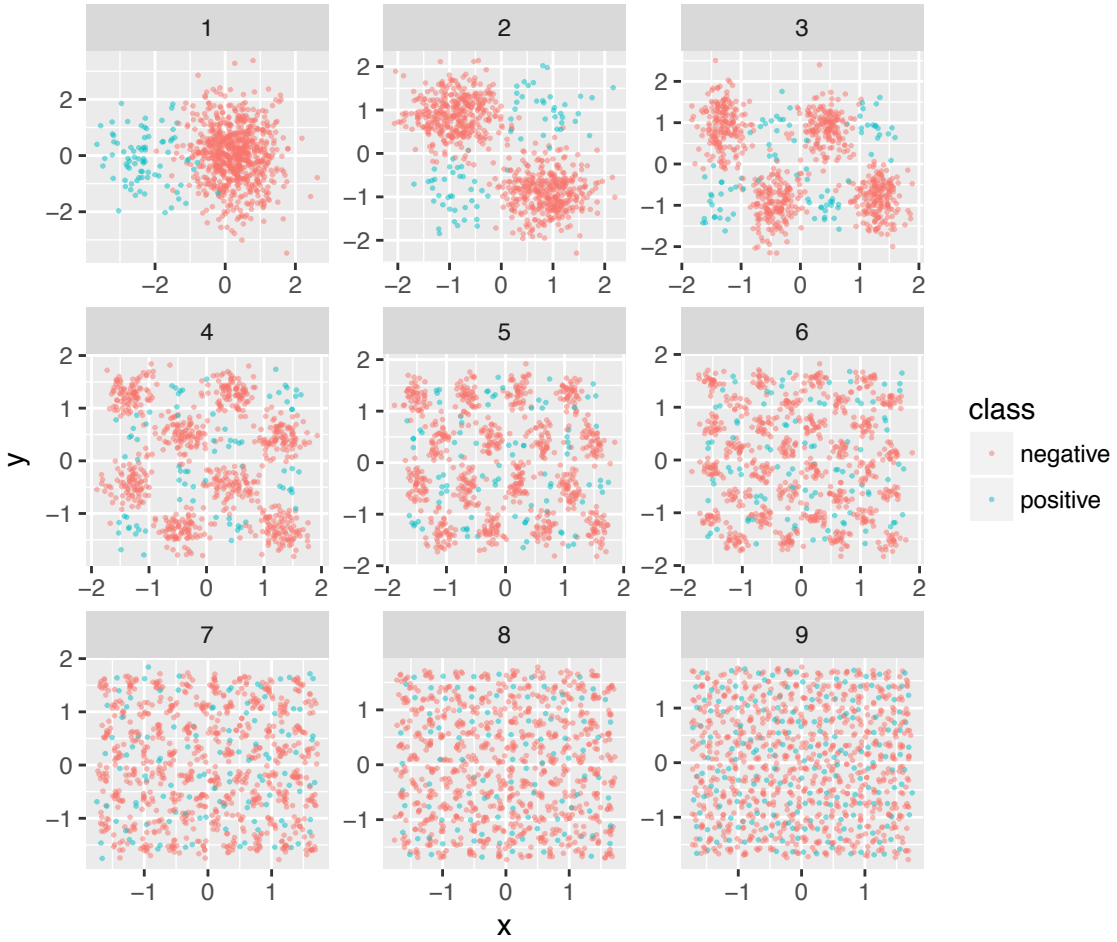

Figure 5: Dataset with growing concept complexity

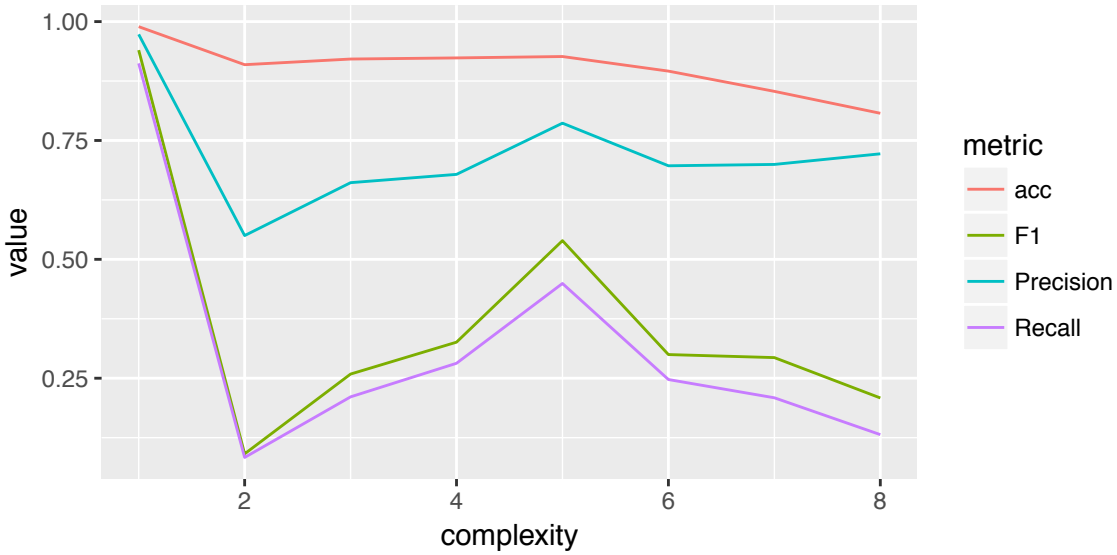

Figure 6: Effect of the complexity level in the classification for C5.0 classification trees

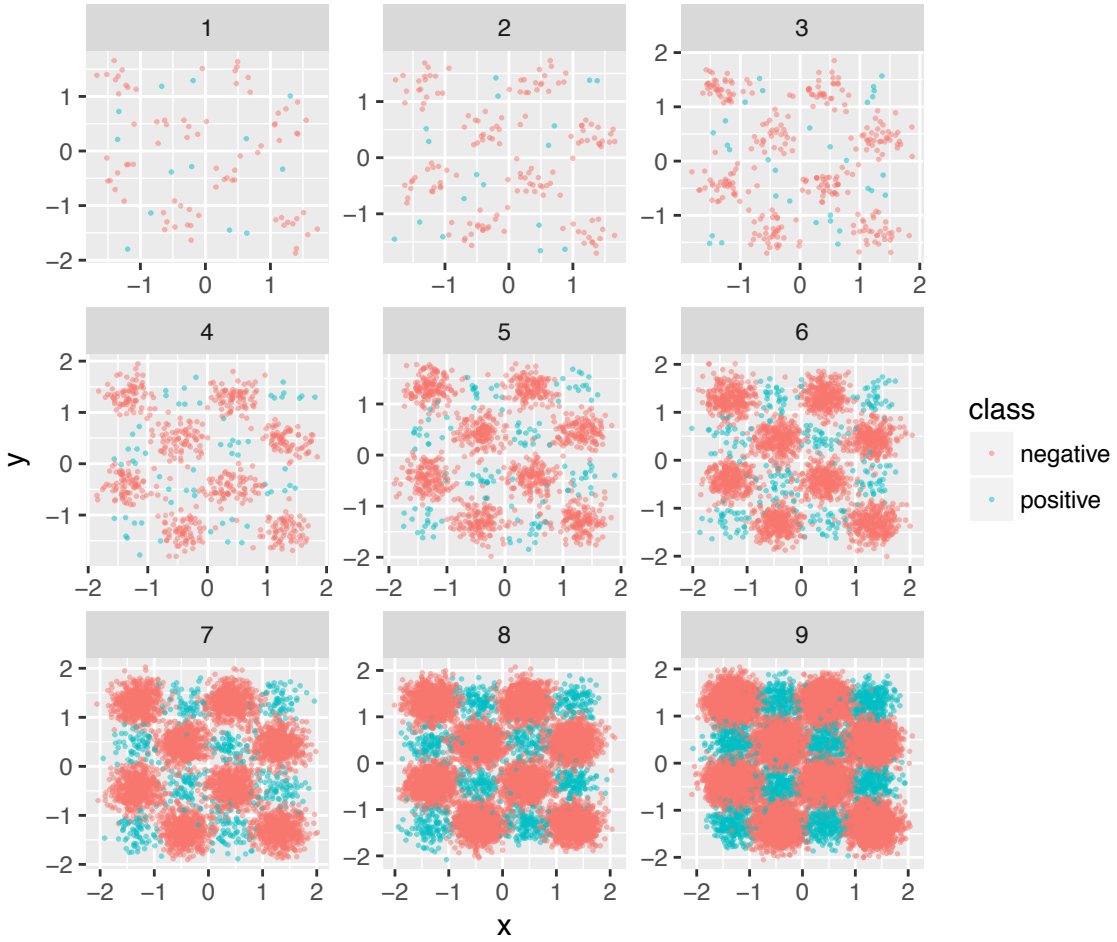

Figure 7: Dataset with growing size

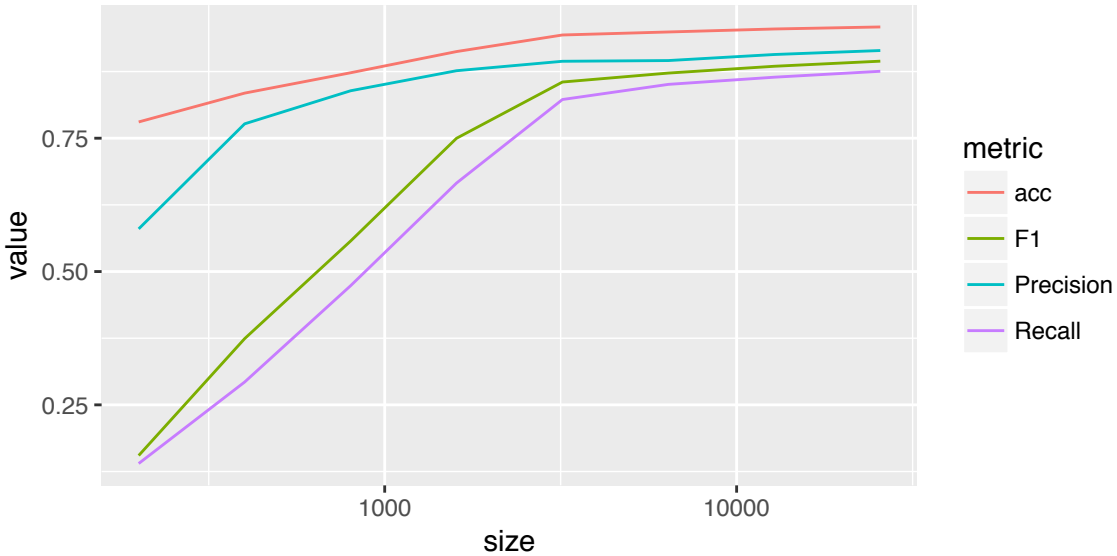

Figure 8: Effect of the dataset size in the classification for C5.0 classification trees

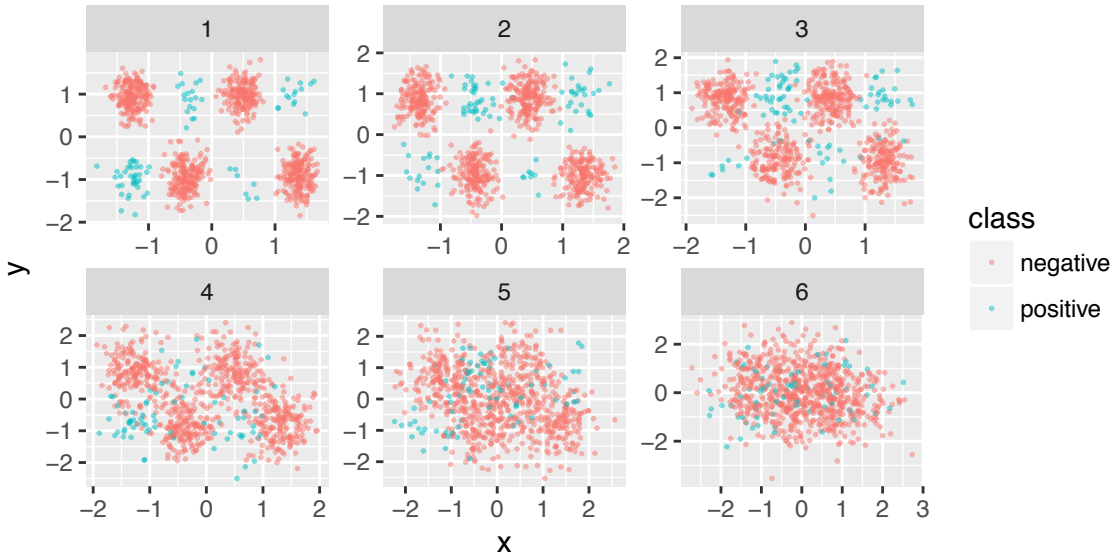

Figure 9: Dataset with growing level of overlap

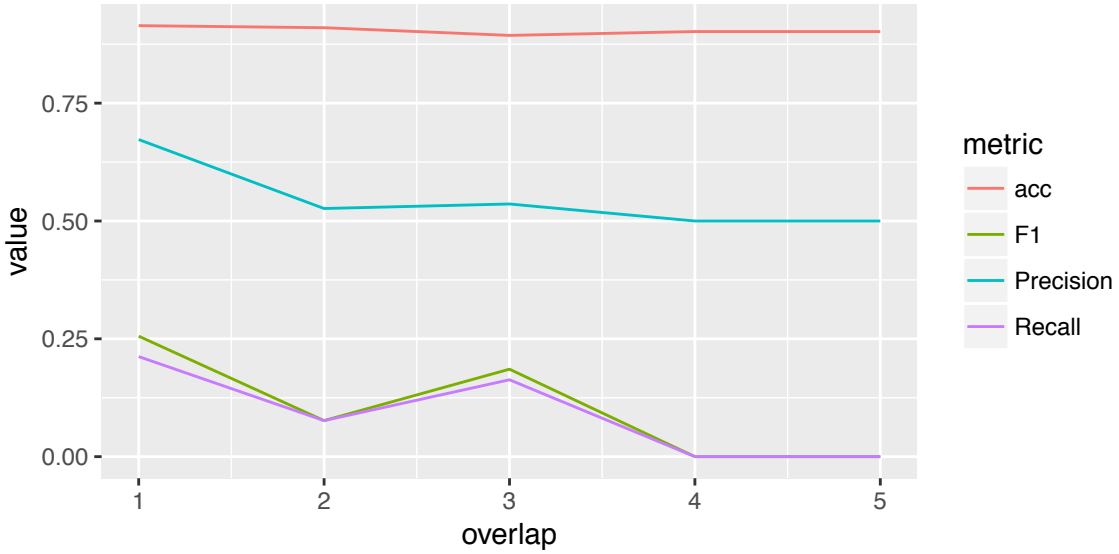

Figure 10: Effect of the dataset size in the classification for C5.0 classification trees

of the factors that reduces the expected average performance in these kinds of datasets: algorithms that fail to generate a complex enough classifier may fail to adequately separate the classes.

3 State of the Art

3.1 Overview

A lot of reasearch effort has been devoted to diminishing the effects of class imbalance in machine learning problems. Most common methods can be grouped in the following categories:

- **Preprocesing methods**, sampling methods being the most popular, that expect to solve the accuracy problem by modifying the datasets, improving the balance between classes (by removing examples of the majority class, or replicating or synthesizing new ones of the minority one) or removing examples that make learning particularly difficult (like borderline examples). Here is important to note that the optimal class distribution for a given problem may not be the balanced classes (unless perfectly balanced classes have proven to be usually a good aproximation of the optimal), but a distribution with some skew, that will depend on the classifier and the dataset characteristics. This fact complicates the task of selecting the appropriate preprocessing method and level.
- Methods based on **new algorithms**, designed specifically with the avoidance of the class imbalance in mind. It's worth mentioning here that different algorithms present different levels of performance in high imbalance scenarios (e.g.: SVMs perform far better than decision trees, that are more commonly found in the imbalance literature).
- Methods based in **Ensembles of algorithms** that combine multiple algorithms or preprocesing techniques for improving the combined accuracy.
- **Weighted learning methods**, that modify the learning algorithm to take into higher consideration examples of the minority class than those in the majority class (as they are usually more important, anyway). Weighting methods will usually be part of ensemble algorithms or will require the modification of existing algorithms to work (and so we will study them as part of those categories).
- In the last years, some **Feature selection methods** have also appeared, specially for high dimensional data, that try to overcome the imbalance problems by selecting a subset of the dataset features over which the classifier behaves better (e.g.: [14]).

The next sections offer an overview of some of the most popular examples provided by the literature for each category, along with a brief description and comparison.

3.2 Preprocessing methods

3.2.1 Evaluation of the preprocessing methods

In order to evaluate the performance of the preprocessing methods in the task of reducing the impact of the imbalance in classification tasks, we will apply them to both real and synthetic datasets first, and then we will apply four different classifiers to the resulting datasets to evaluate the joint performance. We selected four different types of classifiers to better address the different efficiency that particular preprocessing methods could have on different classifier types. All classifier implementations belong to R's Caret package [15], that compiles different popular machine learning libraries. All examples are trained with a 5-fold Cross Validation to estimate the final metaparameters. The selected classifiers are the following:

- A decission tree algorithm, **C5.0** (from the *C5.0* caret package).
- A **Random Forest** algorithm (from the *ranger* caret package).
- A **Bayesian Generalized Linear Model** (from the *bayesglm* caret package).
- A **Suppport Vector Machine** with Radial Kernel (from the *svmRadial* caret package).

3.2.2 Random sampling methods

The most simple type of preprocessing methods are Random Undersampling (RUS) and Random Oversampling (ROS). RUS consists in removing random elements from the majority class in the dataset, until both classes are balanced (or from many classes until a global class balance is achieved in multiclass problems). ROS, on the other hand, consists in replicating random elements of the minority class set, until the total number of elements in both sets are roughly equal.

These simple methods are really easy to implement but offer a poor performance. Each of them has also its particular disadvantages: by replicating existing instances without adding new information, ROS makes the dataset more prone to overfitting; on the other hand, by removing random instances, RUS wastes the information contained in those instances, that may be potentially valuable for classification.

Figures 11 and 12 show a comparison of all the sampling methods with the accuracy and F1 metric obtained in the untreated datasets, for all the classifiers. The vertical blue line shows the maximum value of the metric obtained for each particular dataset for the plain dataset. Red lines show the mean value for each algorithm on the plain dataset (i.e.: without the use of any preprocessing). Transparent bars show the mean value obtained for each metric, dataset and classifier when using the indicated preprocessing method. Finally, solid bars show the improvement over the performance of the plain dataset for each preprocessing method (those that outperform the plain dataset will extend to the right of the 0.0 value, while those that underperform, will do it to the left). The performance of the unprocessed dataset is shown as another bar, with the tag "none".

We can see that there is a general trend for all preprocessing methods to perform slightly worse than the unprocessed datasets for all classifiers and dataset when the considered metric is the Accuracy. This is an expected result, as Accuracy doesn't take into account class imbalance and thus, a classifier could always underrepresent the positive (minority) class to arbitrarily rise the Accuracy until $1 - IR$. While changing to the F1 metric we appreciate that that the behavior changes strongly depending on the considered dataset. We can see that ensembles based in the SMOTE preprocessing algorithm tend to perform better than other algorithms, but this gain in performance also depends on the considered dataset and classifier, as it is overperformed by even the simpler RUS and ROS in certain scenarios.

3.2.3 SMOTE

Another popular preprocessing method, commonly used for method ensembles is SMOTE. SMOTE is a dataset balancing method based on the creation of synthethic examples of the minority class, by making interpolations between neighbour observations. Instead of using the original version (described in [16]), we will use a modified version that creates new examples based solely on those located in the classification borders (as described in [17]).

In order to apply the SMOTE method, we have to find, first of all, the set of all the minority examples that lie in the classification border. To that extent, we calculate the euclidean distance matrix between all pairs of examples, and, if at least half of the k closest neighbours belong to the majority class, we will add them to the borderline set. Figure 13 shows all the training examples, with all the borderline examples highlighted:

Once the borderline set is selected, from each example: a random neighbour is selected from the 5 closest ones. Then, the vector between that example and the original one is computed, and a new example is created just between them. This process can be repeated n times, each time creating new examples between the existing ones. Figure 14 shows the computation of a single iteration of the Borderline-SMOTE algorithm for the previous train set.

The algorithm is usually executed until the desired level of imbalance is achieved. Figure 15 shows the execution of the SMOTE algorithm to achieve a 0.4 level of imbalance. As we can see in the figure, this preprocessing step heavily bias the dataset to provide a stronger representation of the positive class, at the risk of disregarding a high proportion of negative instances as noisy.

According to the classification in Figure 12 we can see that SMOTE has a good overall performance in all classifiers and datasets, being one of the most reliable, as its F1 scores are, in almost all cases, better than

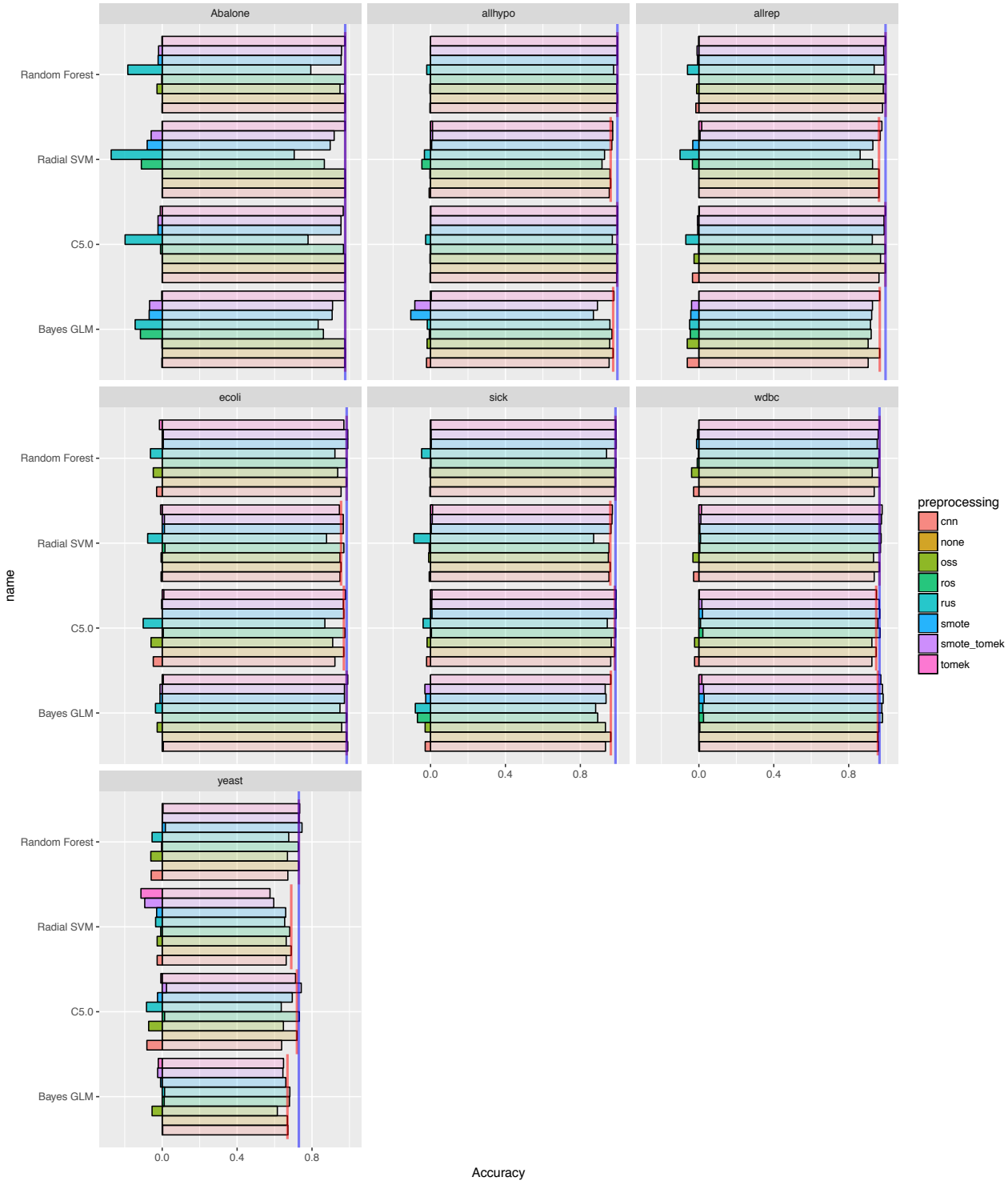

Figure 11: Comparative of random sampling methods using Accuracy metric

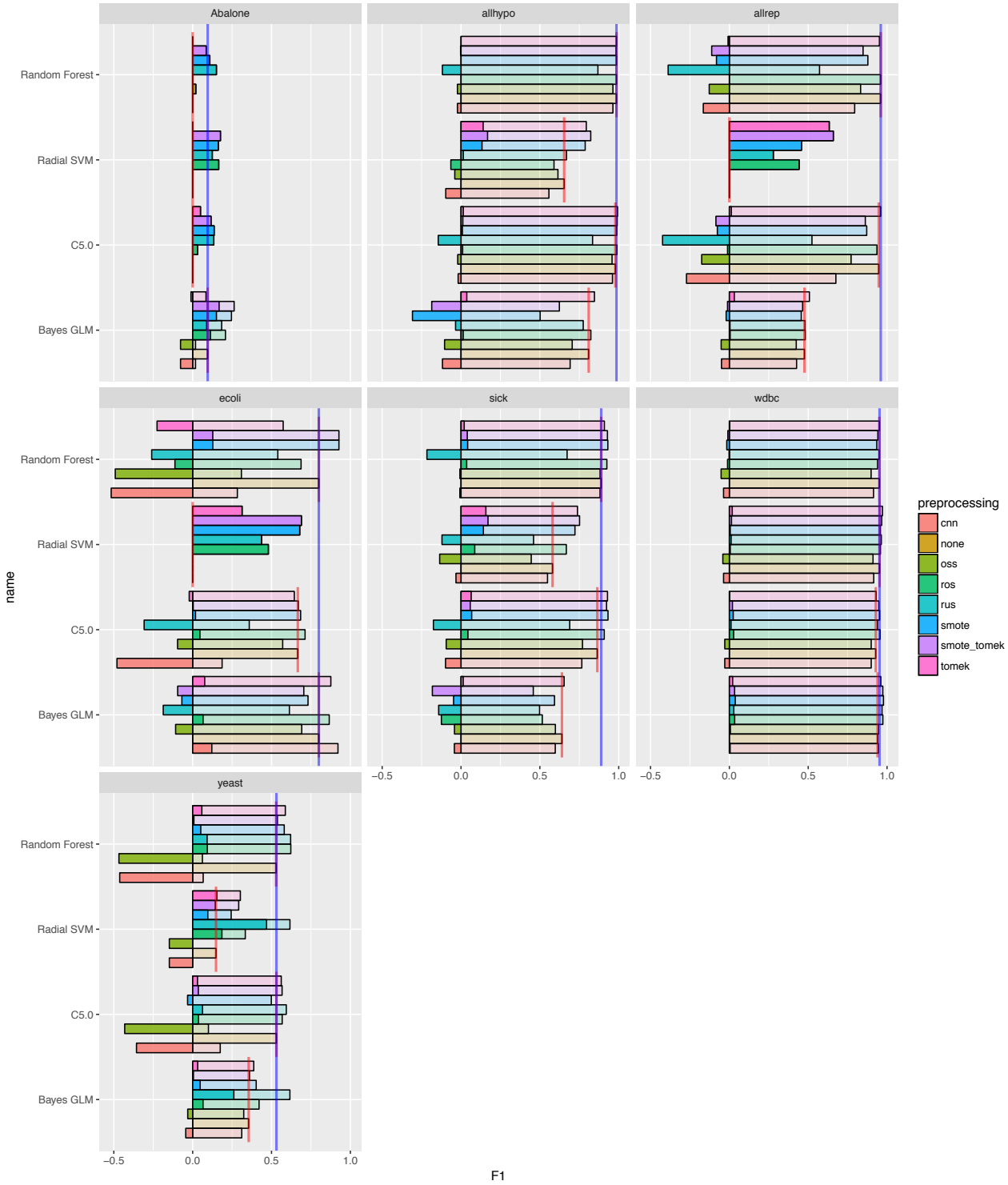

Figure 12: Comparative of random sampling methods using F1 metric

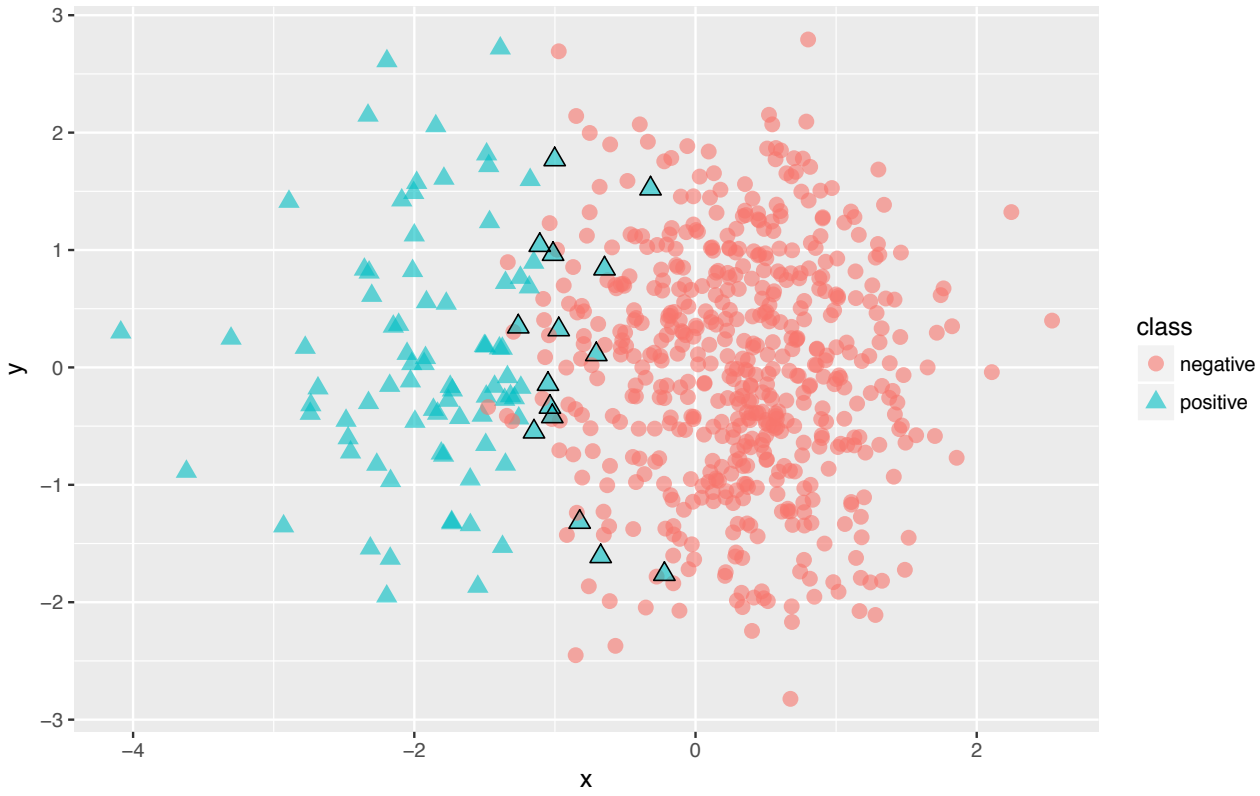

Figure 13: Highlighted borderline instances

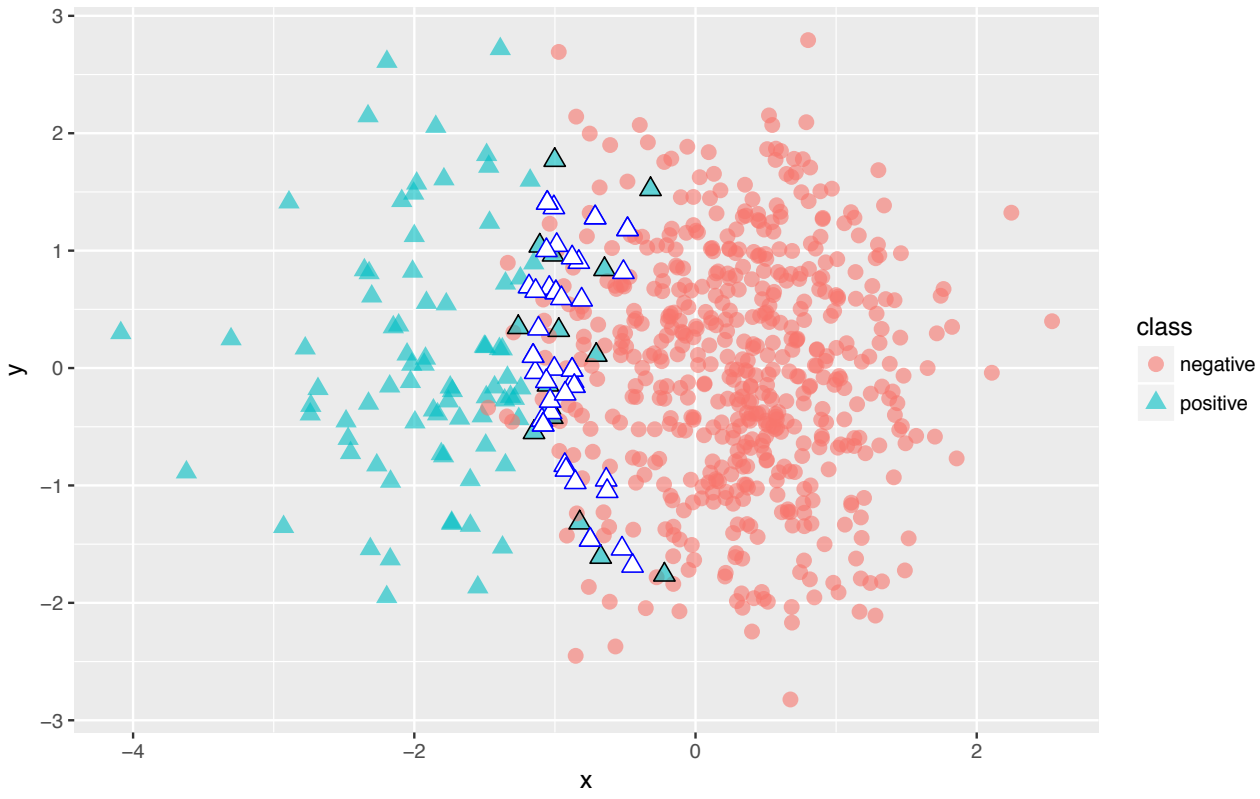

Figure 14: Examples synthesized by the SMOTE Algorithm

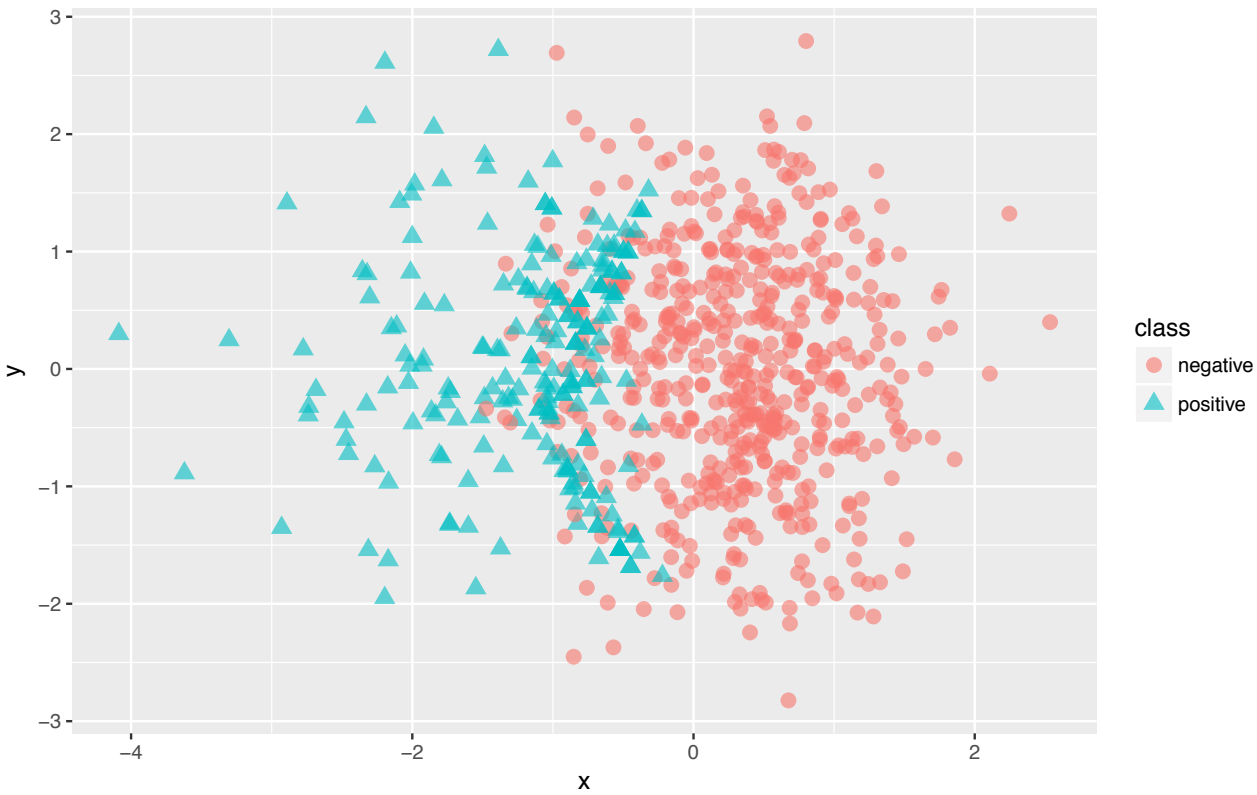

Figure 15: Execution of the SMOTE algorithm for a 0.4 level of imbalance

those of the plain datasets (unlike other methods that exhibit a worse behavior depending on the classifier and dataset).

3.2.4 Tomek Links

Tomek links are pairs of observations each one belonging to a different class such that they are also the nearest neighbours in the complete dataset, i.e.: let $d(\cdot,\cdot)$ be a metric distance, and $P = (x_{min}, x_{maj})$ a pair of examples, one belonging to the minority class (x_{min}) and the other one belonging to the majority class (x_{maj}), if there is no other observation x_k such that $d(x_k, x_{maj}) < d(x_{min}, x_{maj})$ or $d(x_k, x_{min}) < d(x_{min}, x_{maj})$ then P is a Tomek Link. According to this definition, each member of a Tomek link has to be either part of the classification border or classification noise [18].

Figure 16 shows the training examples with the highligted Tomek links, and Figure 17 the resulting dataset once three iterations of the Tomek Links algorithm have been executed.

Directly removing the negative example of each Tomek Link should improve the classification border by two means: removing the noise in the minority class and creating a more defined classification border.

Returning to the classification on Figure 12, we find that, when used in isolation, its effect is usually low compared to that of other more performant techniques (like RUS or SMOTE). It will appear, though, in some of the ensemble preprocessing methods.

3.2.5 Condensed Nearest Neighbours rule

The Condensed Nearest Neighbours algorithm was born as a mechanism to create a minimum set of classification examples for the Nearest Neighbour classification algorithm. It works incrementally, constructing

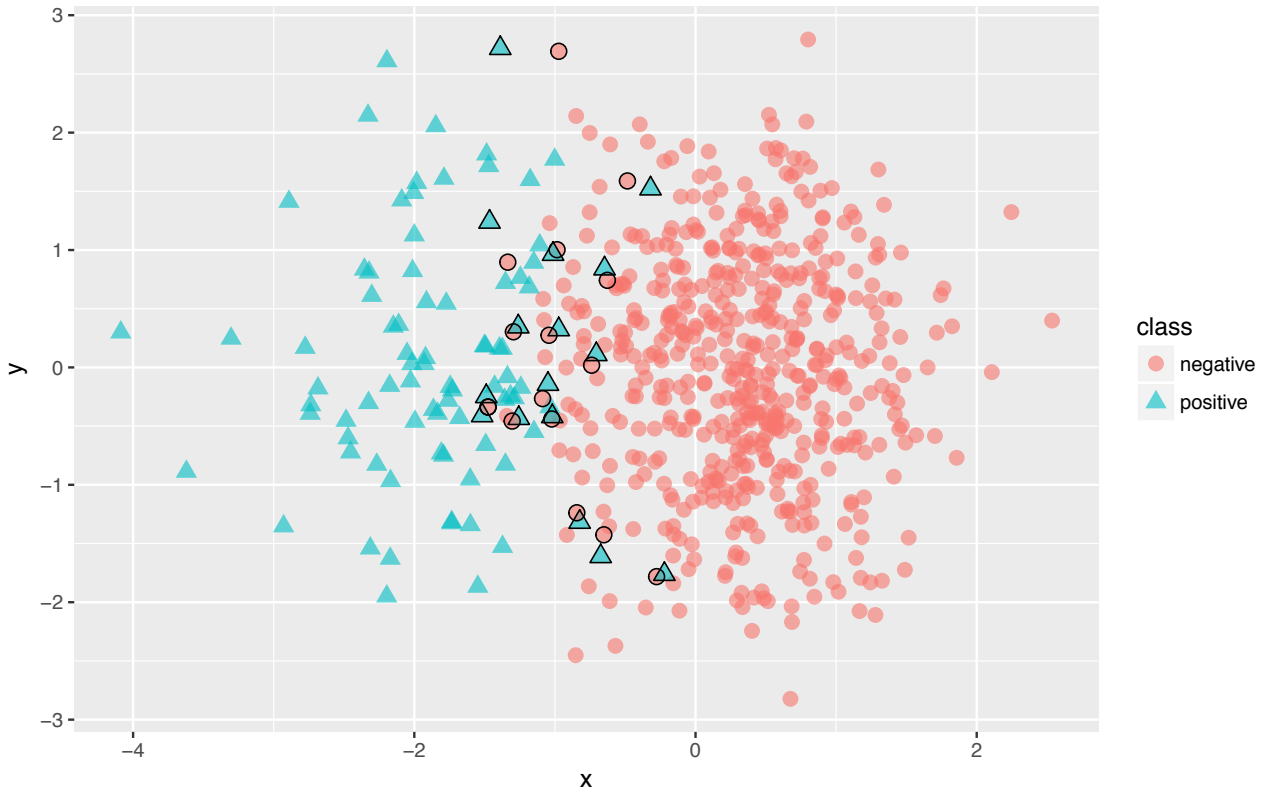

Figure 16: Highlighted tomek links

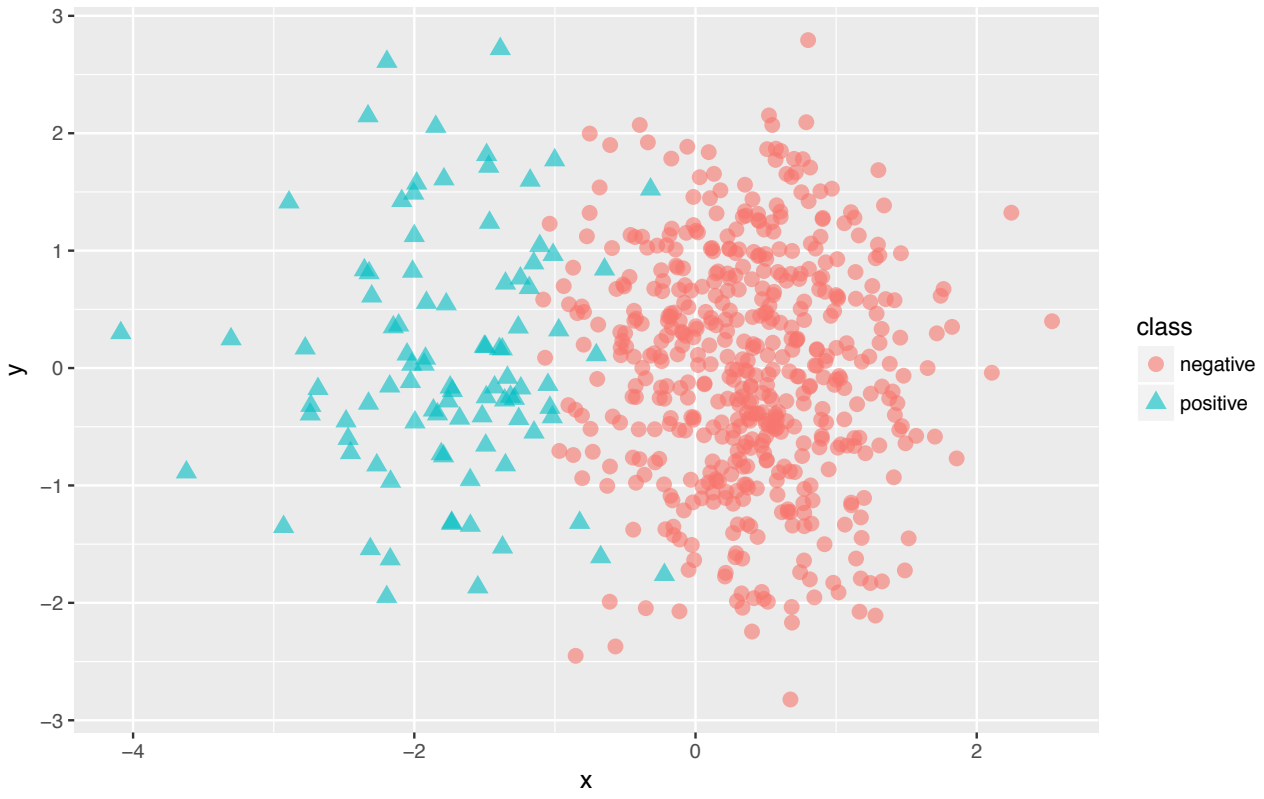

Figure 17: Example dataset with Tomek Links removed

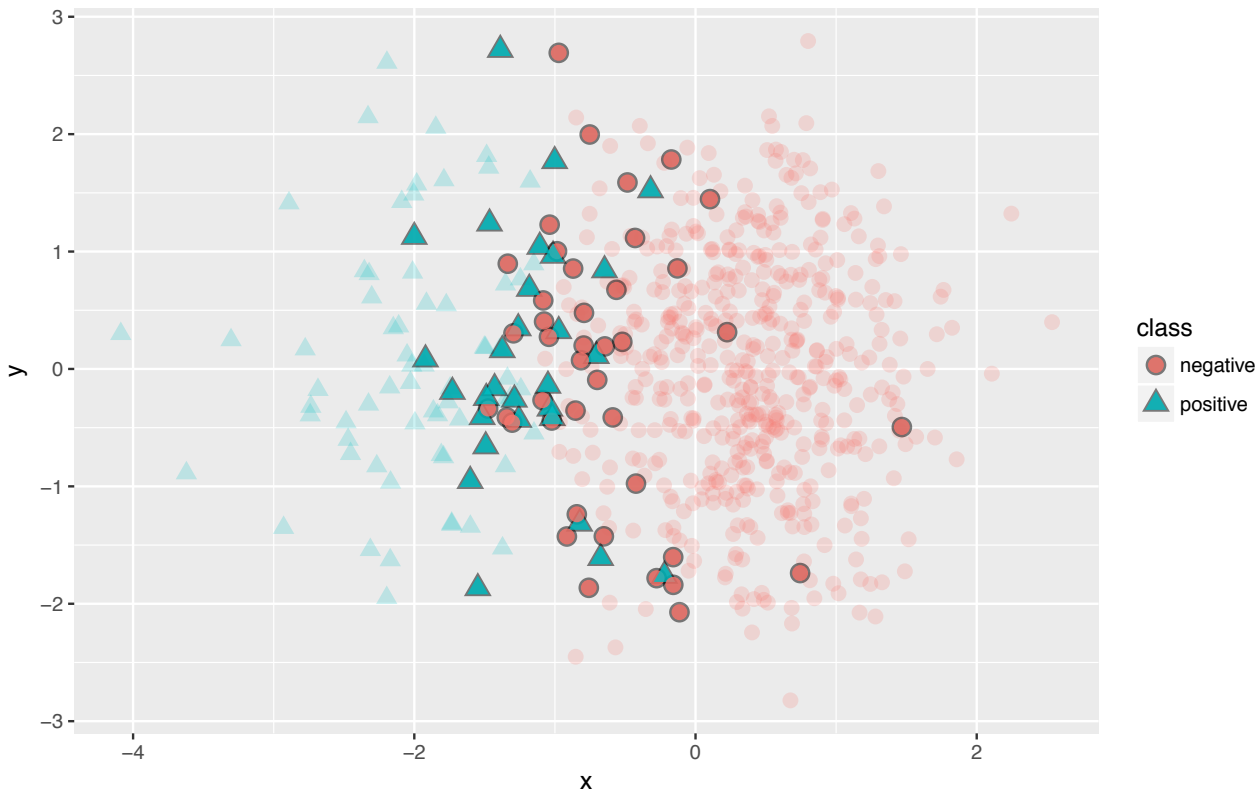

Figure 18: CNN Rule applied to the example dataset

a reference set, *Store* with the following rule: for each example, use the Nearest Neighbour algorithm over the *Store* set to classify the example; if it's correctly classified, the example is put in the discard set, *Grabbag*, otherwise, it is added to the *Store* set. The algorithm continues until all the examples are in one of the two stores. Then, it continues adding examples from the *Grabbag* set, until the later is empty, or a full iteration over the set has been completed without adding any element to the *Store* set. The *Store* set is then the target clean dataset.

Globally, the algorithm will tend to remove redundant examples that are located far from the classification boundaries, thus theorically decreasing the total size of the dataset to be processed, without hampering much the classification accuracy. This reduction has two benefits: being a reduction in redundant examples, it should make the classification boundaries clearer; and the reduction make the training stage easier in computational terms. Figure 18 shows the testing dataset once cleaned with the CNN rule (overimposed over the shaded original dataset).

Figure 12 shows that this method has the worse performance of all the studied methods, usually performing worse than the plain dataset would. It is anyway used in some of the most popular ensemble methods.

3.2.6 Ensemble preprocessing methods

Imbalance literature offers examples of multiple preprocessing proposals that consists in a combination of other baseline preprocessing techniques. The following ones will be used as a representative set:

- **One Side Selection** [19]: this approach consists in a combined use of Tomek Links to remove borderline and noisy examples, followed by the CNN rule to remove the redundant ones. Figure 19 shows an example of application of One Side Selection to the example dataset.
- **Smote + Tomek Links** (taken from the review in [20]): in this case, both classes are removed from

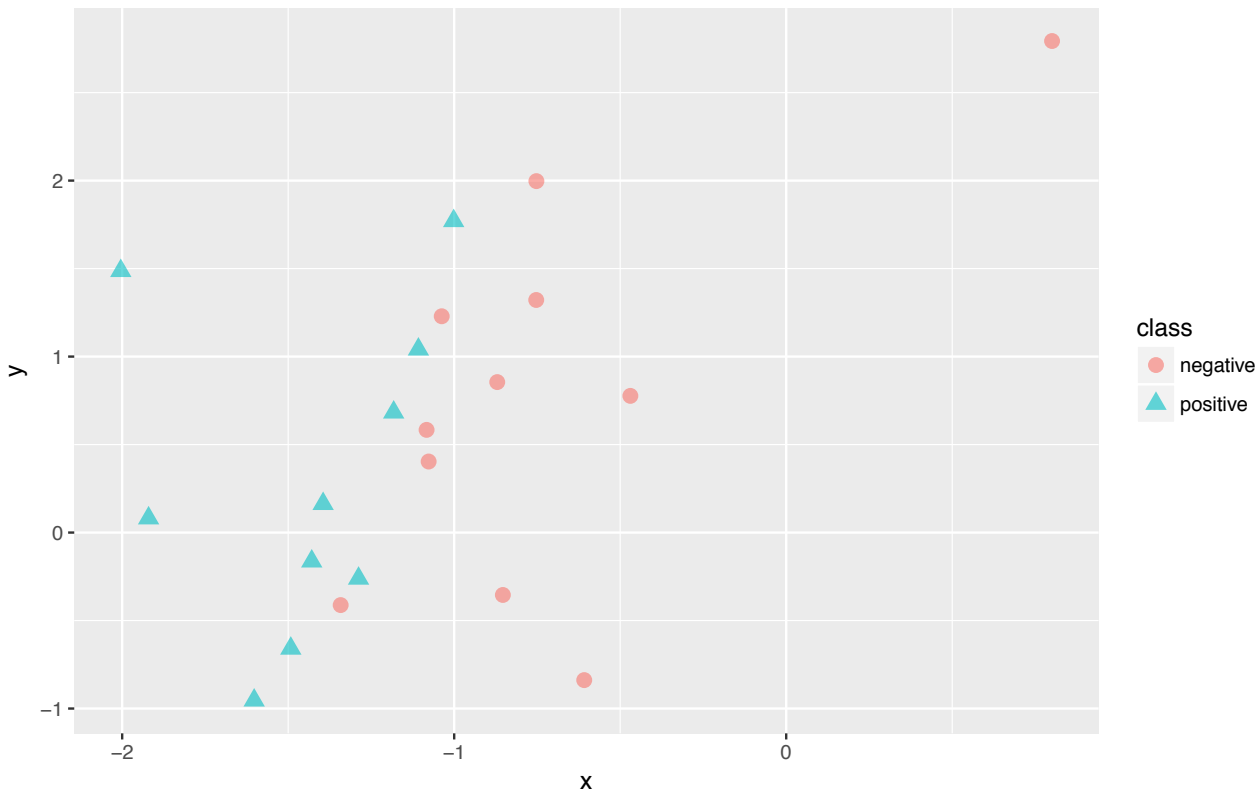

Figure 19: One side selection example

the Tomek Links, in order to clean noisy minority class examples that may invade the majority class while oversampling. Figure 20 shows an example of application of Smote + Tomek to the example dataset.

3.2.7 Current criticism with resampling methods

Despite its widespread use in the imbalanced learning domain, resampling methods have received some criticism in the last years (see [21], for instance). Some of the problems inherent to this methods are:

- Concerning the character of the dataset, resampling don't really add any new information to it, and it can even remove it (as is the case of undersampling methods). Even in the case of the synthetic generation of examples, no new information is generated.
- As they generate no new information, resampling methods have to choose between oversampling (with existing or synthetic data, based in the existing one), that can lead to overfitting; or balancing by removing, that discard potentially useful information.
- There is usually no theoretically sound reason to select a particular class balance level for a given dataset. Equally balanced classes are usually chosen as they tend to offer an acceptable performance; other approaches may search for an appropriate imbalance level in an empirical way, through cross validation, as another metaparameter.
- In any case, the goal of data balancing is to explicitly deviate from the underlying distribution in order to favor learning with current classifier induction algorithms. This objective works against one of the possible objectives of learning, that is the induction of a model of the true underlying distribution from which the dataset has been drawn.

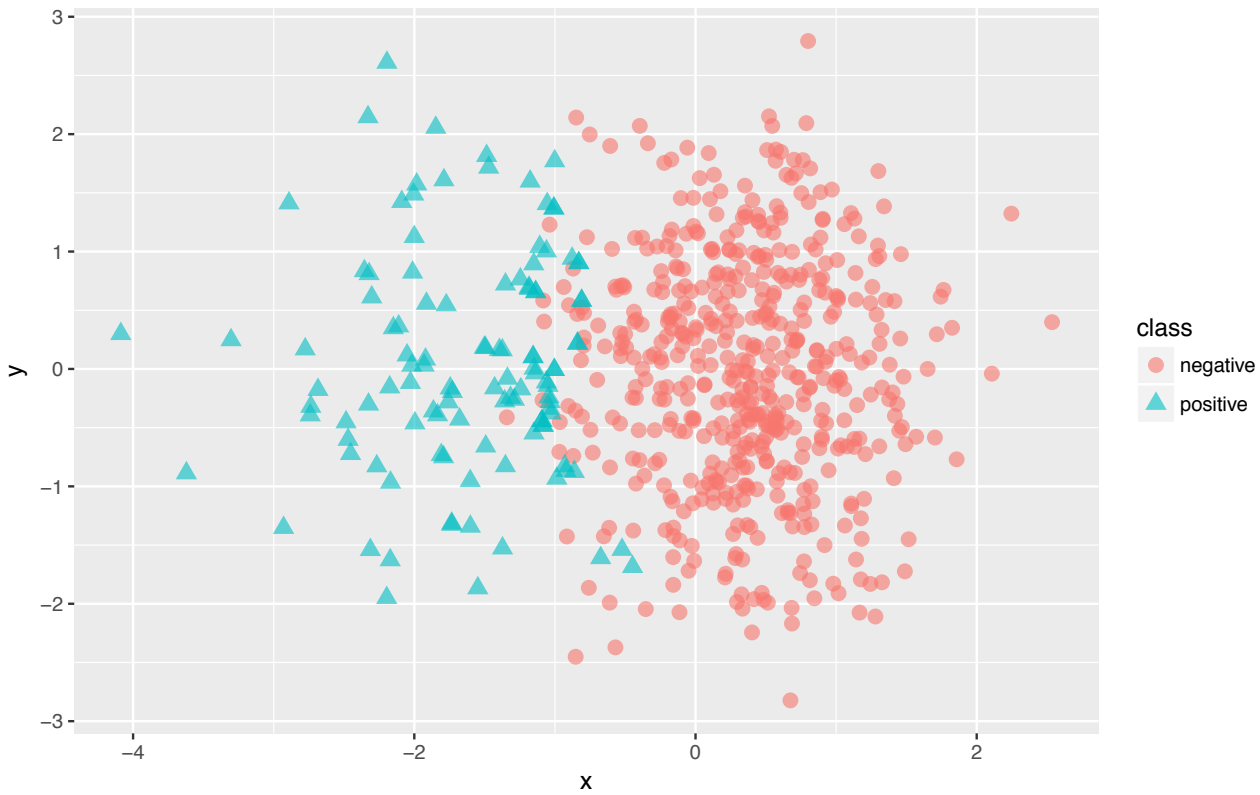

Figure 20: Smote+Tomek example

Critics of the sampling methods argue that those methods are only useful because they help us in using well stablished skew-sensitive algorithms with highly imbalanced data; for those cases, the development of skew-insensitive algorithms would render the sampling methods useless, reducing the complexity of the problem.

3.3 Ensemble methods

In the section about preprocessing methods we've already seen preprocessing ensemble methods, that combined two different preprocessing methods to obtain a better one. In the last years, similar approaches have appeared that combine preprocessing methods with ensemble algorithms. The goal of ensemble algorithms is to offer a way to combine different weak classifiers in order to create a combined algorithm with a performance that is greater than that of any of its components. This improvement will be possible as long as the base algorithms that compose the ensemble have better performance than the simple chance, and are diverse, i.e.: that different base algorithms make different kinds of errors, given the same data. In order for the latter requirement to be accomplished, algorithms are required to be weak, i.e.: algorithms whose performance greatly vary with differences in data (that is, classifiers with high variance). Ensemble algorithms are most commonly used with multiple trained instances of the same base classification weak algorithm.

The two most widely used families of algorithms are *boosting* and *bagging*. In *bagging*, multiple datasets are created from the original one, by sampling with replacement, so each algorithm of the ensemble is simultaneusly trained with slightly different datasets (this difference will allow for the diversity of the base algorithms). Once all algorithms are trained, the class of an example is determined by majority vote among them. In the case of *boosting* the algorithms are trained sequentially, so the results of the first classifiers are used to guide the training of the subsequent ones. This is normally achieved by giving different weights to the misclassified examples, so later classifiers get specialized in difficult examples.

When it comes to design solutions using ensembles to address the class imbalance problem, multiple approaches can be found in the literature. Following the taxonomy in [22], we will divide them in the following classes:

- Boosting-based: approaches that make use of boosting along with some preprocessing mechanisms. Examples of this category are RUSBoost and SMOTEBoost.
- Bagging-based: approaches that combine bagging with preprocessing are usually simpler than boosting-based, as they don't have to modify the original algorithm to apply preprocessing. Some examples are OverBagging, UnderBagging or SmoteBagging.
- Cost-sensitive Boosting: some boosting algorithms can be modified so that the weights they use to train later classifiers are modified based on the class imbalance. Most of the classifiers in this category are modifications of the classical Adaboost algorithm. Some examples are RareCost or AdaC2 ([23]).
- Hybrid ensembles: that is, those that combine boosting with bagging. E.g.: EasyEnsemble and BalanceCascade [24]

In the following sections we will present one example of each of this categories. We will select the examples among the most performant ones for its category, as exposed in [22].

3.3.1 RUSBoost [6]

As it is common in hybrid boosting ensembles, RUSBoost appears as a modificatión of the original AdaBoost algorithm ([25]). This metalearning algorithm by itself already improves the behavior of any weak learning with respect to the imbalance problem, as minority examples tend to be missclassified by earlier classifiers in the ensemble, so their weight increases for posterior iterations. The modifications usually consist on applying a preprocessing step before each of the ensemble's classifiers is trained (thus implicitly changing also the weighting), that makes the transformed dataset more balanced. In the case of RUSBoost, this preprocessing step consist on undersampling the majority class until the desired imbalance rate is achieved (this imbalance rate may not be 0.5 for all cases, as low imbalances have shown to provide better classification accuracies in some cases).

Even though there are more powerful preprocessing steps than RUS, that provide better results when applied in isolation (e.g.: SMOTE), RUS have shown a very good performance compared with boosting classifiers using other preprocessing methods (as boosting algorithms may help overcoming some of RUS's issues, like the loss of information). This, coupled with the fact of its implementation simplicity and the improvement in training speed, due to the reduction of the training set size, make the RUSBoot algorithm an appealing choice to address de class imbalance problem.

Throughout this document, we will use our own implementation of the RUSBoost algorithm, parametrized for the desired level of imbalance (as usually the exact balancing won't lead to the best results).

3.3.2 AdaC2 [23]

In [23], Sun et al propose three new flavours of the Adaboost algorithms, that include a cost item inside the weight update formula. The new weight update formula can be expressed as:

$$D^{t+1}(i) = \frac{C_i^2 D^t(i) * e^{-\alpha_t C_i^1 h_t(x_i) y_i}}{Z_t}$$

where C_i^1 and C_i^2 are the cost functions (depending only on the considered instance of the training set). The three proposed algorithms differ in which of these new cost functions apply: C1 applies C_i^1 alone; C2, applies C_i^2 alone; and C3 applies both. In the original paper, the proposed costs where a set of two constants, C_P and C_N assigned respectively to the postive and negative cases (and selected via grid search). Among the three variants, AdaC2 was the one that provided a greater performance in the original experiments.

In this document we will use our own implementation of AdaC2, implemented in R as a new Caret method, to allow for its use along the other tested algorithms. Considering the big number of datasets we are going to test, we decided to stablish the costs directly based on a function of the imbalance rate. The selected function will be:

$$f(i_r) = \alpha_1 - tan(\beta_1 * i_r - \alpha_2)/\beta_2 - \beta_3 * i_r$$

where the $alpha_i$ and β_i are functions selected to keep the range and domain of the function near the $[0, 1]$ interval. The shape of the function was selected so that:

- if both classes are almost balanced, the ratio between the costs will be approximately one.
- for low imbalances, the weights of the minority class will increase linearly with the imbalance.
- for high imbalances, the weights of the minority will increase faster than linearly with the rate of imbalance.

3.3.3 SMOTEBagging [26]

Instead of building the set of models sequentially, bagging ensembles work by generating all the models in parallel, with different sets of data. In the classical versions of the bagging algorithm, the training set for each model is retrieved by sampling with replacement from the full training data for the whole ensemble. Once all the classifiers have been trained over their respective datasets, predictions are obtained by majority vote over all the predictions. This way of composing classifiers based on improving the ensemble diversity by using sampling methods leads naturally to its use with pure resampling strategies for imbalance correction, like Oversamppling or SMOTE. In [26], the authors propose three of this extensions: Underbagging, Overbagging and SMOTEBagging (using the corresponding resampling strategies). Their proposed extensions were originally intended for multiclass classification problems, but, for simplicity, our implementation will consider just the binary case.

3.3.4 EasyEnsemble and BalanceCascade [24]

EasyEnsemble combines boosting and bagging in a kind of ensemble of ensembles. The idea behind EasyEnsemble is, given an imbalanced dataset, with $|N| >> |P|$, to use bootstrappting to create T subsets $N_i \quad i \in T$ (sampling N with replacement), so that $|N_i| = |P|$. Then, for each N_i, a classifier is trained with the combination of N_i and P using boosting (Adaboost in the original implementation). The results of each of the assemblers are then combined using another boosting classifier.

In the same paper, Liu et al propose a variation, BalanceCascade, that introduces three differences with the previous algorithm:

- Instead of creating T sets from the same original negative instance set N, for each generated classifier, the successfully classified examples are removed from the original set, so the subsets are drawn from a shrinking set of examples.
- The threshold of the Adaboot algorithm is changed in each iteration, based on the new imbalance between the positive example's set and the updated negative set.
- A class is predicted positive if and only if all the trained subclassifiers classify it as positive.

3.4 Methods based on algorithm modifications

In this section we will describe some of the methods that have been developed in order to deal with imbalanced datasets by directly modifiying the learning algorithm, in a way that will make it less skew-sensitive.

3.4.1 Cost sensitive methods

When we presented the class imbalance problem in the introduction we remarked that most of the times, the minority classes were also the ones with the highiest misclassification costs. Cost sensitive methods aim to address this problem directly, by changing the aim of the classification algorithms from the minimization of the number of misclassification to a minimization of the misclassification cost. This strategy has been already reviewed in previous sections regarding the boosting algorithms: they modify the weights of the individual instances when calculating the distribution for the train sets of the following classifier, rising those of the missclasified instances, against the correctly classified ones. The difference with the cost sensitive methods relays on the assumption of equal priors of the boosting algorithm, that the cost sensitive methods will break.

In [27], the authors describe a method to modify decision tree induction algorithms in order to introduce misclassification costs in the decision rule. The basic idea is to change the way in which the probability of a class given the node, $p(j|t)$ is calculated. In traditional trees, this quantity is expressed as:

$$p(j|t) = \frac{N_j(t)}{\sum_i N_i(t)}$$

where $N_j(t)$ is the number of instances of class j in node t of the tree. The proposed modification expresses the weighted conditional probability as:

$$p_W(j|t) = \frac{w(j)N_j(t)}{\sum_i w(i)N_i(t)}$$

where $w(i)$ are the weights of the instances, calculated based on the misclassification costs for that particular class. This change in the quantities that will be measured using the splitting criteria changes the optimization objective of the algorithm from minimizing the number of misclassifications to minimizing the classification costs, without making any change in the rest of the induction process. It's worth mentioning that this change of optimization criteria will decrease the misclassification cost at the cost of increasing the total number of misclassifications.

Another simple approach for cost-sensitive classification is presented in this same paper: the modification of the class prediction process in order to classify each instance using the minimum expected cost criteria (leaving the induction process as it is).

The aforementioned approach was not designed with class imbalance explicitly in mind, but cost-sensitive algorithms may be used to improve performance in imbalanced datasets, by increasing the misclassification costs of the minority class [28]. The study shows classes with a lower number of examples in the training set, and thus, a smaller prior, also have a lower cost (as error rate minimization implicitly assumes equal error costs); but the situation where the minority class has a higher true misclassification cost is very common. This disagreement between class priors, induced costs and real costs has a huge impact in the learning procedure.

The authors also show that there is a strong relationship between prior distribution of classes, misclassification costs and classification thresholds: each of this three can be modified in order to obtain new classifiers, searching for the optimal, and the ROC curves generated for each of the resulting datasets will be quite similar.

Proposed in [29], Metacost is based on the idea of relabeling the training examples before training, to reflect a better estimation of the class the example should be assigned, has it been drawn from a distribution that take the misclassification costs into account. It proceeds by creating a set of resamples from the original datasets (in the spirit of bagging ensembles), each of which is used to train a model. But instead of using the set of models to generate final predictions, the ensemble's votes are used to relabel each of the examples (taking care of ignoring, for each example x_i, the votes of the models that were trained with a set that included that example). Once all the set has been relabelled, the same classification learning algorithm is used in the relabeled dataset, to generate the final model (that now will be implicitly cost-sensitive).

Even if its focus is on multiclass cost-sensitive classification, it also addresses the imbalance problem, by generating misclassification costs, $C(i,j)$, based on a prior class cost $C(i,i)$ and the ratio of the prior probabilities of the two classes, $P(i)/P(j)$.

In their work in [30], a research is conducted on the relation of costs and imbalance from the opposite point of view. The authors show that, given a classifier that output class probabilities, $P(j|x)$ and assign classes based on a threshold p_0, it can be made cost-sensitive by resampling the negative classes by a factor of:

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

where p_0 is the classifier threshold and p^* is a threshold calculated based on the misclassification costs. The way in wich the examples are resampled should not impact the overall result (whether it is a type o oversampling or undersampling, or even a formula to assign weights to the examples). In this same work, the authors provide an analysis on how to treat the case were the prior probabilites of the minority class are different in the training and test sets (while the assumption in machine learning literature is usually the opposite). This analysis is used to justify the use of the resampling for the introduction of costs, by showing that class probabilities can be easilyt adjusted when the prior in the training is different to the prior during testing.

3.4.2 Skew insensitive algorithms

In their work in [31], the authors evaluate the influence of some variations of decision tree algorithms in the inference of classifiers over complex or skewed datasets. In particular, they show how the use of prunning leads to oversimplified decision boundaries. This simplification may help the algorithm to generalize over unknown data, but it may also impact the performance of the algorithm for imbalanced datasets. The authors propose a variant of the common C4.5 decision tree, but without prunning and with a Laplace smooth on the leafs, that is shown to be more skew insensitive than the original version.

On a similar line, [32] examines the split criteria of the most popular decision trees (Information Gain, χ^2, Gini Coefficient...), proposing a metric to define the bias of the different metrics, opening the work for the study of the influence of the impact of the selection of split criteria in the learning process for imbalanced datasets. The proposed metric, based on isometrics over the space of examples covered by selection rules was improved by [33] in order to help in the study of performance metrics, by making use of plots of the 3D ROC Space. The idea behind the 3D ROC Space is to plot in the same graph the relative values in the confusion matrix against the class skew (or the imbalance rate). Horizontal slices along the skew axis of this plane yield plots similar to the ones used in [32], that help comparing metrics for different skew values. Furthermore, by introducing explicitly the class imbalance term in the metric formulas, skew-dependence is made explicit, a feature that is particularly interesting for the analysys of problems with a high class imbalance.

Plotting metrics in the 3D-ROC Space requires a transformation in the metric formula that make it dependent on the selected axis. The authors select the following axis for their plots:

- Skew ratio, **c**, defined as:

$$c = \frac{|Negatives|}{|Positives|}$$

- True Positive Rate, **tpr**, defined as:

$$tpr = \frac{TP}{|Positives|}$$

- False Positive Rate, **fpr**, defined as:

$$fpr = \frac{FP}{|Negatives|}$$

For a given dataset with a known number of examples, this three values are enough to calculate the values of any performance metric (as the missing confusion matrix values can be calculated from the set of constant values and axis values).

As an example, we will show the transformation of the accuracy formula. Accuracy is usually defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Considering the following definitions:

$$TP = tpr \cdot |Positives|$$

$$TN = |Negatives| - FP = (1 - fpr) \cdot |Negatives|$$

$$|Negatives| = c \cdot |Positives|$$

We have the following:

$$Accuracy = \frac{tpr \cdot |Positives| + (1 - fpr) \cdot |Negatives|}{|Positives| + |Negatives|} = \frac{tpr \cdot |Positives| + (1 - fpr) \cdot c \cdot |Positives|}{|Positives| + c \cdot |Positives|}$$

$$Accuracy == \frac{tpr + (1 - fpr) \cdot c}{1 + c}$$

As we can see, this formulation depends explicitly from the class skew, as expected for the Accuracy case. The following formulas shows the new formulation for other common metrics:

$$Precision = \frac{tpr}{tpr + c \cdot fpr}$$

$$F1 = \frac{2 \cdot tpr}{tpr + c \cdot fpr + 1}$$

$$WR_{Acc} = \frac{4c}{1 + c^2} \cdot (tpr - fpr)$$

Figure 21 shows the ROC Space plot with the isometric lines for three different metrics. The results show how the three considered metrics are higly skew-dependent, as can be inferred by the presence of c in their corresponding formulas.

The same analysis can be applied to the tree splitting criteria. Those criteria usually focus on the comparison of the impurity in the parent node against the impurity in the children. [33] offers an impurity formulation that lets express this criteria using the same variables used in the analysis of the performance metrics:

$$m = Imp(pos, neg) - (tp + fp)Imp(\frac{tp}{tp + fp}, \frac{fp}{tp + fp}) - (fn + tn)Imp(\frac{fn}{fn + tn}, \frac{tn}{fn + tn})$$

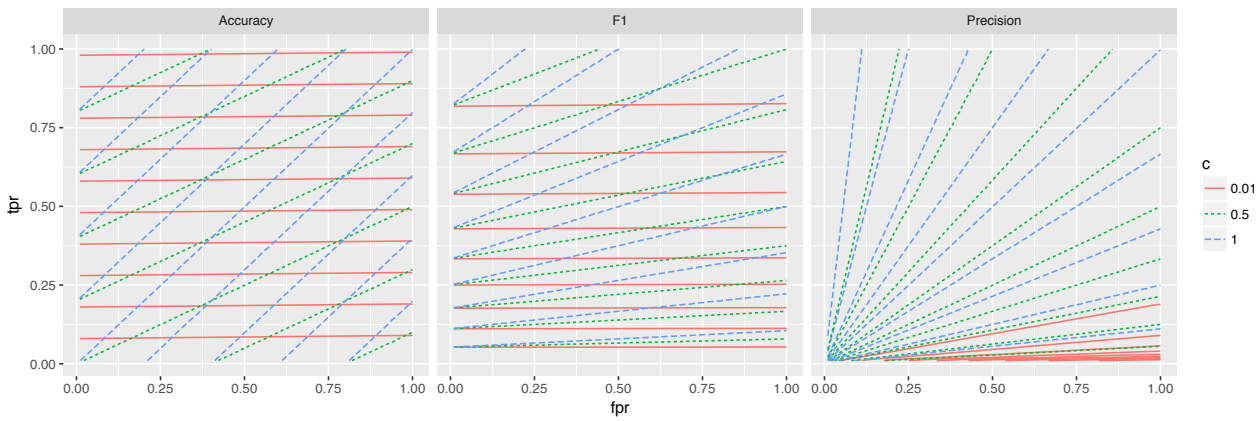

Figure 21: ROC Space for different metrics. Different colors indicate different levels of imbalance.

This equation can be transformed to our two axis, making use of the previous remarks, and the following equalities:

$$TP + FP = |Positives| \cdot tpr + |Negatives| \cdot fpr = |Positives| \, (tpr + c \cdot fpr)$$

$$TN + FN = (1 - fpr) \, |Negatives| + (1 - tpr) \, |Positives| \cdot fpr = |Positives| \, (c \cdot (1 - fpr) + 1 - tpr)$$

Applying this result to the m formula we have:

$$m = Imp(pos, neg) - |Positives| \, (tpr + c \cdot fpr) Imp(\frac{tpr \cdot |Positives|}{|Positives| \, (tpr + c \cdot fpr)}, \frac{fpr \cdot c \cdot |Positives|}{|Positives| \, (tpr + c \cdot fpr)})$$

$$- |Positives| \, (c \cdot (1 - fpr) + 1 - tpr) \cdot Imp(\frac{(1 - tpr) \cdot |Positives|}{|Positives| \, (c \cdot (1 - fpr) + 1 - tpr)}, \frac{(1 - fpr) \cdot c \cdot |Positives|}{|Positives| \, (c \cdot (1 - fpr) + 1 - tpr)})$$

Simplifying, we get the following expression:

$$m = Imp(pos, neg) - |Positives(\, (tpr + c \cdot fpr) Imp(\frac{tpr}{tpr + c \cdot fpr}, \frac{c \cdot fpr}{tpr + c \cdot fpr})$$

$$+ (c(1 - fpr) + 1 - tpr) \cdot Imp(\frac{1 - tpr}{c(1 - fpr) + 1 - tpr}, \frac{c(1 - fpr)}{c(1 - fpr) + 1 - tpr}))$$

For a given size of the dataset, this formulation is $m(Imp, tpr, fpr, c)$, as the number of positives and negatives are constant given the size and c. The criteria are then expressed using its formulation as an $Imp(p, n)$ function:

$$Entropy = -p \cdot log(p) - n \cdot log(n)$$

$$Gini = 4pn$$

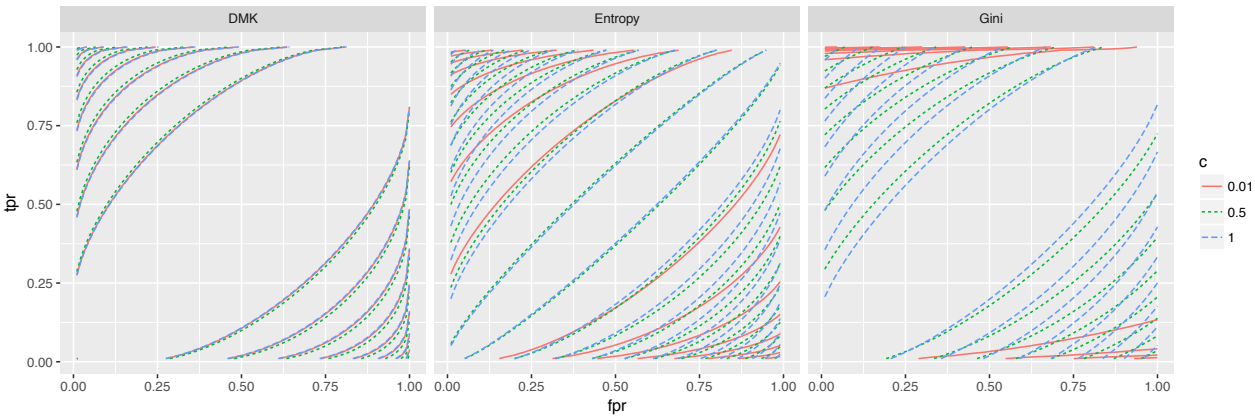

Figure 22: ROC Space for different metrics. Different colors indicate different levels of imbalance.

$$DKM = 2\sqrt{pn}$$

Figure 22 shows the ROC Space for the exposed criteria, and different levels of class skew, overimposed in the same plots. The plot shows how Entropy and Gini criteria are strongly affected by class skew, while the DMK criteria seem to be quite skew-insensitive.

Based on this framework of ROC Spaces and isometrics for comparing splitting criteria, the authors of [34] propose to use the Hellinger Distance, similiar to DMK, as the splitting criteria, as it is shown to be completely insensitive to skew. In the previous framework, the Hellinger Distance can be defined with the following expression:

$$D_H = \sqrt{\left(\sqrt{tpr} - \sqrt{fpr}\right)^2 + \left(\sqrt{1 - tpr} - \sqrt{1 - fpr}\right)^2}$$

Figure 23 shows a comparison between the isometrics of both criteria for different skew values. The figure shows that the DMK metric shows some skew-dependance for high skews, while the Hellinger distance is strictly skew independent (a fact that can be induced from the formula above).

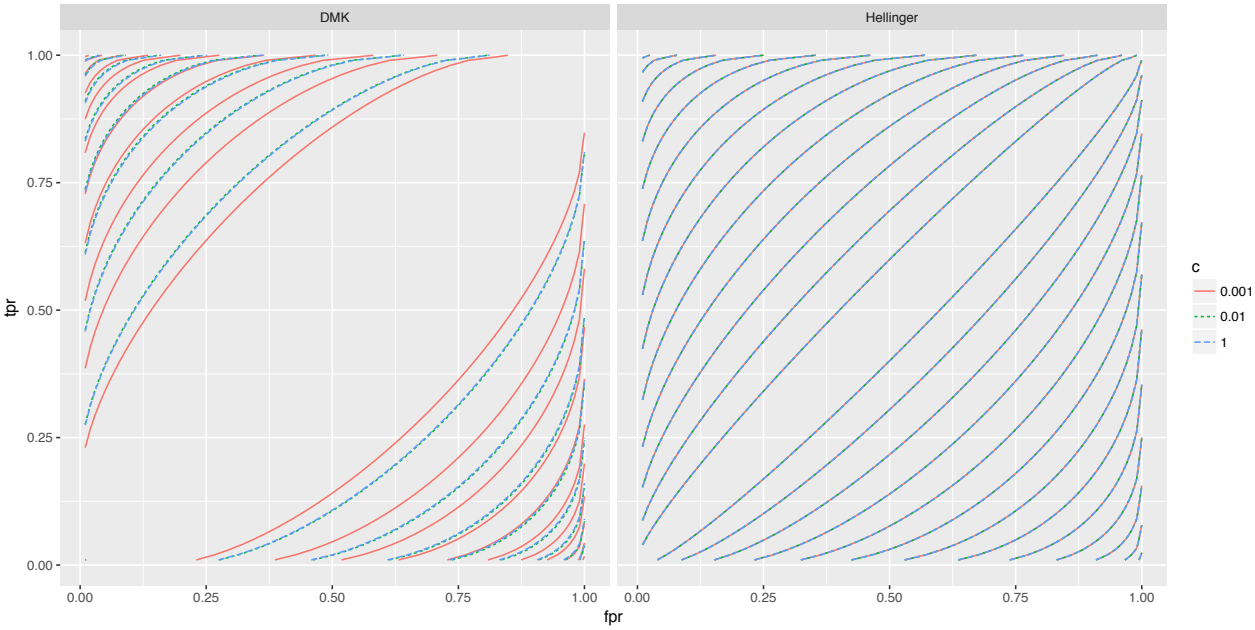

Figure 23: ROC Space comparing Hellinger Distance with DMK. Different colors indicate different levels of imbalance.

4 Introduction to Bayesian Classifiers

4.1 Overview

The motivation behind Bayesian Classifiers slightly differs from that of other learning models. While other learning models aim only at inducing a classification (or regresion) function from the data, Bayesian models aim at fully representing the underlying probability distrubution from which the dataset instances are supposed to be drawn. The discovery of the real distribution can have multiple applications:

- It servers as a knowledge discovery tool, as it gives us a hint on the causal relationships between variables in the model.
- It may help to treat missing values, as it may provide an accurate description of the conditional probability of the missing variables conditioned on the observed ones for each dataset instance.
- It gives us the ability to generate new synthetic instances from the domain, sampling from the model.
- It gives us a more general tool to perform queries from the model than other classification models. Where a simple classification model allows only to make queries of the form $P(Y|x_1, x_2, ...x_N)$ for complete data, a full joint distribution gives us the ability to query information for any of the variables of the model, conditioned on the rest.

The problem that arises in order to capture the full joint distribution over both the predictor variables and the target variables is that the number of parameters required to fully specify a full joint distribution is exponential in the number of variables. Just considering binary variables, the number of parameters needed to express the full joint distribution is $O(2^n)$. Probabilistic graphical models overcome this problem by using a factorization of the full joint distribution. For the simple case of a probability distribution with three variables, using basic probability theory we can write the following:

$$P(A, B, C) = P(A|B, C) \cdot P(B, C) = P(A|B, C) \cdot P(B|C) \cdot P(C)$$

If we analyze the number of parameters needed for each of these factors we will find that both expressions need the same number of parameters to be defined (7 for the case of binary values). But, if we have knowledge about the independencies between variables, some of those parameters could be omitted, and we would end up with a more compact expression for the distribution. As an example, if we had the information that, for the previous example ($B \perp C$) (B is marginally independent of C), then we have:

$$P(A, B, C) = P(A|B, C) \cdot P(B|C) \cdot P(C) = P(A|B, C) \cdot P(B) \cdot P(C)$$

So both X_2 and X_3 can be expressed with a single parameter, and the conditional over X_1 will require another four, for a total of five parameters. The expression of the independence relations over the variables will then helps us reducing the complexity of the model needed to express the probability distribution underlying our data.

Next section introduces Bayesian Networks: a probabilistic graphical model that helps us define the set of independences of a distribution, and that will be the base for the description of the Bayesian Classifiers.

4.2 Bayesian Networks

Bayesian Networks can be seen as an abstraction that encodes the independence relation between the random variables of a probability distribution. The structure of a Bayesian Network (BN) is a directed acyclic graph, in which every node is a variable, and there exists an arc between two nodes A and B if the probability of B has direct dependency on the value of A. Thus, the full network can be seen as a way of decomposing the full probability distribution of the variables in terms of conditional dependencies. This can be expressed with the chain rule for Bayesian Networks:

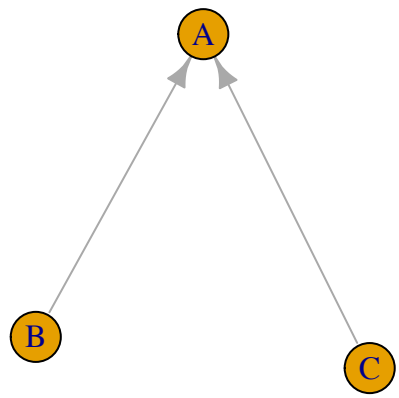

Figure 24: Basic Bayesian Network

$$P(X_1, X_2..X_n) = \prod_{i=0}^{n} P(X_i|Pa(X_i))$$

Making use of this property, the full distribution can be expressed as a set of independent Conditional Probability Distributions (CPD), one for each node of the Bayesian Network. Returning to the previous example, we can express the example distribution with the network shown in Figure 24.

Each of the two edges of this graph represents a different dependence relation between one of the variables A and the rest B and C.

In order to use it as a representation of a joint distribution, a Bayesian Network also needs a way to encode the probability distribution of each node given its parents. This local Conditional Probability Distributions (CPD) can be expressed in multiple ways (e.g.: Table CPDs, Trees, deterministic rules...) without affecting the global characteristics of the BNs. For simplicity of implementation, along this project we will mainly use Table CPDs.

4.3 The Näive Bayes Classifier

The Näive Bayes classifier is a popular simplified approach to the classification with Bayesian Networks, where some assumptions are made to reduce the complexity of the induction problem. In particular, Näive Bayes classifiers assume that:

- all the variables of the problem are conditionally dependent on the class variable,
- and all the other variables are conditionally independent.

This gives rise to a Bayesian Network that is just a tree, with the class being the root and all the other variables the leafs of the tree, as shown in Figure 25. Using the chain rule for BNs, we get the following simplified expression:

$$P(X_1, X_2..X_n, C) = P(C) \prod_{i=0}^{n} P(X_i|C)$$

In order to make a prediction we would like to get the probability of the class conditioned on the observations, so we make use of the Bayes Rule:

$$P(C|x_1, x_2, ...x_n) = \frac{P(x_1, x_2, ...x_n|C)P(C)}{P(x_1, x_2, ...x_n)} = \frac{P(C)\prod_{i=0}^{n} P(X_i|C)}{P(x_1, x_2, ...x_n)}$$

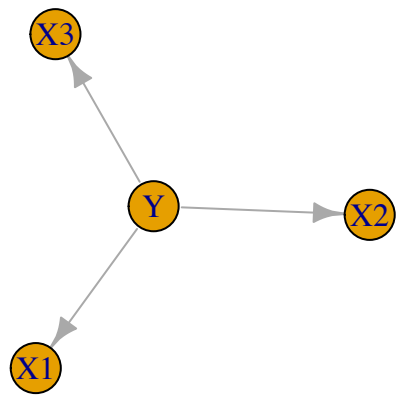

Figure 25: Basic Bayesian Network

Where we have made use of the fact that all of the x_i are independent.

Despite its popularity, the Näive Bayes approach has some problems to it as the structure assumptions may be too strong in multiple scenarios, leading to a poor classification performance. This assumptions may be relaxed without resorting to a fullly unrestricted BN (and several algorithms have been proposed, restricting the BN to trees, restricting the number of arcs or parents), but this changes to the set of assumptions come at the cost of losing the simplicity of the original algorithm. Another important caveat is that the Näive Bayes model loses the knowledge discovery and representation function of other Bayesian models, as it gives us no hint on the dependency structures inside the model.

4.4 Structure induction in Bayesian Networks

For scenarios where the simplification assumptions of the Näive Bayes model are too strong, or where we want to use the Baysian model as a tool for knowledge discovery or instance sampling, we need a way to induce the independence structure of the random variables given tha data. Unfortunately, the general case of full unrestricted Bayesian Network structure induction based in a dataset has been proved to be an NP-Hard problem on the size of the dataset. Thus, a suboptimal algorithm has to be used as a practical way of structure inference. Two general approaches can be taken to this problem:

- To add restrictions to the space of Bayesian Networks that can be explored with the algorithm, as it was done for the case of the Näive Bayes algorithm (e.g.: induce only trees, induce networks with no more than k parents, etc.).
- Or to use a metaheuristic algorithm to induce an unrestricted Bayesian Network, that will capture an approximate model. This option requires the use of a scoring metric for the different generated networks.

Along this project we will use the second approach, studying different metaheuristics in order to optimize the selected metrics, while we maintain the total training time as low as possible.

4.4.1 Metaheuristic search

The metaheuristic algorithms perform a search over the complete space of bayesian networks for the variables of the problem. The first requirement for this search is a set of operations that, given a Bayesian Network, can produce another valid BN. Being a graph, this set of operations will consist in graph operations with a series of restrictions to ensure that the result remains a BN. This is the implemented set:

- Adding an arc between two edges that do not share an arc.
- Changing the direction of an arc.

- Removing one of the arcs of the BN.

All the metaheuristics will share this same set of operations. The following sections describe the different implemented metaheuristics.

Most of the metaheuristic algorithms require an operation to generate a completely random BN to start the search process. These BNs are generated by filling an adjacency matrix with random 1's and 0's based on a given probability (the lower the probability the smaller the average number of arcs in the randomized BNs). We will keep this probability low in order to encourage simpler BNs.

4.4.1.1 Hill Climbing

The most basic metaheuristic that has been implemented is the Hill Climbing algorithm. This algorithm starts with a random candidate graph, and proceeds in the following way:

- It generates all the possible graph candidates that can be achieved by executing a single operation in the previous best graph.
- The given score function is executed for every neighbour BN, and the resulting scores are sorted.
- The best of the neighbour graphs is selected as the next best graph.
- If there is no neighbour with a higher score than the current candidate, or if the limit of iterations have been achieved, the algorithm stops.

This algorithm performs a simple full local search in the neighbourhood of the initial solution, and will be used as the base solution for the comparisons.

4.4.1.2 ILS with Randomized Bias

The Iterated Local Search pattern is a metaheuristic pattern that follows this template:

1. A random solution is created as the first step and its score calculated. This solution is set as both the current and best candidate.
2. The current candidate is perturbed N times, and the score is calculated for each perturbed solution.
3. One of the perturbed solutions is selected.
4. An acceptance criteria is applied to the solution. If the criteria is not met, the current solution remains the same.
5. If the termination criteria is met, the algorithm finish. In other case, it returns to step 2.

This template can be applied to different domains by defining three functions:

- A perturbation function, that generates solutions near the current one, to move around the solution space. In our case, the perturbation function is just the function that applies the aforementioned graph operations over the BN, with the given restrictions.

- A selection function. Typically, candidate scores are sorted and the best one is chosen, but adding a random probability of choosing a different element of the list bellow the first place has been shown to improve the results of this kind of metaheuristic, by providing more ways of exploring the space of solutions. In our case, in the spirit of [35], we will use a geometric distribution to select the candidates from the list ordered with the different score functions.

- An acceptance criteria. Also following [35] even in the case in which the selected candidate is not better than the previous one, the algorithm may continue with the found candidate, provided the difference between the current score and the best one is below the last increase in the score.

- A termination criteria. In this case, the algorithm will stop if it iterates more than M times without any improvement, or if a maximum number of iterations is reached.

4.4.2 Scoring functions

All the metaheuristic algorithms direct their search based on a fitness score for the candidate BNs. Although the literature propose several alternative score functions, most of them are based on measuring the Mutual Information between variables:

$$I(X,Y) = \sum_{x \in X, y \in Y} P(x,y) log(\frac{P(x,y)}{P(x) \cdot P(y)})$$

This is the formula for the Maximum Likelihood Estimation:

$$MLE(X,Y) = M \cdot \sum_{i=1}^{n} I(X_i, Pa(X_i^G)) - M \sum_{i=1}^{n} H(X_i)$$

Where M is the total number of examples in the dataset and $H(X_i)$ is the entropy. In order to ease the computation of the formula, the sum over the mutual information can be expressed in terms of the graph adjacency matrix. If V is the set of variables of the problem, $A = \{a_{i,j}/v_i, v_j \in V\}$ is the adjacency matrix of the BN's graph, and $I_M = \{i_{i,j} = I(v_i, v_j)/v_i, v_j \in V\}$ is the matrix of all mutual informations between variables of the problem, we can express the MLE formula as:

$$MLE(X,Y) = M \cdot \left(\sum_{i=1}^{n} A^T \cdot I_M - \sum_{i=1}^{n} H(X_i) \right)$$

Where we have taken advantage of the fact that the columns of an adjacency matrix determine the parents for each node of a graph.

4.5 Parameter induction in Bayesian Networks

Structure induction algorithms provide a way to identify a set of good independence relations between the different variables of the problem. But, in order to have a complete statistical model of the problem at hand, given the data, a set of CPDs must be induced, one for each variable. The fact that all the CPDs (Conditional Probability Distributions) are conditionally independent let us face the problem of parameter estimation considering just the subgraphs of each node and its parents. Since all the variables we will consider for classification are categorical (numerical ones are previously discretized), the target of the estimation will be to find the parameters of a multinomial distribution (one different multinomial distribution for each variable), that most likely fit the data.

We will consider in the first place, for simplicity, the case of a variable without any parents. If we define $P(X = x_i) = \theta_i$ as the probability of each of the k possible values of the variable, we can define the probability of a daset, given the parameters as:

$$L(X|M) = P(x_1, x_2, ...x_m|\theta) = \prod_{j \in D} \theta_{x[j]} = C \prod_{i \in X} \theta_i^{M_i}$$

That is, the probability of a dataset is equal to the probability of each value of the variable to the power of the number of appereances of that variable (times the number of combinations, that is constant with respect to θ and will not influence the maximization). In this context, the task of learning the parameters can be thought of as searching the value of θ that maximizes the probability of the data given the parameters. Using standard optimization methods we get the following expression for the maximum value of θ_i:

$$\theta_i = \frac{M_i}{M} \quad \forall i \in X$$

Where M is the total number of observations in the dataset and M_i the total number of observations for value i.

We can adapt this result to the case of a full Bayesian Network by using the chain rule to split the probability distribution into a product of local probability distributions:

$$L(X|M) = P(m_1, m_2..m_m|\theta) = \prod_{i=0}^{m} P(m_i|\theta) = \prod_{i=0}^{m}\prod_{j=0}^{n} P(x_j[i]|\theta_j) = \prod_{j=0}^{n} L_{loc}(X_j|\theta_j)$$

For each of the local likelihood functions, and considering that all the θ_i are independent of each other, we can perform exactly the same analysis we did in the one variable case, but instead of considering the number of occurrences of each value k of a variable X_1, M_{k,X_1}, considering the number of simultaneus occurrences of a certain set of parent values u_{X_1} for each value of the variable, $M_{k,X_1|u_{X_1}}$. This lead to the final formulation for the MLE for parameters of BN's:

$$\theta_{k,X_j|u_{X_j}} = \frac{M_{k,X_j|u_{X_j}}}{M} \quad \forall k \in X_j$$

Bayesian Estimation improves the MLE approach by considering each of the parameters as a random variable in its own. In this way, if we consider that all the θ_i are conditionally independent, we can rewrite the probability distribution as:

$$P(m_1, m_2..m_m, \theta) = \prod_{i=0}^{m} P(m_i|\theta_i)P(\theta_i)$$

And use the Bayes theorem to calculate the posterior probability of the parameters given the data:

$$P(\theta|m_1, m_2..m_m) = \frac{P(m_1, m_2..m_m|\theta)P(\theta)}{P(m_1, m_2..m_m)}$$

where the first part of the numerator is just the expression for the multinomial MLE we have just seen. This approach has the advantage of introducing an explicit prior over the parameters, that we can use to introduce external knowledge about the problem into our model. The most common probability distribution for Bayesian Inference of parameters is the Dirichlet distribution (for reasons that will be clear later on). The Dirichlet distribution is a function of k hyperparameters, that can be expressed as:

$$P(\theta) = \frac{1}{Z}\prod_{i=1}^{k} \theta_i^{\alpha_i - 1}$$

Where Z is the partition function makes the function a valid probability density distribution (integrating over all values). If we now ignore the partition functions for both the posterior and the prior, we have the following results:

$$P(\theta|m_1, m_2..m_m) \propto P(m_1, m_2..m_m|\theta)P(\theta)$$

$$P(\theta) \propto \prod_{i=1}^{k} \theta_i^{\alpha_i - 1}$$

$$P(m_1, m_2..m_m|\theta) \propto \prod_{i=1}^{k} \theta_i^{M_{X_i}}$$

From which we can infer that:

$$P(\theta|m_1, m_2..m_m) \propto \prod_{i=1}^{k} \theta_i^{M_{X_i}+\alpha_i-1}$$

so the posterior, as well as the prior, follows a dirichlet distribution, but with metaparameters $(M_{X_1} + \alpha_2, M_{X_2} + \alpha_1 ... M_{X_n} + \alpha_n)$. We can then conclude with the following expression for the probability for each value for each variable in the m_{n+1} example:

$$P(X_n[m_{n+1}] = x_i) = \frac{\alpha_i + M_i}{\alpha + M}$$

Where $\alpha = \sum_i \alpha_i$ is the equivalent sample size and M is the total number of elements in the data set. This result is consistent with the intuitive value for the probability as the number of instances having the value x_i over the total number of instances, modified to add certain number of virtual instances that reflect our prior knowledge.

4.6 Classification with Bayesian Networks

Once a full Bayesian Network has been inducted (both its parameters and structure), the Bayes Theorem can be used to calculate the probability of a particular value of the class given the evidence:

$$P(C|X_1, X_2, ...X_n, \theta) = \frac{P(C) \cdot P(X_1, X_2, ...X_n|C, \theta)}{P(X_1, X_2, ...X_n|\theta)} = \frac{P(C) \cdot \prod_i P(X_i|Pa(X_i), C, \theta)}{P(X_1, X_2, ...X_n|\theta)}$$

The product in the numerator coming from the chain rule for Bayesian Networks. When complete evidence for all the features is given, each factor in the product can be read from the table CPD; and an estimation of the marginal probability of the class value can be obtained from the training dataset. The probability in the denominator is just a proportionality constant, and in the case of binary classification that we are considering, we can omit its calculation, and renormalize, i.e.: we will calculate $\tilde{P}(c_i|X_1, X_2, ...X_n, \theta)$ for all class values c_i, and then we will renormalize them to get a valid probability distribution.

One practical problem that is usually encountered when using Table CPDs is that for some combinations of values there is no appareance in the training dataset, and thus the table would assign those a zero probability. In order to avoid zero-valued probabilities (that have been shown to cause practical problems in classification with Bayesian Networks; see [3]), a minimum frequency of 1 will be assigned to any combination of values (a practice usually called Laplace Smoothing).

5 Analysis of the effect of class imbalance in the induction of FBCs

5.1 Overview

A typical approximation to the solution of the imbalance problems in Bayesian models (and most others) is to preprocess the dataset before training the model, usually by using resampling methods. An example of this kind of solutions applied to Bayesian Networks can be found in [36]. This paper shows that this methods can be used to greatly improve the classification performance of the induced models. But, as we will show in next sections, the effect of resampling can only impact the performance of the classifier by changing the sufficient statistics (or the priors if Bayesian Information Criteria is used to estimate the parameters), so there should be ways of tailoring the learning algorithm to account for these changes, without the need for modifying the incoming data. Analysis such as the one performed by [34] also encourages us to try and isolate the Imbalance Rate term in the objective functions, in order to directly reduce its influence in the learning process.

In section 5.4, we will also evaluate the use of ensembles of models, taking advantage of the inherent variance of heuristic models and on the ability of the BCs to compute the likelyhood of a test case for a given model, given the training data.

5.2 Impact of the imbalance in the structure induction

Bayesian structure induction is usually based on scoring formulas that depend on the concept of mutual information between variables, from Information Theory. This concept relates the marginal probabilities of certain values of the variables with the joint probability of each of the values, using the following formula:

$$I(X,Y) = \sum_{x \in X, y \in Y} P(x,y) log(\frac{P(x,y)}{P(x) \cdot P(y)})$$

In the special case of the structure induction algorithms, those probabilities are derived from sufficient statistics M_x, M_y, $M_{x,y}$ that involve the count of the number of occurrences of each variable combination in the training dataset:

$$P(x,y) = \frac{M_{x,y}}{\sum_{i \in X, j \in Y} M_{i,j}} = \frac{M_{x,y}}{M}$$

$$P(x) = \frac{M_x}{\sum_{i \in X} M_i} = \frac{M_x}{M}$$

Substituting in the above formula, we get the following result:

$$I(X,Y) = \sum_{x \in X, y \in Y} \frac{M_{x,y}}{M} log(\frac{\frac{M_{x,y}}{M}}{\frac{M_x}{M} \cdot \frac{M_y}{M}}) = \frac{1}{M} \sum_{x \in X, y \in Y} M_{x,y} log(\frac{M_{x,y}}{M_x \cdot M_y})$$

In order to analyze the impact of the class distribution in the structure induction, we must now take into account how this mutual information gets affected by particular values of one of the variables (the class in this case). In the following, we will suppose that Y is the class variable without loss of generality (as mutual information is simmetric).

In the simple binary case, we would have the following expansion of the mutual information expression:

$$I(X,Y) = \frac{1}{M} \left(\sum_{x \in X} M_{x,pos} log(\frac{M_{x,pos}}{M_x \cdot M_{pos}}) + \sum_{x \in X} M_{x,neg} log(\frac{M_{x,neg}}{M_x \cdot M_{neg}}) \right) =$$

$$= \frac{1}{M}\left(\sum_{x\in X}\left(M_{x,pos}log(\frac{M_{x,pos}}{M_x}) - M_{x,pos}log(M_{pos})\right) + \sum_{x\in X}\left(M_{x,neg}log(\frac{M_{x,neg}}{M_x}) - M_{x,neg}log(M_{neg})\right)\right) =$$

$$= \frac{1}{M}\left(\sum_{x\in X} M_{x,pos}log(\frac{M_{x,pos}}{M_x}) + \sum_{x\in X} M_{x,neg}log(\frac{M_{x,neg}}{M_x}) - (M_{pos}log(M_{pos}) + M_{neg}log(M_{neg}))\right)$$

The imbalance rate can be expressed as the quotient of both values of the class variable, i.e.:

$$IR = \frac{M_{pos}}{M_{neg}} = \frac{M - M_{neg}}{M_{neg}} = \frac{M}{M_{neg}} - 1$$

$$M_{neg} = \frac{M}{IR+1}$$

$$M_{pos} = M - M_{neg} = M - \frac{M}{IR+1}$$

We can now reformulate the expression $M_{pos}log(M_{pos}) + M_{neg}log(M_{neg}$ as a function of the imbalance rate, IR and the number of cases M:

$$M_{pos}log(M_{pos}) + M_{neg}log(M_{neg}) = \left(M - \frac{M}{IR+1}\right)log\left(M - \frac{M}{IR+1}\right) + \frac{M}{IR+1}log\left(\frac{M}{IR+1}\right)$$

Then, we can express the mutual information as a function of the IR:

$$I(X,Y) = \frac{1}{M}\left(\sum_{x\in X, y\in Y} M_{x,y}log(\frac{M_{x,y}}{M_x}) - M \cdot IR_{term}(M, IR)\right)$$

$$IR_{term}(M, IR) = \left(M - \frac{M}{IR+1}\right)log\left(M - \frac{M}{IR+1}\right) + \frac{M}{IR+1}log\left(\frac{M}{IR+1}\right)$$

We can graphically analyse the behavior of this term for different levels of M, as depicted in Figure 26.

The graph suggests some of the predictable interactions of the IR with the scoring algorithm:

- The IR term has a minimum at 1.0, where both classes have exactly the same number of examples.
- The term grows larger as the class values get more unbalanced, in both directions (it grows to infinity in both extremes of the domain, 0 and ∞.
- The shape of the function is the same with independence of the number of examples, but the effect of the imbalance as a function of M grows bigger with M.

These interactions suggest the use of a rectified Mutual Information function may be used to define a new MLE or BIC scoring function that should be less affected by the imbalance in the datasets.

5.3 Impact of the imbalance in the parameter estimation

5.3.1 Previous considerations

We will begin by considering how the probabilities for each value of a certain variable are calculated in the case of Bayesian Estimation. For the case of a binary valued variable, that depends on a set of parents U we have the following expression for the probability of a certain value:

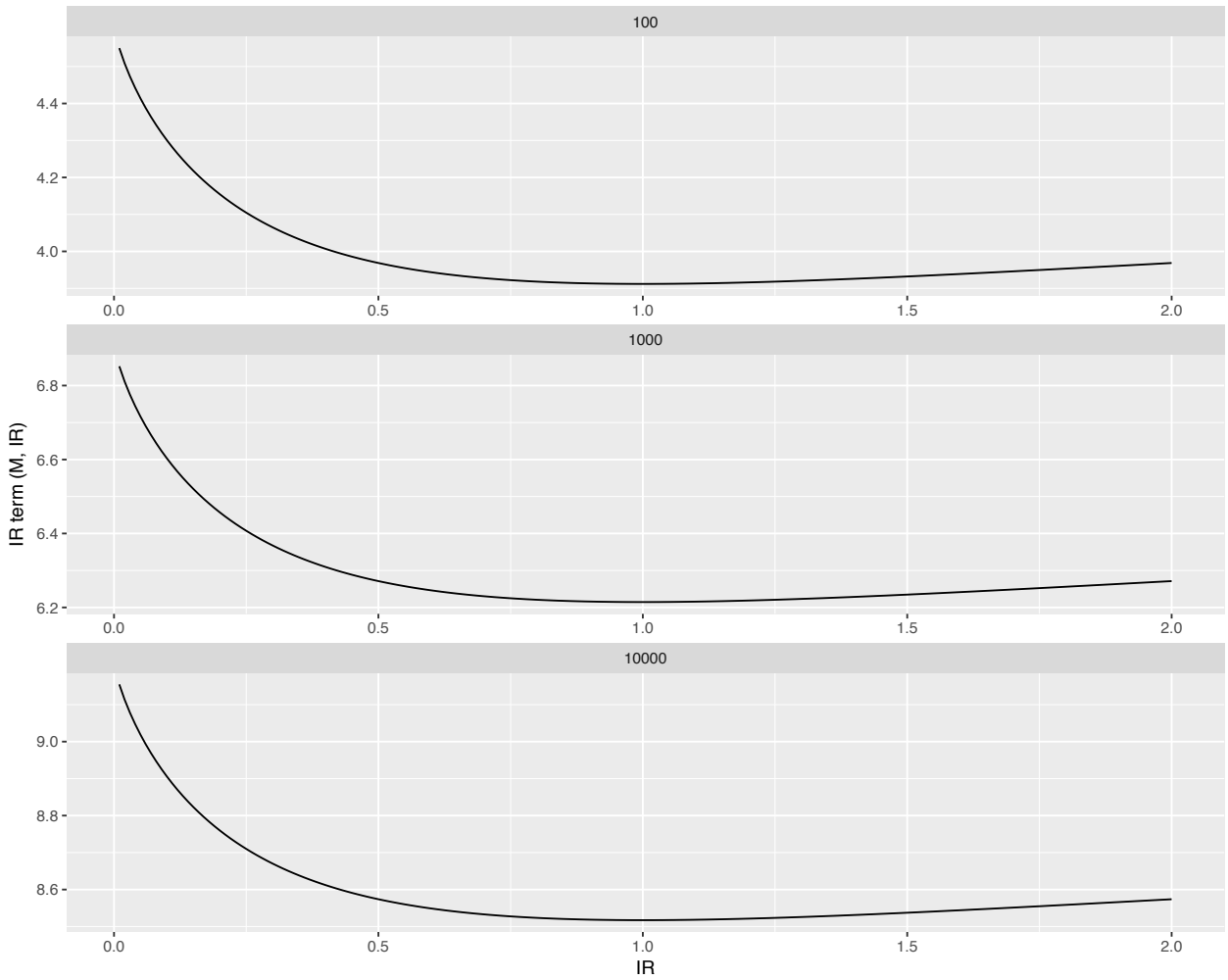

Figure 26: Value of the $IR_{term}(M, IR)$ value for differents levels of Imbalance Rate and M

$$P(X = x_1|u) = \frac{\alpha_{x_1|u} + M_{x_1|u}}{\alpha + M}$$

The existance of an imbalance is defined as the case where the marginal frequencies of class value x_1 are a lot higher than those of x_2, that is:

$$M_{x_1} = \sum_u M_{x_1|u} >> M_{x_2} = \sum_u M_{x_2|u}$$

The appeareance of those conditional frequencies in the expression of the imbalance give raise to some of the problems we found in the literature regarding imbalance:

- Global class imbalance reveals a general lack of data about a certain class, but doesn't take into account how this lack of data is distributed. This means that, with the same level of class imbalance, it can be the case that, for a particular sets of values of the parents, u_i, the minority class x_2 has a higher number of occurrences than the majority class x_1, while keeping the imbalance condition correct, as long as considering the total set of parent's values, the sum for x_1 is higher than x_2.
- An analysis based solely in global class imbalance would miss the interclass imbalance problem, i.e.: different sets of values of the parent's variables (that will determine different disjuncts), u_i, may have a much lower number of instances than other, so even in the case where the imbalance rate seems low, there may be small disjuncts that would lead to a poor performance.

An interesting advantage of the Bayesian Estimation approach is that it gives us a powerful tool for fine-grained manipulation of the probabilities of the different classes, by manipulating the Dirichlet Priors of the parameters, as well as the equivalent sample size α. It can also provide a tool to make an online examination of some of the real problems behind the lose of performance caused by the imbalance problems, as the small disjuncts problem. It's worth noting, however, that, given a fixed equivalent sample size, the effect of the priors will vanish as the number of examples in the dataset grows (as the number of examples in each class overgrows the fixed metaparameter α_i). Any action over the priors should then make convenient modifications in the equivalent sample size, to achieve its effect.

5.3.2 Theoretical comparison with sampling techniques

Before delving into the analysis of the prior modifications, an analysis of how the sampling techniques affect the parameter estimation is worth making. As we have seen, probability estimation depends basically on a set of priors and a count of the number of appereances of different sets of values in the dataset. The only direct effect that a modification of the dataset can have in the parameters, in the case of Bayesian Learning algorithms, is the modification of the count number, $M_{x_i|u}$.

In this sense, the effect of Random Undersampling, where n instances of the majority class are removed from the dataset, would be to substract n from the marginal frequencies M_{x_1}. As it was the case when we analyzed the effect of imbalance, this random action over the marginal frequencies can have very different effects, depending on which $M_{x_1|u}$ affects, and, being a random algorithm, there is little control over which disjunct it will affect. The case of Random Oversampling is very similar, but adding a constant to the marginal frequencies of the minority class instead, M_{x_1}. The problem with this solution is similar, as random additions may skip the small disjuncts where the problem of imbalanced data is particularly severe, increasing the frequencies for already well represented disjuncts instead.

There is another set of resampling technics that aim to reduce the effects of class imbalance by generating new synthetic intances of the minority class, based on the current ones, like the popular SMOTE algorithm. The effect of this algorithms is usually based in the continous nature of variables, generating instances that are near the existing ones with some interpolation mechanism. This action has the effect of increasing the example density in the minority class disjuncts. In the case of the Bayesian classifiers that work only with categorical data, all the numerical variables get discretized, and so, these oversampling methods will increase the count selectively for each disjunct.

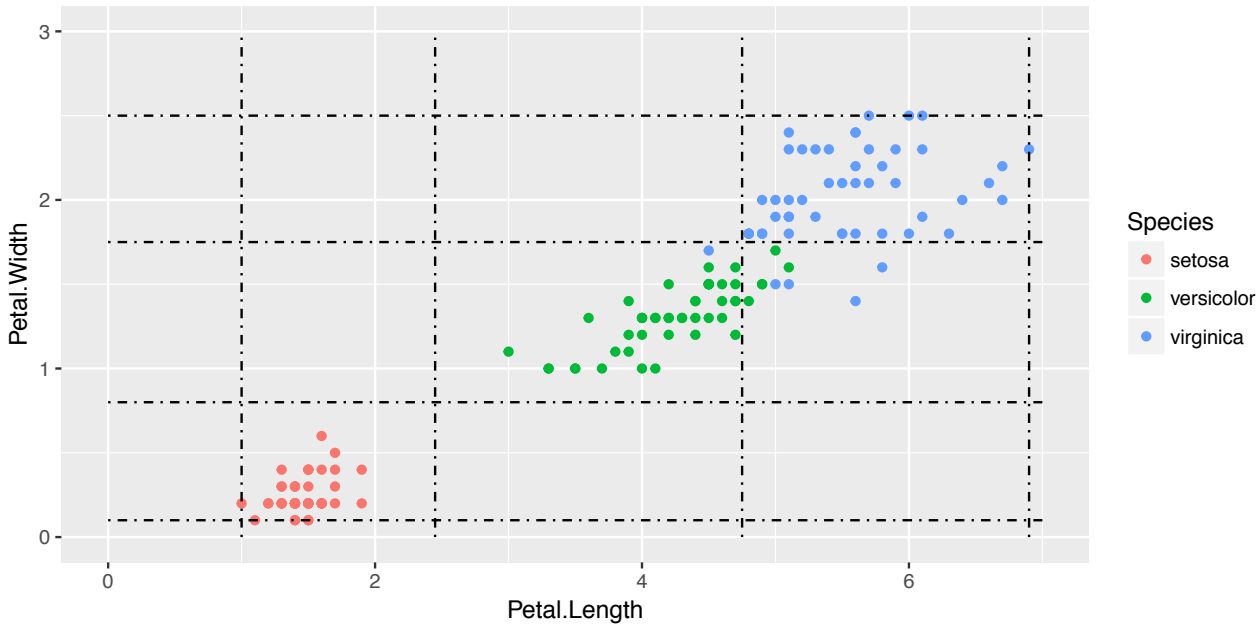

Figure 27: Discretization of the Iris dataset following the TopDown algorithm

Figure 27 shows a discretization of the Iris algorithm based on the TopDown discretization algorithm (just two of the variables are considered to make the representation easier). As we can see, the discretization algorithm has already clustered the observations belonging to the different classes. There are three disjuncts that belong clearly to one particular class:

- Observations in the disjunct roughly centered at $(1.8, 0.4)$ belong to the Setosa species.
- Observations in the disjunct roughly centered at $(3.6, 1.5)$ belong to the Versicolor species.
- Observations in the disjunct roughly centered at $(5.9, 2.2)$ belong to the Virginica species.

There is another disjunct, roughly centered at $(5.9, 1.5)$ that contains few observations of two species. If we consider that our minority class was Virginica, we would find two effects of the SMOTE algorithm:

- Since the discretized square of the domain that contains most of the virginica instances is adjacent to the disputed square, some examples would be randomly created in this square, when at least one of the randomly chosen pairs of minority elements belong to it.
- In those cases where both minority elements lie in the rectangle that is already well defined, redundant elements would be created. The corresponding effect on the Bayesian Classifier would be that the corresponding frequency $M_{x_{virginica}|u}$ would be increased.

This example ilustrate another important fact of the classification with discrete variable Bayesian Networks: the discretization algorithm will have an important effect on the performance of the complete algorithm. As an example, Figure 28 shows the discretization of the same dataset using the Equal Frequency algorithm. As it can be seen, the created categories are very different, and some of them are clearly more ambiguous than those found by the TopDown algorithm. Even though in this case, considering the two removed variables could help desambiguating the disjuncts, it should suffice to show that the discretization algorithms play an important role in disjunct definition for classification, and thus should be taken into account when looking for a classifier that correctly takes into account class imbalance.

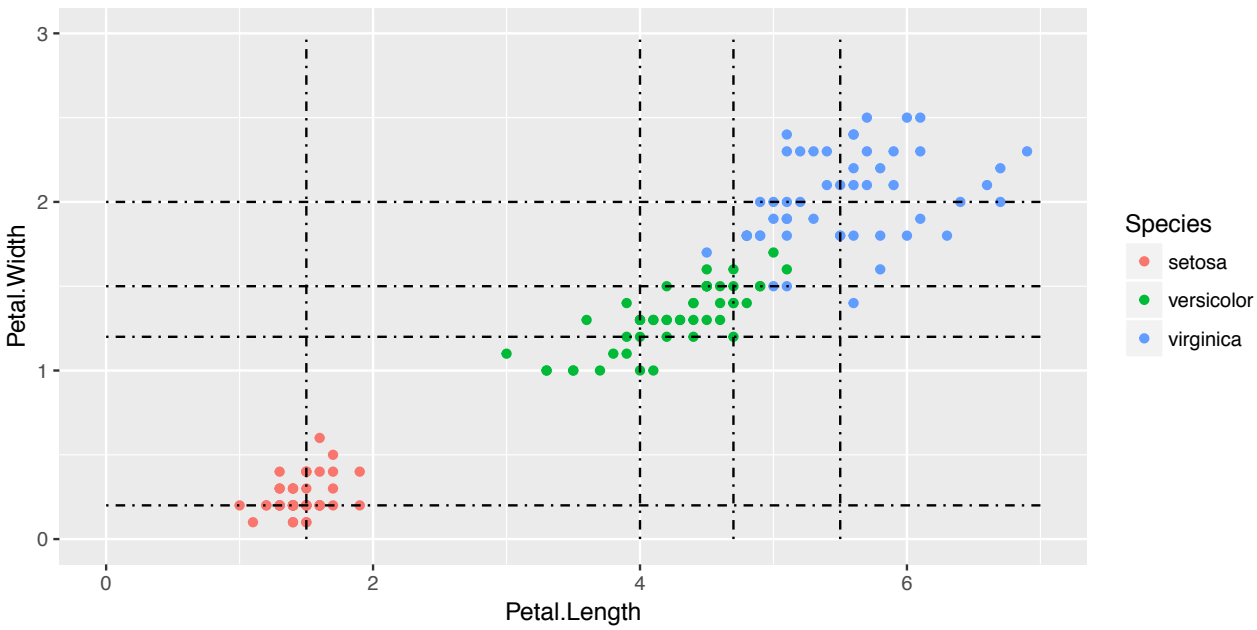

Figure 28: Discretization of the Iris dataset following the qual Frequency algorithm

5.4 Algorithm proposals

5.4.1 Rectified structure induction score

As we saw earlier, the imbalance rate has an impact in the induction of the structure of the Bayesian Networks, that can be analytically deduced from its dependence on the Mutual Information. A possible way of removing this dependence is to express this mutual information directly in its IR version, so the we can use a reduction constant on that term to account for the imbalance rate damping we would like to apply to our structure induction algorithm. The new expression for the MI would be the following:

$$I(X,Y) = \frac{1}{M} \sum_{x \in X, y \in Y} M_{x,y} log(\frac{M_{x,y}}{M_x}) - \phi \cdot IR_{term}(M, IR)$$

where ϕ is the dampening metaparameter and IR_{term} is defined as:

$$IR_{term}(M, IR) = \left(M - \frac{M}{IR+1}\right) log \left(M - \frac{M}{IR+1}\right) + \frac{M}{IR+1} log \left(\frac{M}{IR+1}\right)$$

As we can see, this term doesn't depend on the particular values of $x \in X$ so it can be calculated just once for each variable. This same approach can be applied just for the class variable as the Y variable or to all the variables, for their particular imbalance rates (that would pose the same kind of problems as the *class* variable regarding structure induction). In our case, we will apply the same treatment to all the variables, in order to try to induce the most appropriate structure for the network.

5.4.2 Parameter estimation priors

5.4.2.1 Acting globally over the imbalance rate

For those cases were the imbalance rate may be the main problem causing a decrease in the classification performance, the Bayesian Estimation paradigm offers an easy way of rectifying the class balance. Since the probability of detecting a positive class value is defined as:

$$P(X = x_1|u) = \frac{\alpha_{x_1|u} + M_{x_1|u}}{\alpha + M}$$

modifying the Dirichlet priors lets us change the overall class imbalance. For the cases where there is no knowledge of the real priors, these are usually defined as:

$$\alpha_{X_C|Pa(X_C)} = \frac{\alpha}{\prod_{n \in Pa(X_C)} k_n}$$

where α is the equivalent sample size and k_n is the number of possible values of variable X_n and X_C is the target class variable. In order to rectify the imbalance, a new set of priors is defined so that the equivalent sample size is not changed. For the case of balanced priors for a binary variable:

$$\alpha = \alpha_{x_1} + \alpha_{x_2} = \sum_{u \in Pa(X_C)} \frac{\alpha}{\prod_{n \in Pa(X_C)} k_n} + \sum_{u \in Pa(X_C)} \frac{\alpha}{\prod_{n \in Pa(X_C)} k_n} =$$

$$\sum_{u \in Pa(X_C)} \frac{C_1 \cdot \alpha}{\prod_{n \in Pa(X_C)} k_n} + \sum_{u \in Pa(X_C)} \frac{C_2 \cdot \alpha}{\prod_{n \in Pa(X_C)} k_n} = (C_1 + C_2) \cdot \sum_{u \in Pa(X_C)} \frac{\alpha}{\prod_{n \in Pa(X_C)} k_n}$$

Where we have introduced two variables that sum to two in order to maintain the equalities. This constants will be the ones used to rectify each prior, with the following formula:

$$\alpha'_{x_1|Pa(X_C)} = \frac{(1 - IR \cdot \gamma)\alpha}{\prod_{n \in Pa(X_C)} k_n} \quad ; \quad \alpha'_{x_2|Pa(X_C)} = \frac{(1 + IR \cdot \gamma)\alpha}{\prod_{n \in Pa(X_C)} k_n}$$

$$C_1 = 1 - IR \cdot \gamma \quad ; \quad C_2 = 1 + IR \cdot \gamma$$

Where γ is a metaparameter that controls the amount of correction based on the IR.

5.4.2.2 Acting locally over small disjuncts

As we saw when discussing the similarities between disjunct frequencies manipulation and resampling based on generation of new synthetic samples, these oversampling techniques acted by increasing the count of elements of one class for particular values of the predictor variables. But even if algorithms like SMOTE generate more selective examples than ROS, in the root of this algorithms lies an assumption that may itself be problematic: that the lack of performance (measured with an enfasis in the positive class recall, or measures including it, like F1) can be improved by fixing the class imbalance (whether it is global or local to a series of disjuncts). The problem with this approach is that real datasets usually suffer from other problems like overlap between classes or random noise that may increase the difficulty of classifiying ambiguous disjuncts (defined as clusters of observations that contain examples of both classes, but most probably more of the majority class).

This makes the task of defining what a small disjunct is a hard one. In order to have some range of control over what our algorithm considers a small disjunct, and what it considers random noise, two metaparameters will be defined:

- ν_0 will be the lower limit for a small disjunct with a high imbalance rate to be considered a pure disjunct with noise. Noisy cases will be attributed to the class value with the highest frequency in the disjunct during prediction phase, so no action is required in these cases.
- ν_{MAX} will be the highest level of imbalance for which the algorithm will make a corrective action. This level is important for if this value is too high, all the disjuncts with any positive examples on them would be attributed to the positive class, most probably reflecting poorly the underlying distribution.

When manipulating these parameters, as it occurs frequently in highly imbalanced scenarios, some analysis must be made about what is the expected result of learning: if the only expected result consists in an identification of the positive cases, without any regard to how accurate is the resulting probability distribution, the interval $[\nu_0, \nu_{MAX}]$ can be broader; if obtaining a plausible probability distribution is important, the interval could be narrower, allowing to rectify just the cases where we believe, with a certain degree of confidence, that the imbalance between classes has been caused by a lack of the positive data, and it isn't a characteristic of the distribution in that part of the feature space.

On the practical side, the same mechanism that allows us to act over the global imbalance lets us act over the imbalance for particular disjuncts. That is, the algorithm will define a set of Dirichlet priors, $\alpha_{x_i|u}$, that will separate from the uniform distribution in a degree proportional to the imbalance rate, just for the $u \in Pa(X)$ for which:

$$\nu_0 < \frac{M_{x_2|u}}{M_{x_1|u}} < \nu_{MAX}$$

This should provide an effect resembling the one obtained by the SMOTE resampling algorithms, without the added noise of modifying the dataset (as the desired effect is directly implemented in the algorithm, instead of indirectly implemented in the resampling strategy).

5.4.3 Bayesian ensemble methods

Using ensembles for imbalance learning has come to be one of the most popular learning methods. In the particular case of the Bayesian Networks, the heuristic nature and the high dependance on the original dataset of the structure induction process, gives rise to a set of very high variance weak classifiers. These types of classifiers are very well suited for its use in Bootstrapping ensembles.

The idea of using combinations of models is not new; for example, it was suggested in [37] while speaking of model averages. In this paper on Bayesian Network induction, the authors give the idea of combining multiple models $M_1, ..M_m$ in a weighted average, where the weight for each model could be computed as the likelihood of the training data D_T given the model. Thus, when used in the context of test instance classification, we could express the conditional probabilities of the class, given the features as:

$$P(C|X_1, X_2, ...X_n) = \sum_m P(D_T|M_m) \cdot P(C|X_1, X_2, ...X_n, M_m)$$

This expression, though, was thought for its use with a given training set D_T, that would be equal for all models. In the bootstrapping scenario, training data for each of the models have a slight variation, as the data is sampled with repetition from the original training data, so that the previous expression is no longer a valid probability distribution for this case. One solution would be to calculate the weights either way, and then renormalize.

Another posibility would be to take into account the likelihood of the test data for each different model, during prediction. Since all the models used in the bootstrapping scenario are just approximations of the real underlying distribution (both because of the heuristic structure induction and because of the dataset bootstrapping process), these models may capture with different accuracy different subsets of the probability distribution space. Considering this, an option would be to weight the importance of each model in prediction time by the probability of the observed data under each model. That is:

$$P(C|X_1, X_2, ...X_n) = \sum_m P(X_1, X_2, ...X_n|M_m) \cdot P(C|X_1, X_2, ...X_n, M_m)$$

The experiments will test the bootstrapping ensembles with different weighting criteria, to compare their efficiencies.

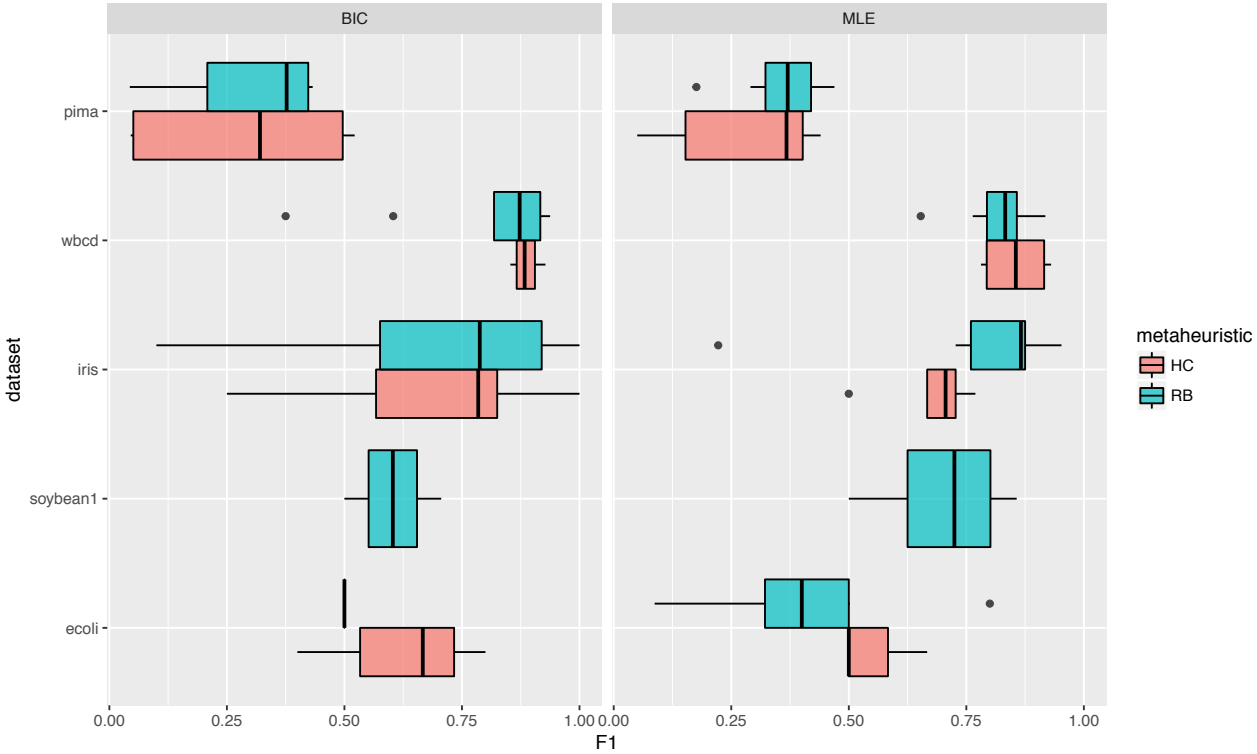

Figure 29: F1 scores for different metaheuristics per structure score type

6 Experiments

6.1 Heuristics selection

The first set of experiments aims to select the most appropriate heuristics from those described in the structure induction section. Along with the heuristics, two more general characteristics should be selected for the Bayesian Classifier: the discretization method (with two options: equal frequencies and top down discretization) and the structure score.

Figure 29 shows a comparison of the different metaheuristics for different structure score types.

Figure 30 shows the aggregated results for the different structure scoring types, by dataset. Table 1 shows the results ordered by F1 value.

Figure 31 shows the same results aggregated by discretization type.

Since the heuristics depend on certain number of metaparameters, another experiment was created to evaluate the influence of each metaparameter in the global result of the classifier. Since the number of metaparameters to evaluate is quite big, a grid search would have been a highly expensive choice. Instead, for each iteration a set of random values for each parameter was drawn from an exponential distribution. The following list shows each of the tested metaparameters and the ranges of values that were tested:

- Maximum number of iterations of the heuristic (maxIters): (40, 3000)
- Maximum number of iterations without improvement (maxWithoutImprovement): (3, 30)
- Parameter for the geometric distribution for random selection of candidates: (geomParameter) (0, 1)
- Number of candidate graphs to consider (candidateNumber): (5, 50)

The maximum time in seconds to execute the heuristic (maxTime) was kept under 30s, to make the experiments shorter.

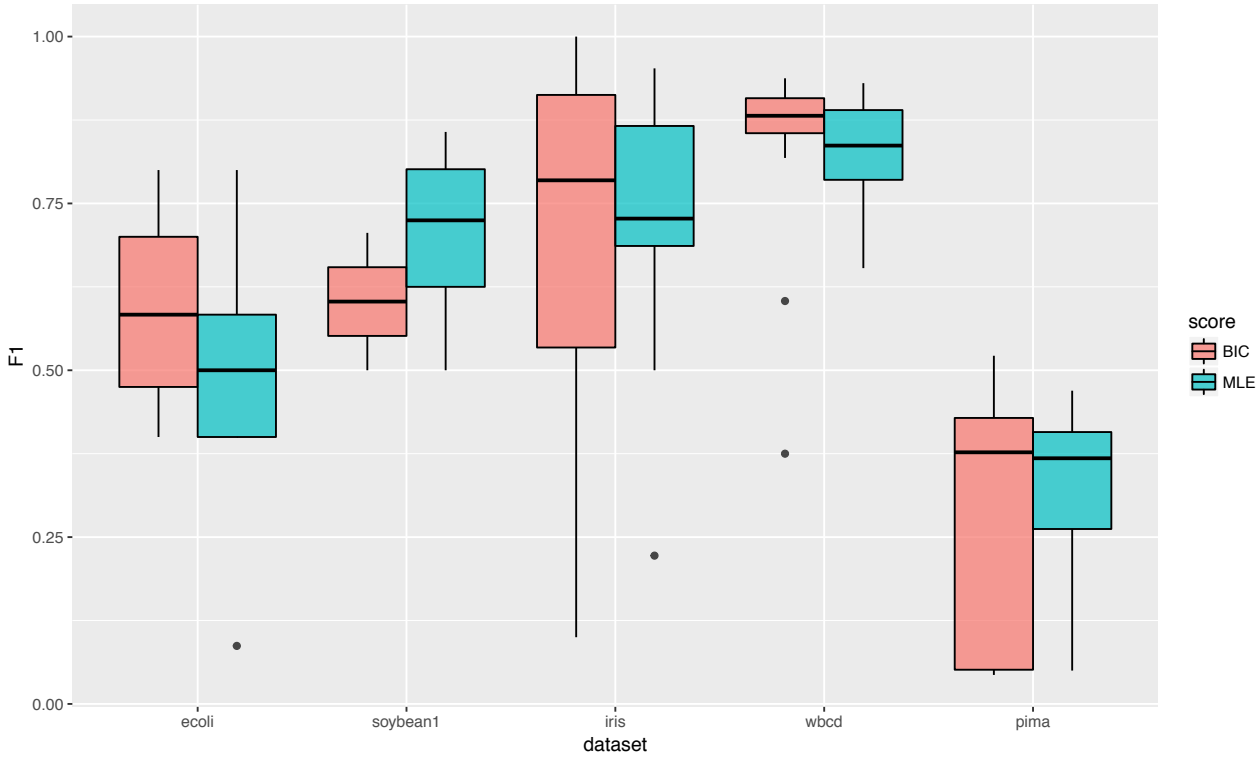

Figure 30: Results aggregated for different score types

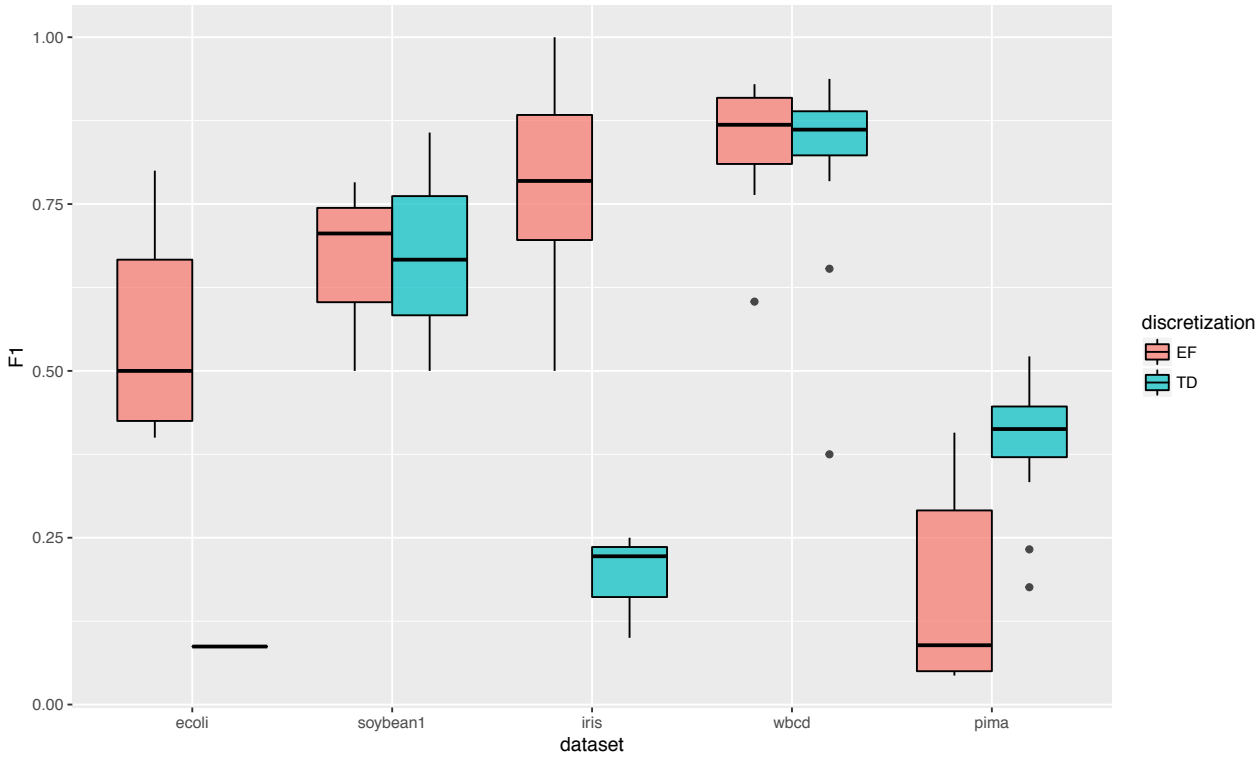

Figure 31: Results aggregated by discretization type

Table 1: Summary of F1 values for diferent metaheuristics and discretization methods

score	discretization	metaheuristic	F1
MLE	EF	RB	0.70
BIC	EF	HC	0.65
MLE	TD	HC	0.63
BIC	EF	RB	0.62
BIC	TD	HC	0.62
MLE	EF	HC	0.60
MLE	TD	RB	0.55
BIC	TD	RB	0.51

Figure 32 shows the results of the experiments for different levels of Imbalance Rate, grouped by metaparameter. The results of the experiments can be seen aggregated in Figure 33. Some conclusions can be extracted from the figures:

- First, that the appropriate values for each of the metaparameter heavily depend on the dataset. This suggests that, in order to get good results using the BC in an imbalance scenario, a Cross Validation step to determine the appropriate values for the parameters would be useful.
- Even in a scenario of high variabilty, some of the metaparameters (e.g.: *maxWithoutImprovement* and *candidateNumber*) seem to have distinctive peaks at certain values. Those peaks may be used as sensible defaults for those parameters, while using Cross Validation to optimize others like the parameter of the geometric distribution.

6.2 Impact of structure complexity in classification accuracy

The goal of this experiment was to analyze two effects:

- How the complexity of the classifier influenced the classification performance of the Bayesian Network. The complexity will be measured by the number of direct dependencies of the class variable (i.e.: the total number of edges linked to the class node).
- And whether the score used for structure induction had a correlation with the F1 classification accuracy metric.

The setup of the experiment consisted in the execution of a single Full Bayesian Classifier, with:

- all its metaparameters fixed
- the Randomized Bias heuristic for structure induction
- and the BIC criteria for parameter estimation

over the complete list of datasets. For all these datasets, apart from the usual metrics, the following ones were reported:

- the total number of edges conecting to the class variable
- the maximum value of the structure score
- the mean of the frequencies for the probabilities of the class given its parents

Figure 34 shows the number of class descendants in the induced model versus the classification performance of the model. The figure suggests that there is an optimal number of class descendants for each dataset, probably reflecting the most appropriate structure for the underlying distribution (i.e.: a simpler structure would not capture all the distribution details, while a more complex one would add noise to the induced distribution). On the other hand, Figures 35 and 36 show that the number of class children is not dependant

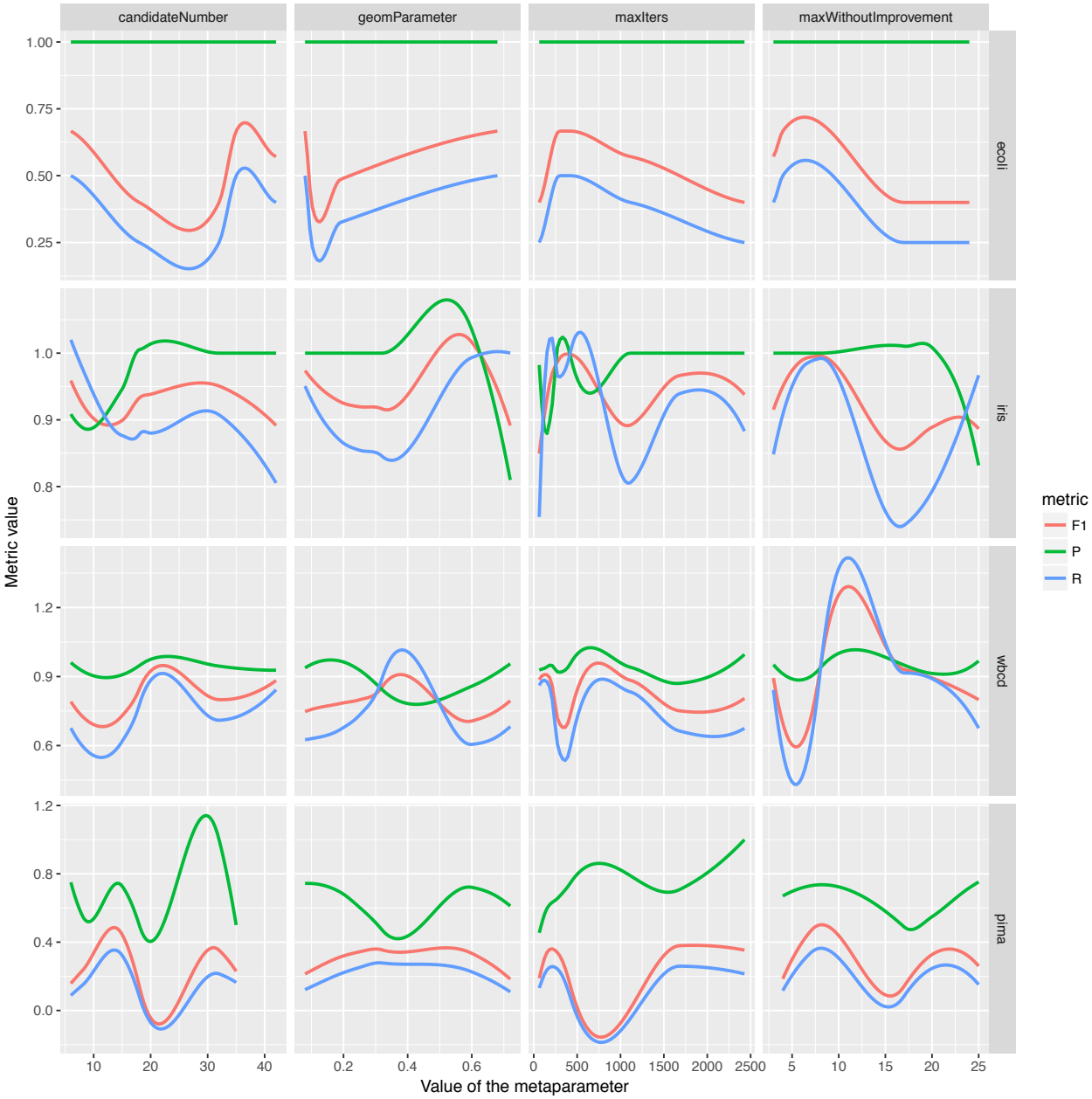

Figure 32: Metaparameter results for different datasets

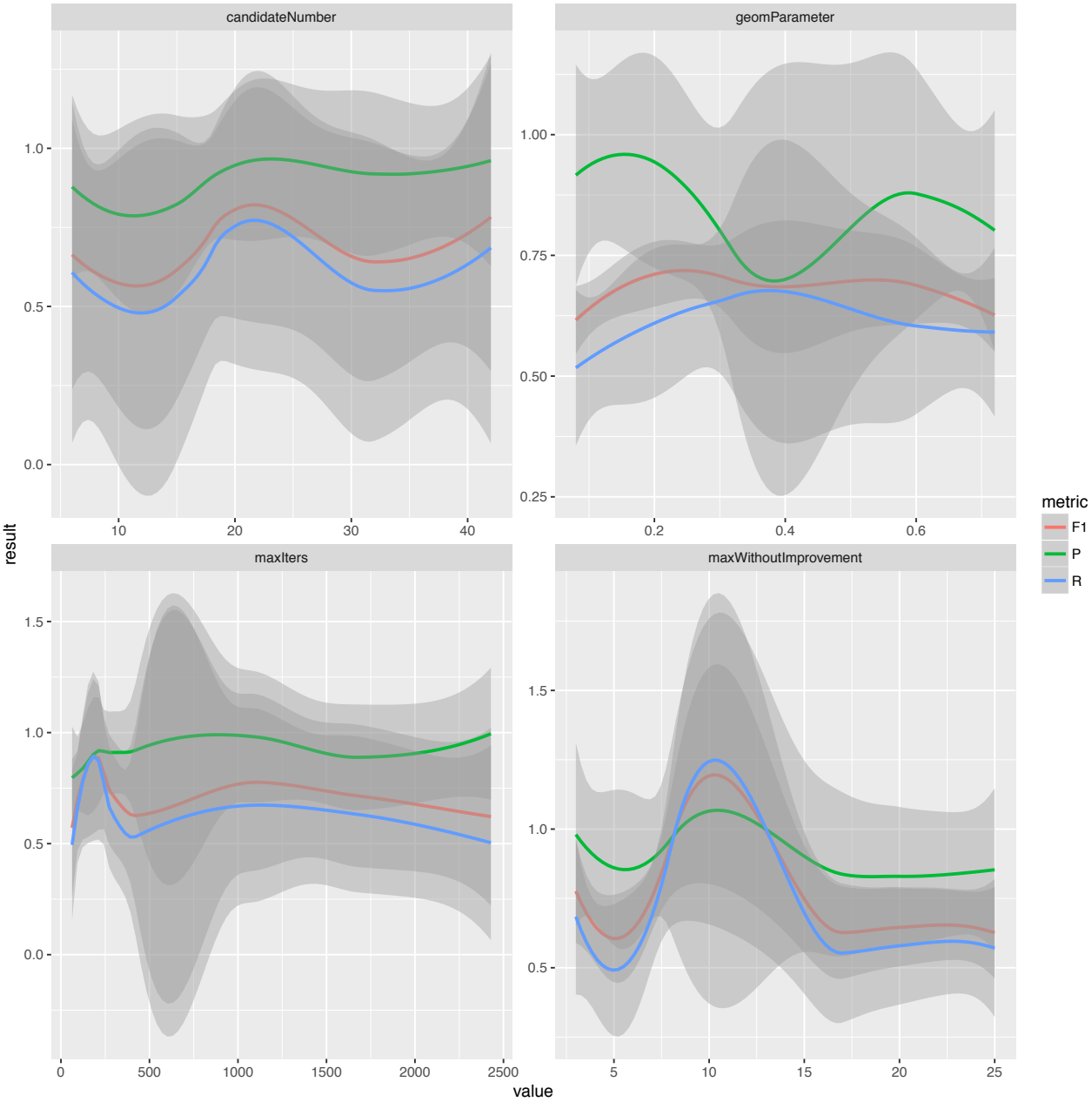

Figure 33: Global metaparameter results

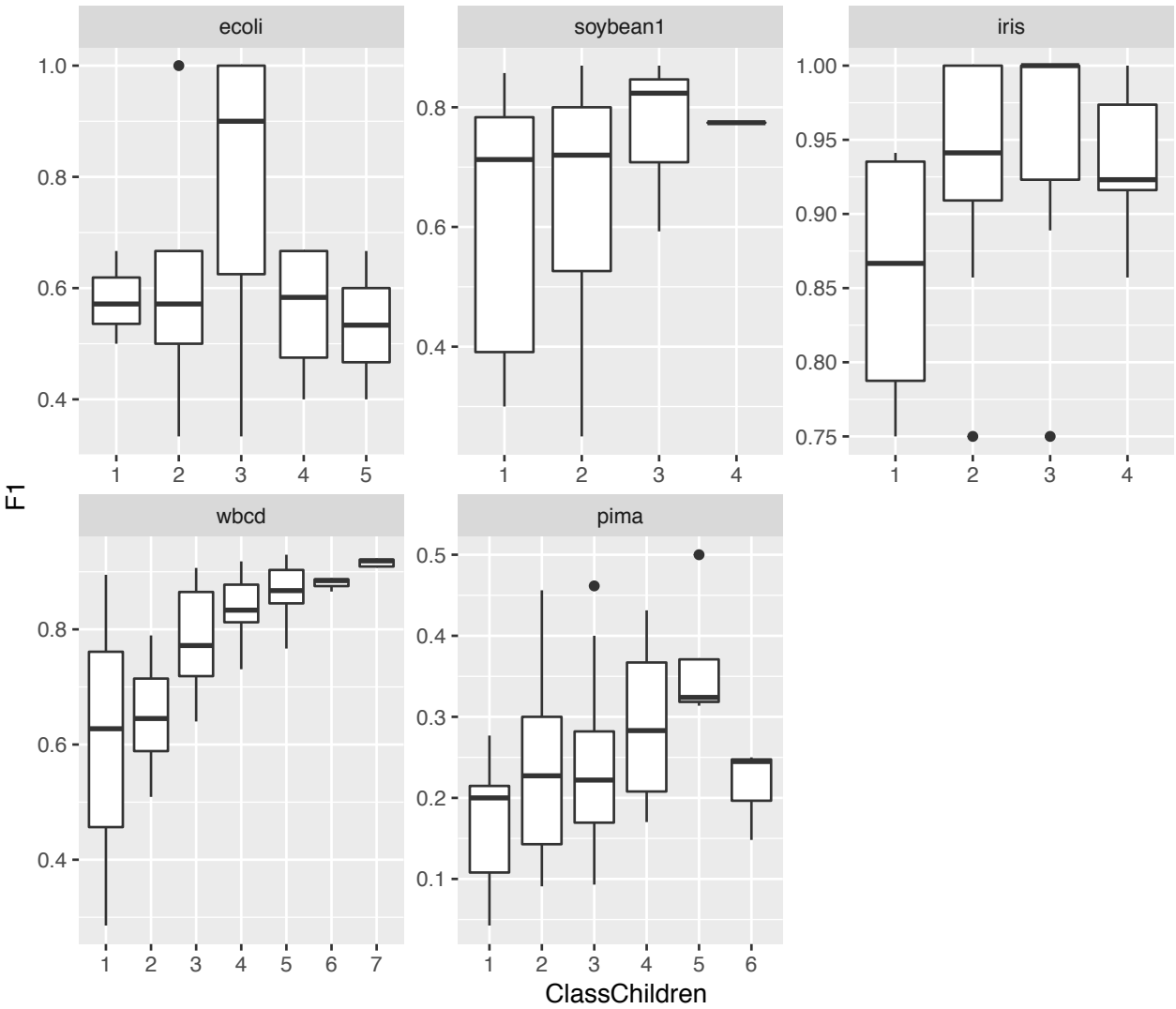

Figure 34: F1 metric for different numbers of class descendants grouped by dataset

neither on the MLE value nor on the total complexity on the graph, difficulting the task of designing an algorithm that optimizes this characteristic.

6.3 Impact of the regularization of Mutual Information in classification

One of the suggestions that came up in the analysis was to isolate the impact of the imbalance rate from the information metric used in the structure induction algorithm, in a similar way as to the one used to generate skew-insensitive decission trees in [34]. In order to test this assumption, a BC was trained for every datasets with four configuration:

- MLE score with Mutual Information Rectification
- MLE score without Mutual Information Rectification
- BIC score with Mutual Information Rectification
- BIC score without Mutual Information Rectification

Figure 37 shows a boxplot with the results for all the datasets, divided by score. Only the results with a positive value for F1 have been plotted. As the figure shows, the rectification greatly reduces the variance of

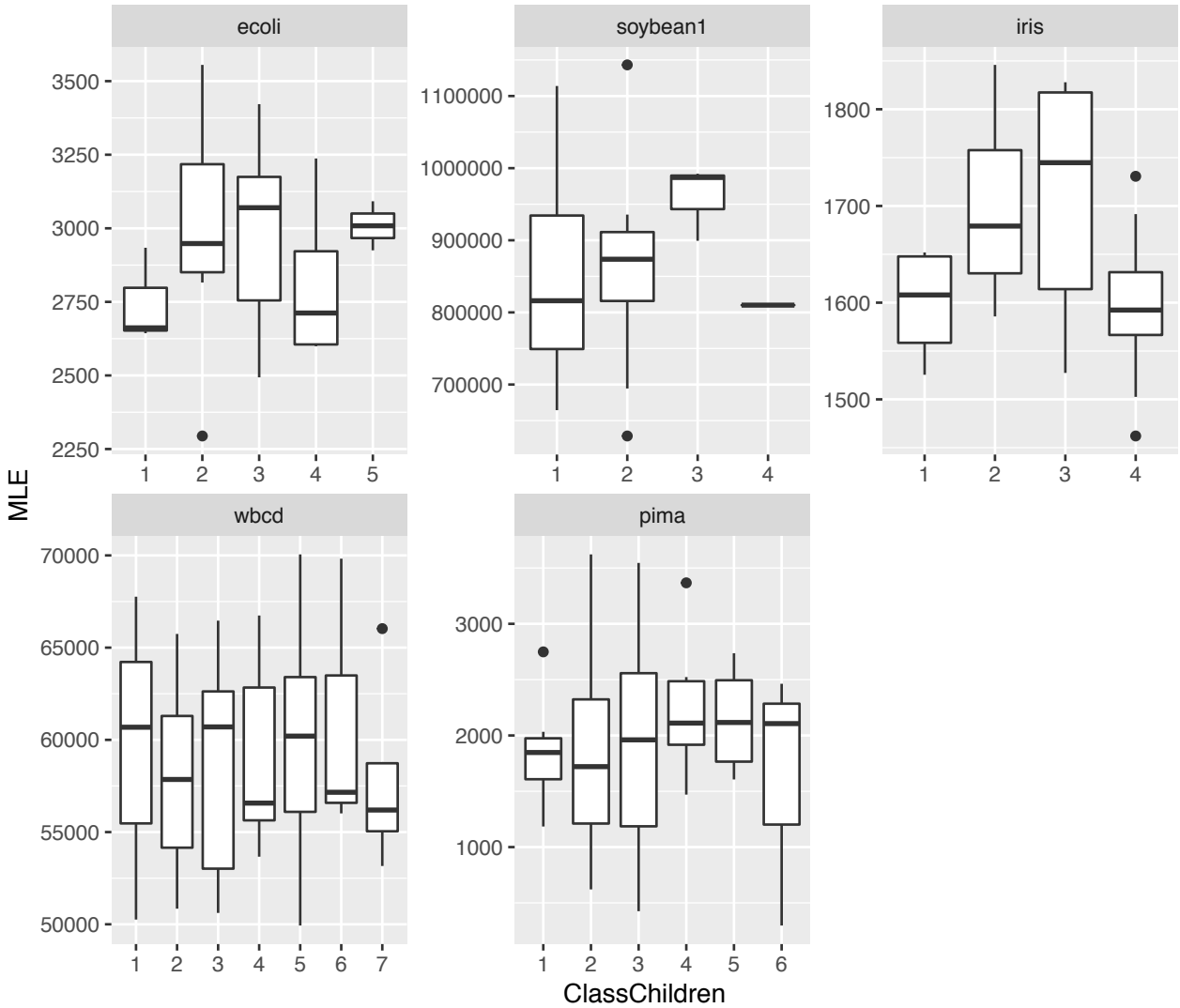

Figure 35: MLE against different numbers of class descendants grouped by dataset

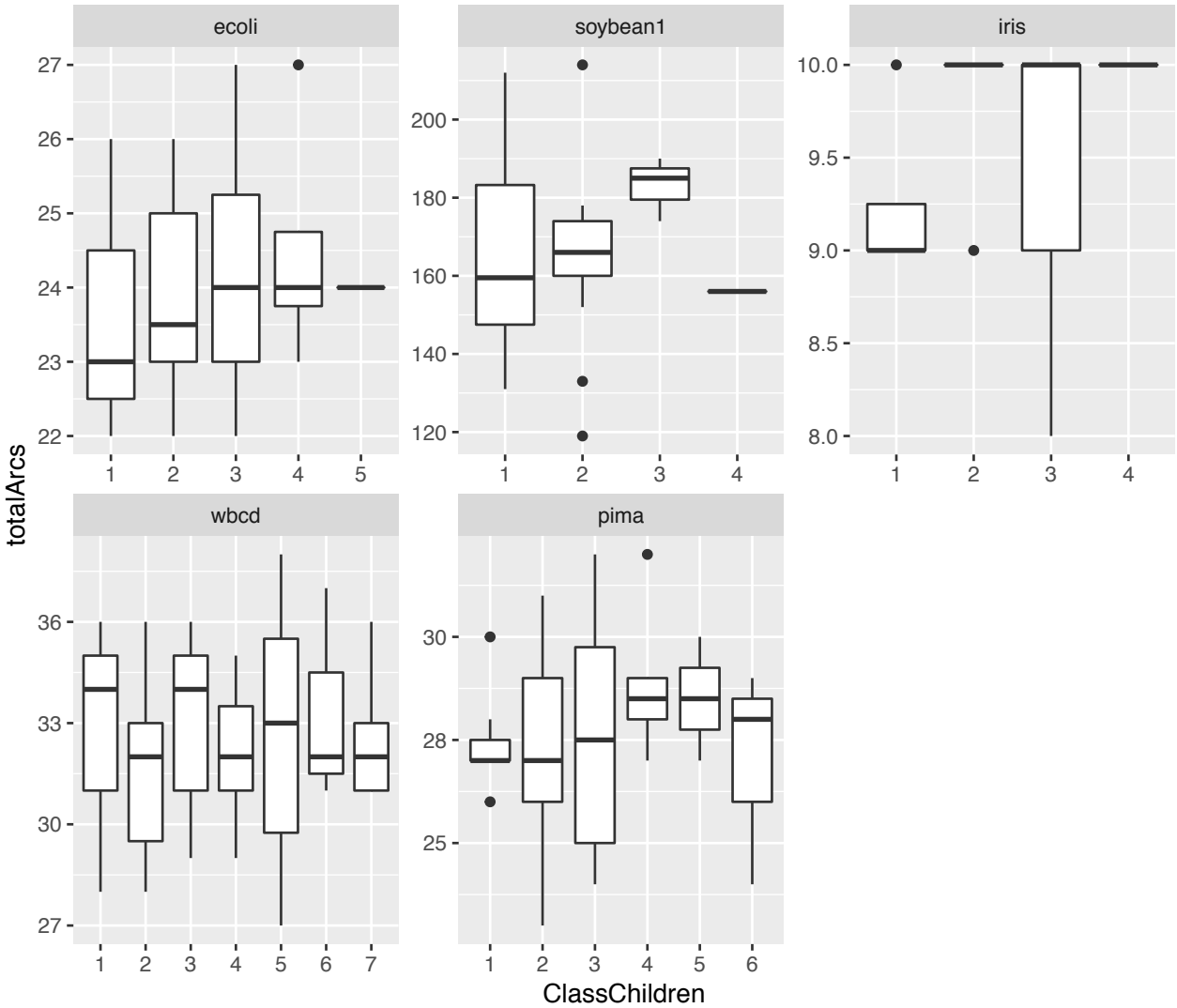

Figure 36: Total number of arcs against different numbers of class descendants grouped by dataset

the results, but its performance shows no clear correlation with the Imbalance Rate: it completely fails to mitigate the imbalance in the *ecoli* dataset, where there isn't any non-zero value for the rectified algorithm, while offering a better performance in the *soybean1* dataset, that is the second most unbalanced dataset.

6.4 Effect of the equivalent sample size and prior balance in parameter estimation

One of the techniques proposed to alleviate the effect of the imbalance was the use of the prior distributions to globally rebalance the frequencies of both classes. This should have a similar effect to those produced by the simple global resampling techniques. On the other hand, the prior distributions have a metaparameter of their own, the equivalent sample size, whose impact in the classification accuracy we would like to stablish. The equivalent sample size can be interpreted as the strength of our prior believes, compared with the observed data. In this sense, a low equivalent sample size should weaken the effect of a prior rebalance, while a high one should strengthen it.

In order to test the effect of both the equivalent sample size, σ and the rebalance constant γ, a set of experiments were conducted,

Figure 38 shows the values for Accuracy, F1, Precision and Recall for different values of the equivalent sample size alpha in all the datasets. Figure 39 shows the same graph for the values of the gamma parameter (that controls the balance between prior values). Most of the figures show a maximum value at particular values of the equivalent sample size and balance constant. Since this point depends on the dataset, a cross validation process should be executed for each of the datasets to optimze the values for both metaparameters, prior to the final classification.

Figure 40 shows a tiled heatmap of F1 values for different combinations of γ and α values.

6.5 Use of ensemble methods in imbalanced situations

In order to test the performance of the Bootstrapping solutions, two Bayesian Classifier Bootstrap algorithms where tested in all the datasets with two different weighting implementations:

- The first one (called *plainBootstrap*) used equal weights for all the models, using an average of the model results as the global result of the bootstraping.
- The other weighting algorithm (called *weightedBootstrap*) weighted each classifier with the probability of the test example in that model.

Both flavours were executed for a number of weak classifiers $N = 5$ and $N = 10$. The variants with ten classifiers are reported as a separate classifier in the results.

Figure 41 shows a boxplot diagram of the results for four different metrics of the different bootstrapping versions. Color bands were added as a visual aid for tight boxplots.

Tables 2 and 3 shows the result of a Wilcoxson Signed Rank test of the superiority of the *plainBootstrap* ensemble over the *weightedBootstrap* ensemble for each dataset. We can see that the former is vaguely superior to the latter in most of the tests, but with a low p value (probably due to the small number of samples) that give us low confidence in the result. The global value for all the datasets gives an estimate of 0.07 and a p-value of 0.07, showing also a slightly but low-confident superiority of the average of models over the probabilistic weighting. Increasing reaffirms this result and slightly increases the confidence.

6.6 Comparison with other algorithms

In order to evaluate the performance of the implementations against the true difficulty of the problem, all tests were also executed against two well-known classifiers:

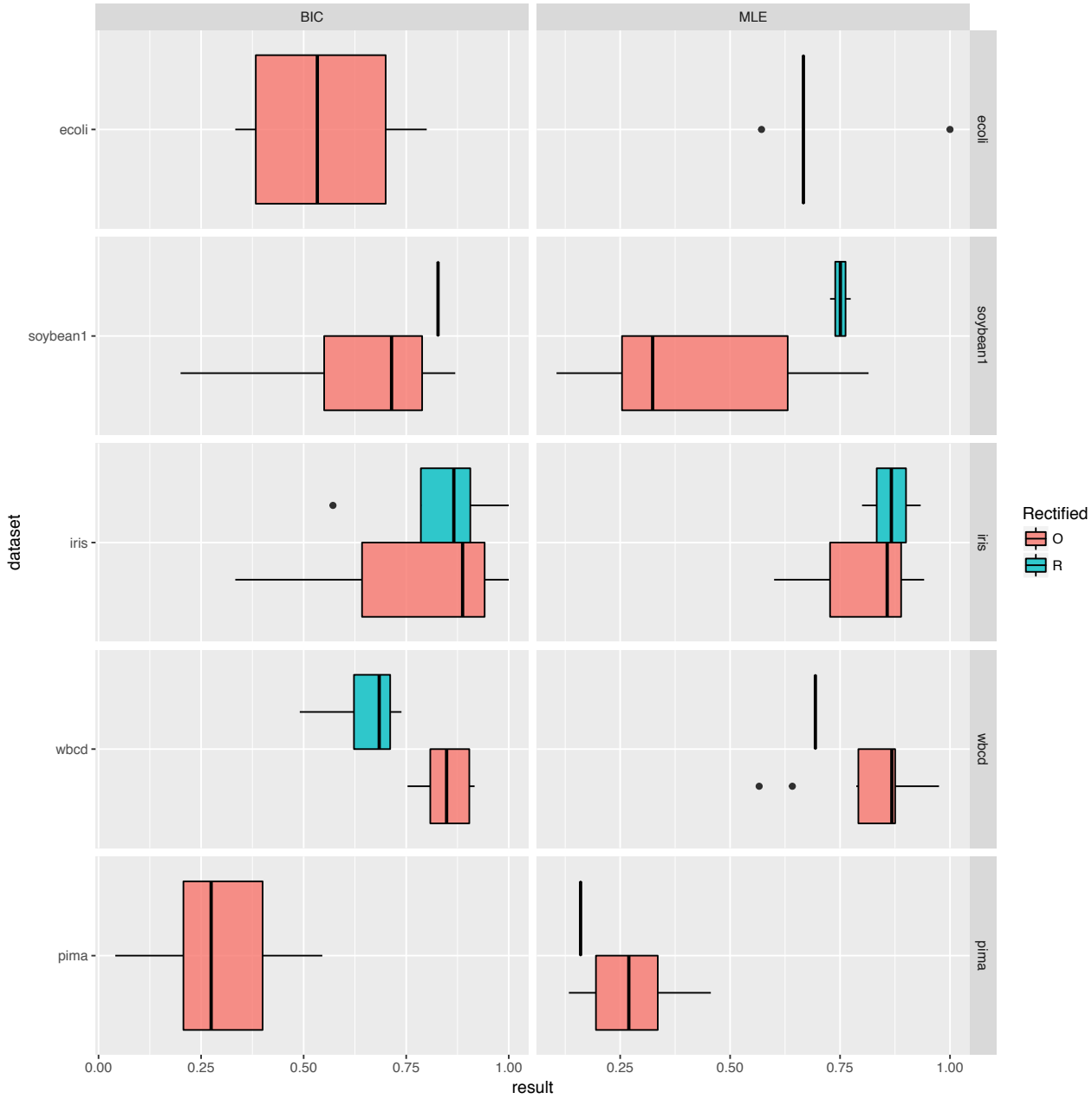

Figure 37: F1 value for each dataset with rectified and unrectified metrics

Table 2: Results for the Wilcoxon Signed Rank tests for bootstrapping versions results (N=5)

dataset	estimate	p
ecoli	0.00	0.21
soybean1	0.12	0.09
iris	0.08	0.10
wbcd	0.00	0.50
pima	-0.03	0.65
overall	0.07	0.17

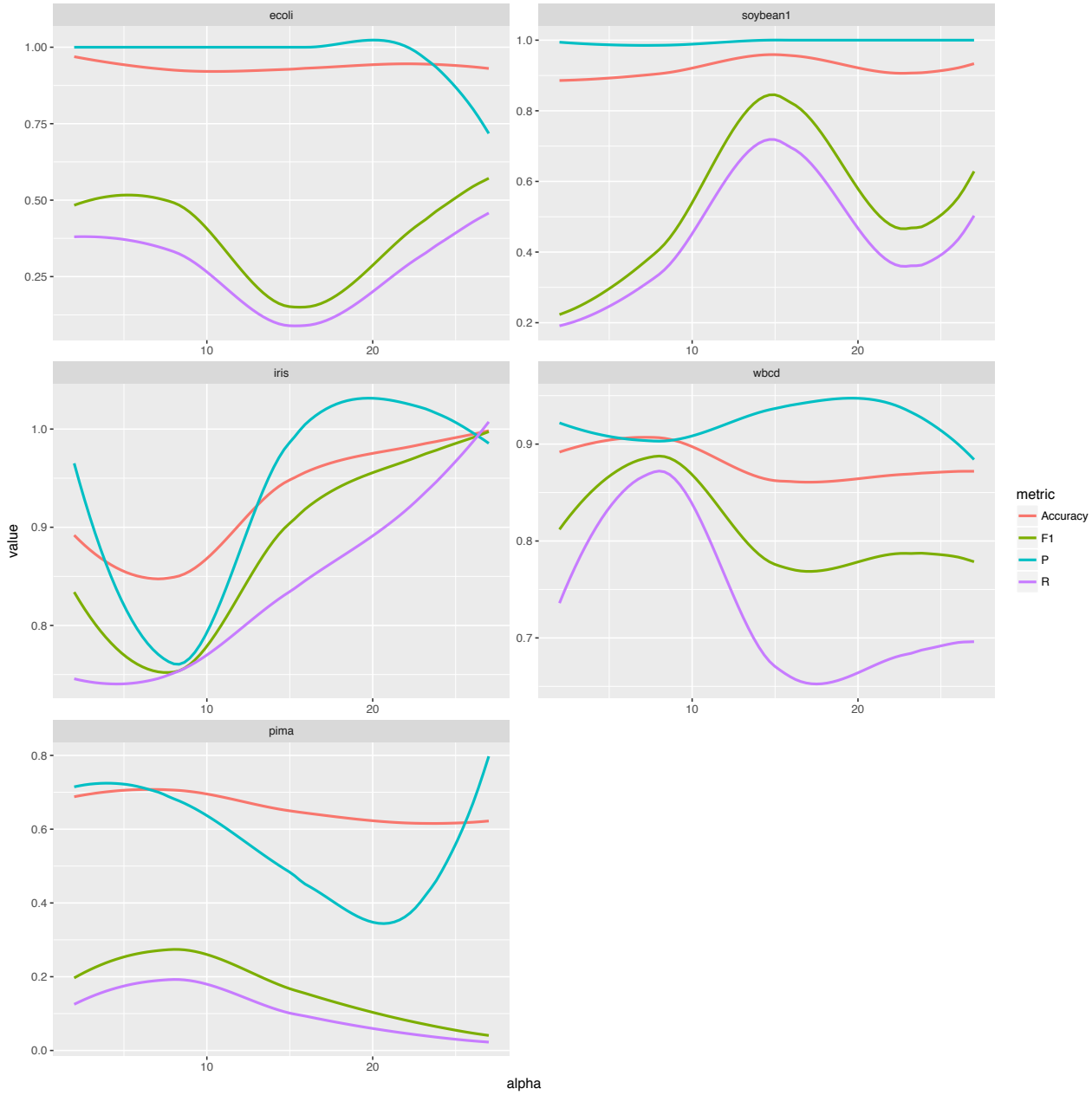

Figure 38: Metric values for different values of alpha (equivalente sample size)

Table 3: Results for the Wilcoxon Signed Rank tests for bootstrapping versions results (N=10)

dataset	estimate	p
ecoli	0.00	0.86
soybean1	0.46	0.02
iris	0.29	0.02
wbcd	0.05	0.09
pima	0.07	0.42
overall	0.13	0.07

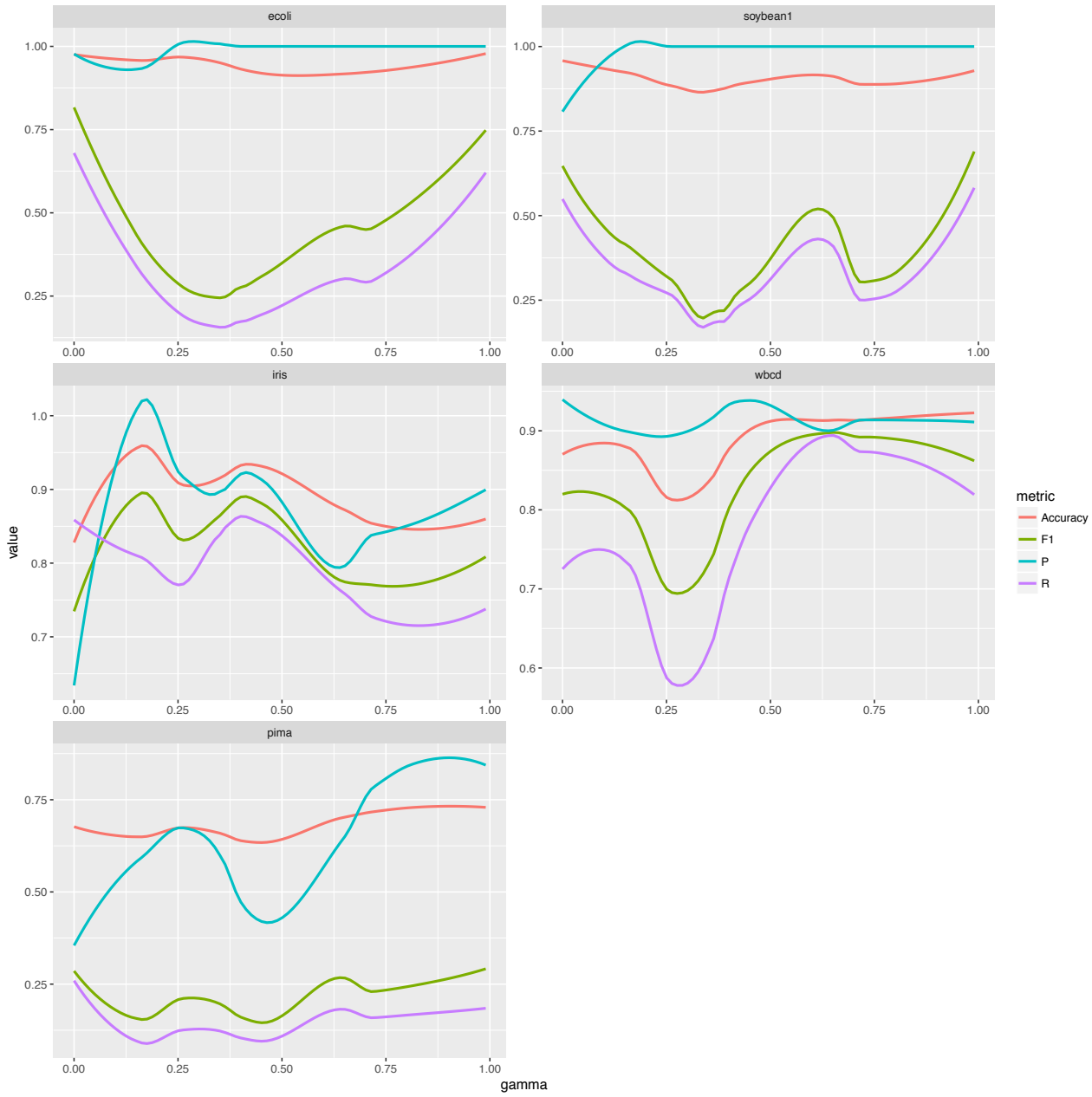

Figure 39: Metric values for different values of gamma (rebalance constant)

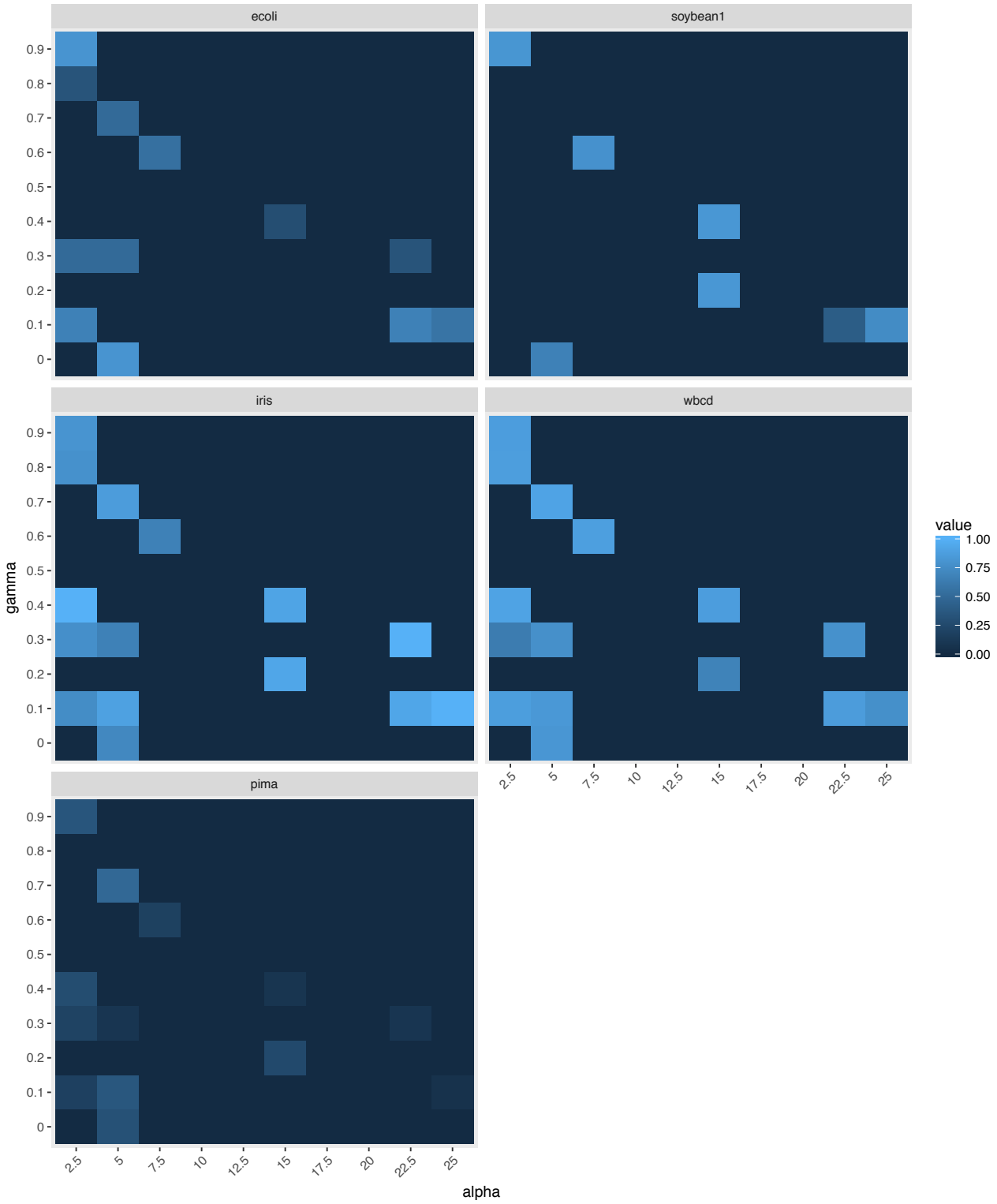

Figure 40: F1 values for different combinations of gamma and alpha

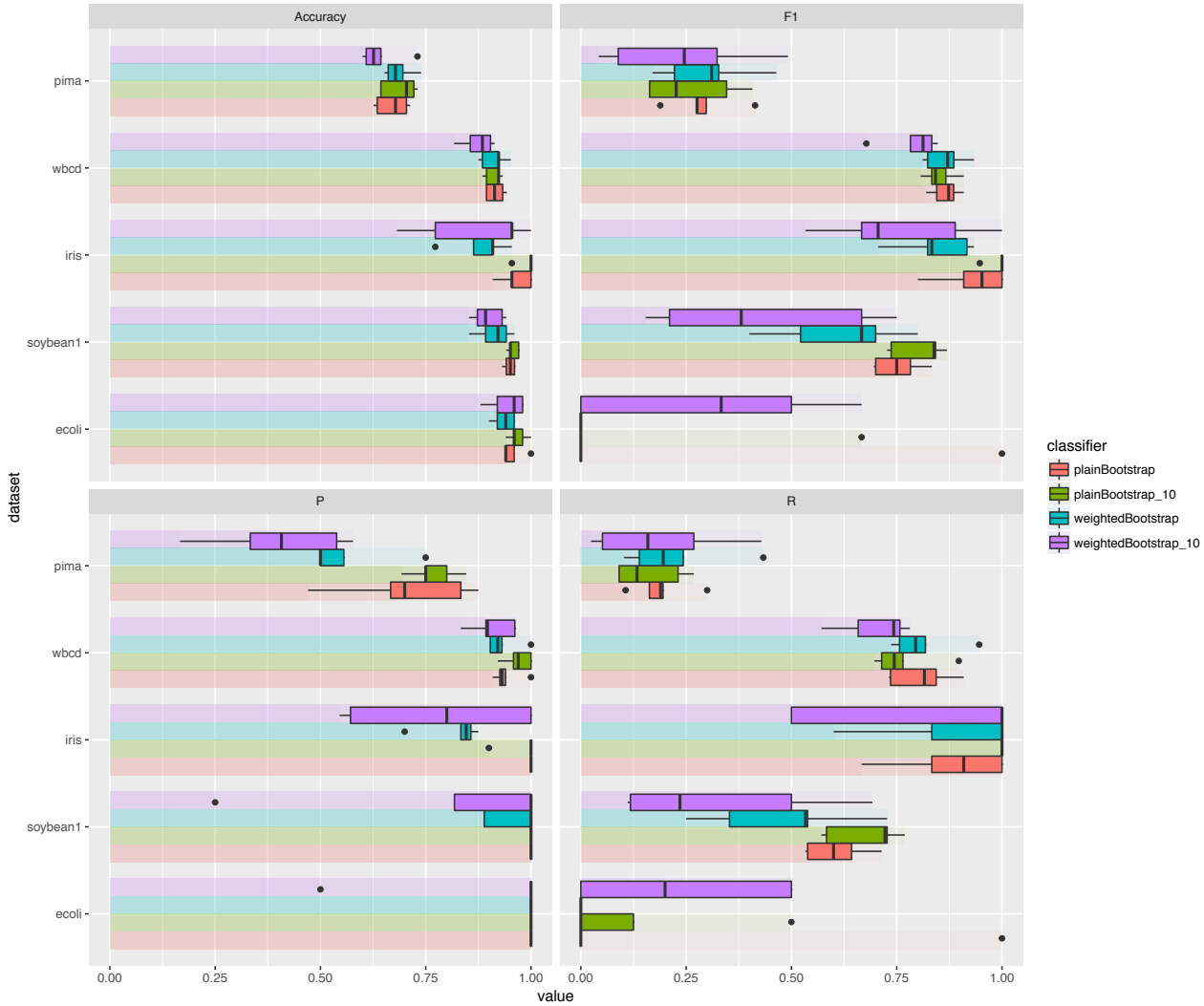

Figure 41: Comparison between bootstrap mechanisms

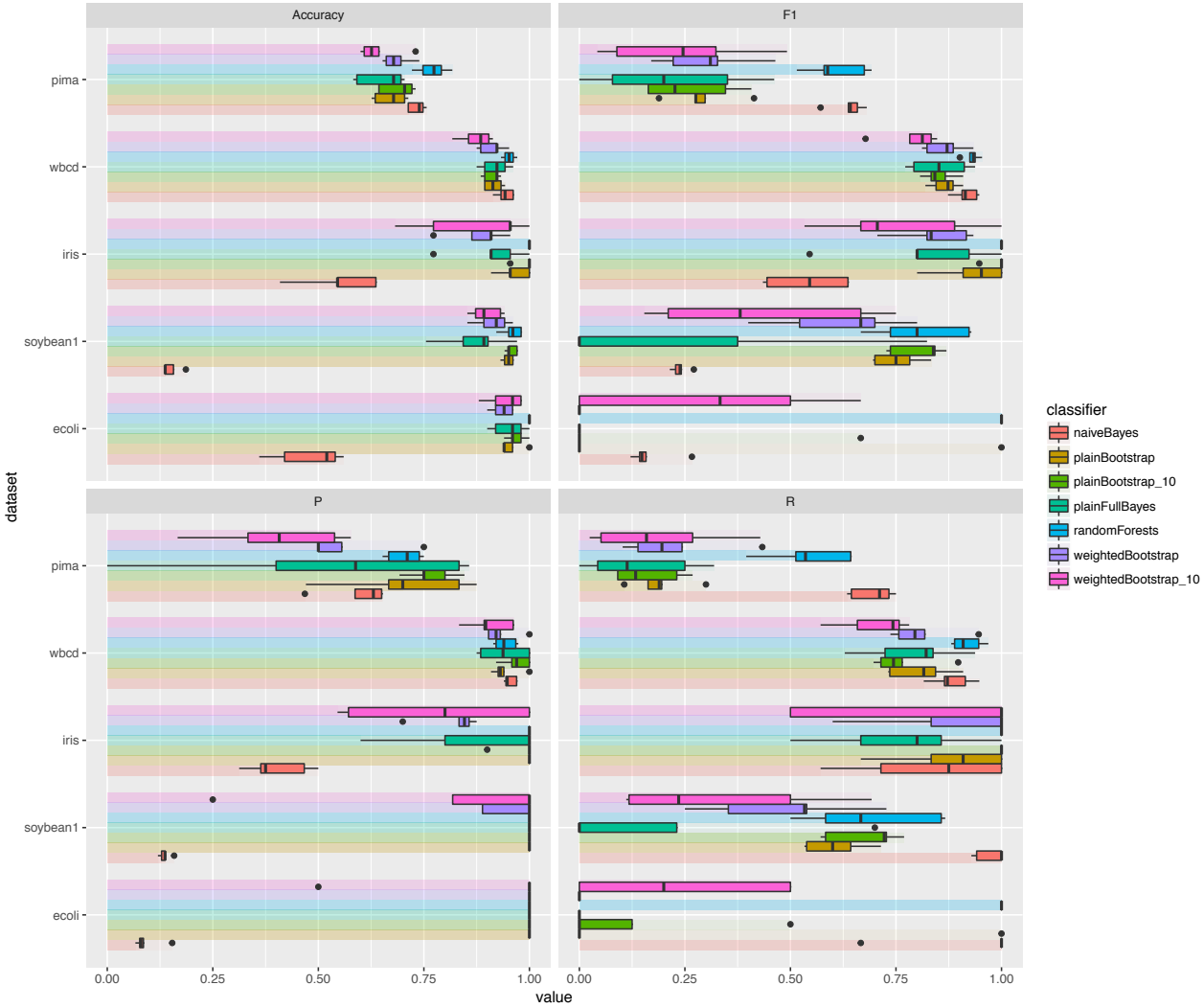

Figure 42: Classifier comparison

- A Näive Bayes classifier, in a version implemented ad-hoc for this project. The inclusion of Näive Bayes will aid in determining to what extent the structural assumptions made by this classifier (the independence of the features given the class) impact the real classification performance.
- A Random Forests classifier, taken from the implementation in R's Caret package. Both the algorithm and the implementation are known to have a good general performance, so this value will give us an indication on the expected maximum accuracy for the dataset.

Both classifiers where mainly included to stablish a reference level in the classification difficulty of the datasets. Since the goal of full Bayesian Classifiers is broader than that of other specialized classifiers (as the BC aims to capture an expression for the joint distribution the underlying distribution that generated the dataset), a lower classification accuracy is expected when compared with classification-oriented models.

Figure 42 shows the results for different metrics for all the classifiers tested in the experiments. Color bands were added as a visual aid for tight boxplots.

Tables 4 and 5 show, respectively, the results of the Wilcoxon Signed Rank test for the one to one comparison of classifiers. Some remarks are clear from the results:

- First of all, as expected, the Random Forests algorithm clearly outperforms all of the other classifiers in

Table 4: Estimates from the Wilcoxon Signed Rank tests for classifier comparison

	plainBootstrap	weightedBootstrap	plainFullBayes	naiveBayes	randomForests
plainBootstrap	0.00	0.07	0.14	0.14	-0.17
weightedBootstrap	-0.07	0.00	0.03	0.03	-0.27
plainFullBayes	-0.14	-0.03	0.00	-0.12	-0.49
naiveBayes	-0.14	-0.03	0.12	0.00	-0.36
randomForests	0.17	0.27	0.49	0.36	0.00

Table 5: P-value from the Wilcoxon Signed Rank tests for classifier comparison

	plainBootstrap	weightedBootstrap	plainFullBayes	naiveBayes	randomForests
plainBootstrap	0.00	0.17	0.06	0.07	1
weightedBootstrap	0.83	0.00	0.19	0.36	1
plainFullBayes	0.94	0.81	0.00	0.84	1
naiveBayes	0.93	0.65	0.16	0.00	1
randomForests	0.00	0.00	0.00	0.00	0

the comparison, with a very high confidence.

- The Plain Bootstrap Bayesian Classifier performs better than the rest of the algorithms, excluding the Random Forests, but the confidence levels are lower in this case. The addition of more experiments could enhance the confidence in this superiority.
- The simple Full Bayesian Classifier has the worst performance of the ranking, even outperformed by the simple Näive Bayes classifier. Looking at Figure 42 we can see that this problem may be caused by the extreme variability in performance it has shown during the tests. As we will analyze in the next section.

6.7 Variability and zero-valued F1 metrics

One of the biggest problems we have encountered during the experiments is the inherently variability in the structure of the induced classifiers. This variability affected the learning process in two distinct ways:

- It makes the performance of the classifier somewhat impredictible compared with other learning algorithms.
- For hard to learn datasets, it generates some null results, where the classifier have zero recall (and so also zero F1 measure).

Figure 43 shows the number of null experiments for each of the dataset (taken from the heuristic determination experiments). We can see how the easier Iris and WBCD datasets have no null entries, while the harder Soybean1 dataset has more null entries than positive ones. This condition compromises the utility of the classifier under real data, but this extreme results seem to be due to an error in the structure induction process, that should not be hard to spot.

In order to shed light on the problem, some additional experiments were conducted, using the same settings as per the heuristic process, in order to measure some characteristics of the null vs. not null solutions.

Figure 44 shows boxplots for the values of the different characteristics for all the datasets.

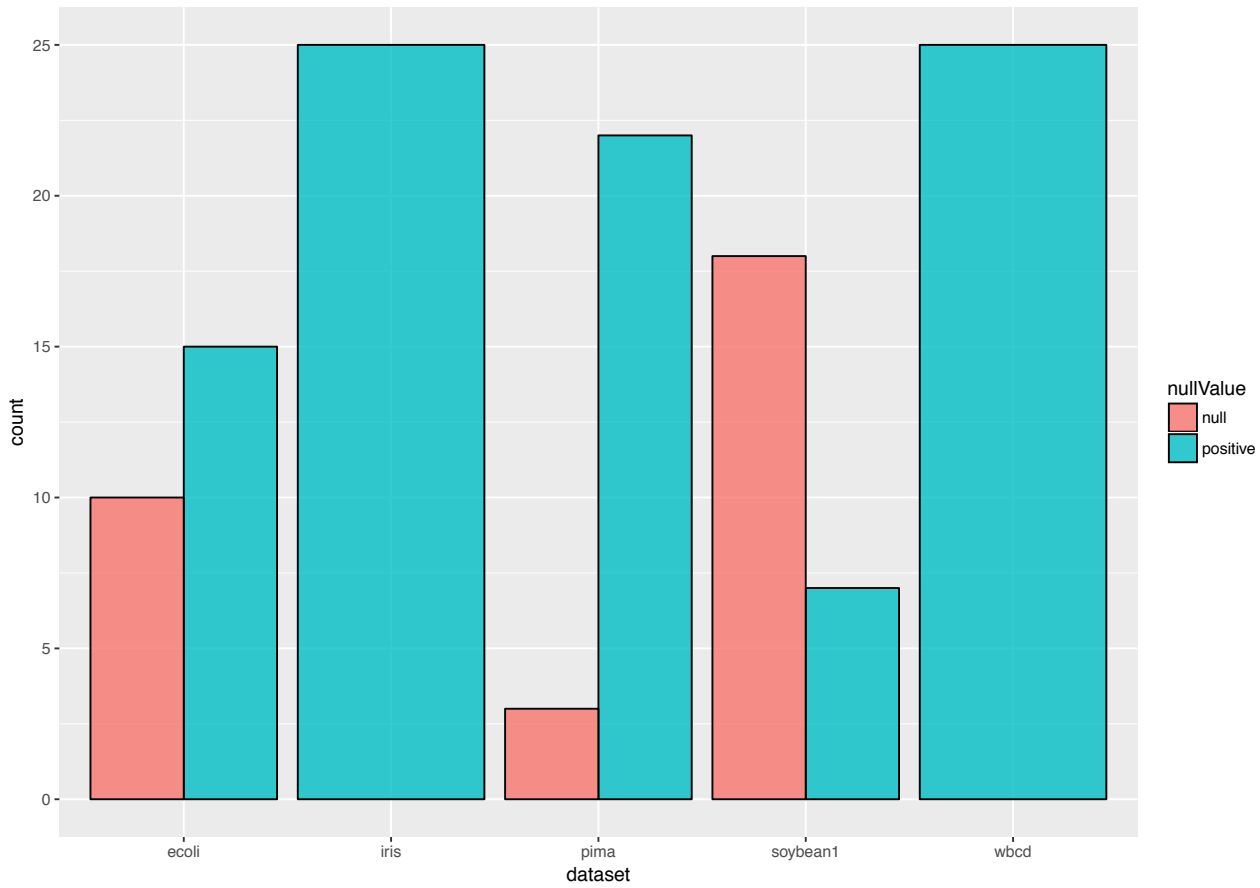

Figure 43: Number of zero values per dataset

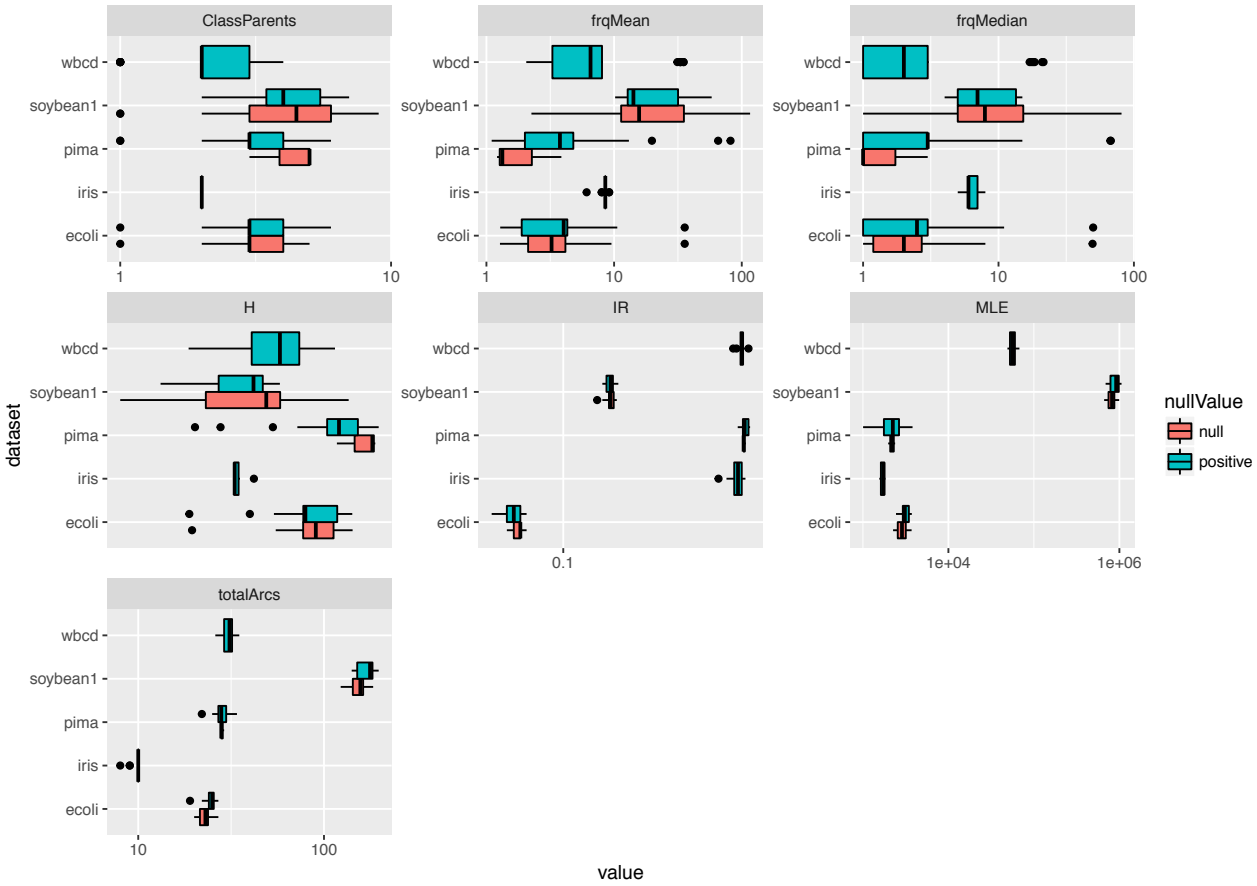

Figure 44: Characteristics analysis for null cases

7 Conclusions

This project aimed to analyze one important problem in the field of Machine Learning Classification: the class imbalance problem. The problem was described thoroughly, as were the metrics used to diagnose it. A long study of the State of the Art was conducted, showing the most relevant solutions found in the literature to tackle with the problem. Building on these ideas and on the theory behind Bayesian Classifiers, some algorithm modifications were proposed to try to improve the performance of the basic algorithms behind the BCs.

Three improvements were proposed and implemented in order to deal with the class imbalance problem: the rectification of the Mutual Information term in the structure induction; the rebalance of the parameter priors based on the imbalance; and the use of bagging ensembles based on the probability of occurrence of the test examples. Another proposal was defined in the analysis section (the use of fine-grained prior control), but was not implemented due to the lack of time. Experiments were conducted for all the implemented features, and its performance was compared with other well known classifiers, to stablish a reference of the classification difficulty.

For the case of the rectification, the results are ambiguous, as it had a positive effects in some datasets and a negative effects in other. It also made worse the null performance error described in the last section of the experiments for all datasets. In the case of the parameter priors, some evidence was found that both the equivalent sample size and the rebalance affected the performance of the classifiers, depending on the dataset, but due to small size of the experiments, its high variability generated a low confidence in the results. The last proposal, the use of a probabilistic-weight Bagging Ensemble, was outperformed by the most simple plain average of bootstrapped models (although it succeeded in improving the global performance of the plain Bayesian Classifier).

The tight project schedule imposed a reduction in the number of experiment replicas that could be executed as well as in the number of features of the implementation. As Figure 1 in the introduction reflects, a large part of the project was spent in State of the Art documentation and prototyping, while most of the implementation and experimentation was delayed until the last months of the project. This restriction forced to limit the scope of the implementation, and reduced the number of replicas for each experiment and the number of datasets originally selected (what led in turn to a reduction in the significance of the results).

A clear future line of development would be to extend the scope of the experiments, both in number of datasets and in replicas, in order to strengthen the significance of the results (leading to the confident acceptance or rejection of the proposed implementations). Another assertion that was suggested in the analysis (the theoretical simmilarity of some of the resampling mechanisms with prior rebalancing) remains untested, and could shed light on how to more appropriately use those priors to make a skew-insensitive BC.

The implementation of the classifier itself should also be improved. A small percentage of the induced classifiers showed abnormal zeroed metrics, a behavior that may be caused by an implementation bug or a characteristic in the induced model that we hope to be able to statically diagnose from the induced classifier (and so, to solve it in training time). It should also be easy to broaden the classifier scope by accepting multiclass problems and missing values (so taking some advantage of the BCs specific capabilities).

As a last line of improvement in the analysis side, it would also be important to perform another set of experiments focusing on testing with which degree of accuracy does the implemented classifier induce the appropriate dataset distribution, not only regarding classification, but also regarding the precision with which the probability distribution is captured

8 Appendix A. Classifier Implementation

In order to execute the designed experiments, and to allow all the desired parameters to be changed, a Bayesian Classifier was implemented from scratch, as an R library. To simplify the implementation, the BC was chosen to work only with discrete binary values and complete data (making the implementation multiclass and able to deal with missing data should be straightforward, due to the nature of the BC).

In order to facilitate the experiments, the implementation was made very configurable, in order to test all the options and metaparameters described in the experiments. Three basic algorithmic variants were implemented:

- **scoring**: score function to use in the training process: "bic" or "mle".
- **discretization**: method used for the discretization of numerical data: "equalFreq" or "topDown". The first discretization method was built into the classifier code: it divides the training data into N buckets of the same size, taking the top value of each bucket as the breaking point for the discretization. The "topDown" was chosen from the available algorithms in R's 'discretization' package.
- **heuristicSearch**: metaheuristic algorithm used to search for the optimal structure: 'rb' (Randomized Bias), or 'hc' (Hill Climbing).
- **rectifiedMI**: use of the recification of the Mutual Information: *true* or *false*.

All this options can be combined together in any way, giving rise to eight possible basic variations of the classifier. To simplify the experiments, a default value was selected for each options, so only the option being tested was changed for each experiment. The default values were: "bic", "equalFreq" and "rb".

Here is a list of the implemented metaparameters:

- α: equivalent sample size for the Dirichlet prior distributions.
- γ: balance level between the values that make up the equivalent sample size. Used in the experiments to balance the prior distribution against the imbalance rate.
- φ: this parameter controls the amount of rectification of the Mutual Information in case the *rectifiedMI* flag were active.
- C: this structure constant multiplies the regularization term in the BIC score function. It can be used to regulate the target complexity of the induced model (the greater the constant, the lower the complexity of the model).

Apart from the implementation of the Bayesian Classifier, a Bootstrap Aggragating module was created in order to create the ensambles to test. All the weak classifiers for the ensemble are instances of the plain Bayesian Classifier. Two options of classifier combination were implemented: one with a simple average of models and another one that weighted each classifier based on the normalized probability that it assigned to each of the test instances.

9 Appendix B. Datasets

9.1 Real-world datasets

In order to asess the influence of the imbalance level in the classification error, several reallife datasets will be used, among those most frequent in the literature. Most of them come from the UCI Machine Learning Repository [38]:

- Pima Indians Diabetes Data Set [39]
- Breast Cancer Wisconsin (Diagnostic) Data Set [40]
- Abalone Data Set [41]: this is not strictly speaking a classification problem, as it consists in guessing the age of the specimen given the other traits. We follow the approach finding in the literature of classifying the specimens as less than 19 years-old (negative class) or equal or greater (positive class).
- Ecoli Data Set [42]: this is a multiclass classificaiton problem. Since in this preliminar experiments we are dealing with binary classification problems, we will consider the problem of deciding whether the protein is localized in the outer membrane.
- Thyroid Data Set [43]: this dataset is actually broken into other smaller ones. We have selected the following ones: "sick", "allhypo" and "allrep"
- Yeast Data Set [44]: this is also a multiclass dataset; it will be transformed into a binary classification problem by selecting a subset of the classes as positive. Following the approach in [45] just the "POX" and "CYT" classes will be considered as positive classes.
- Iris Data Set [46]: one of the most common datasets in the classification literature, it describes specimens of Iris from three diferent species: Setosa, Versicolor and Virginica.
- Soy Bean Data Set [47]: another classical dataset regarding Soy Bean diseases.

For all the datasets that provided a partition in test and training data, that partition has been honored. For those who not, a random partition is created on loading, dividing the dataset in 85% of training data and 15% of test data.

Prior to its use in the experiments, all the datasets were cleaned, i.e.: missing values were removed, and types were adjusted to avoid format problems in the training step.

All the datasets were collected into an R Package for ease of management.

Due to time restrictions, most of the experiments were restricted to the following list of datasets: *ecoli*, *soybean1*, *iris*, *wbcd* and *pima*.

Table 6 shows the summary of the dataset information ordered by Imbalance Rate:

9.2 Synthetic datasets

All examples from the problem description section are generated using synthetic datasets, with the required characteristics. All the datasets consist of random instances drawn from two distributions, in a space formed by two numeric variables X and Y, and a target binary class that select the distribution. Each distribution is composed of a set of clusters of examples, normally distributed around a center point (with all the center points located in a rectangular grid around the origin).

The following subsections describe how each of the characteristics of the dataset are generated.

Table 6: Summary of real-life datasets information

dataset	examples	features	ir
abalone	4177	9	0.02
soybean3	683	36	0.03
allrep	3772	28	0.03
ecoli	336	8	0.06
sick	3772	28	0.06
soybean2	683	36	0.06
allhypo	3772	28	0.08
soybean1	683	36	0.13
yeast	1484	10	0.33
iris	150	5	0.33
wbcd	699	10	0.34
pima	768	9	0.35
ionosphere	351	34	0.64

9.2.1 Complexity

The complexity level of the dataset is determined by the number of normal distributions that compose each cluster. The number of generated clusters is 2^n with n the complexity number, with all the cluster centers lying in an evenly spaced grid around the origin.

9.2.2 Interclass imbalance

Interclass imbalance is changed by modifying the probability that a new example have of being drawn from a particular disjunct of the minority class. When this factor has the value 1, all the disjuncts have equal probability, and so, the examples tend to be evenly distributed among disjuncts. Higher values generate more skewed distributions, using the following expressions:

$$\tilde{P}(c) = 1.75 * (d-1)^c$$

$$P(c_i) = \frac{\tilde{P}(c_i)}{\sum_{x \in C} \tilde{P}(x)}$$

Where $C = 1, 2, ...N_c$ is each of the cluster identifiers.

9.2.3 Overlap between classes

Since all the synthetic datasets are composed of sets of normally distributed clusters, the degree of overlap between them will be governed by the standard deviation of those normal distributions. These deviations are calculated as:

$$\sigma = \frac{W_G}{6 - L_o}$$

Where W_G is the width of each square of the grid where the clusters are allocated (a parameter that depends on the complexity level) and L_o the overlap level (that vary from 1 to 5 in the experiments).

Bibliography

[1] A. Solanas *et al.*, "Smart health: A context-aware health paradigm within smart cities," *IEEE Communications Magazine*, vol. 52, pp. 74–81, IEEE, 2014.

[2] A. Solanas, F. Casino, E. Batista, and R. Rallo, "Trends and challenges in smart healthcare research: A journey from data to wisdom," *2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)*, pp. 1–6, IEEE, 2017.

[3] D. Koller and N. Friedman, "Learning," *Probabilistic Graphical Models*, pp. 695–1006, MIT Press, 2009.

[4] D. Koller, "Probabilistic graphical models specialization," 2009. [Online].
Available: https://www.coursera.org/specializations/probabilistic-graphical-models .

[5] J. Ortigosa-Hernández, I. Inza, and J. Lozano, "Towards competitive classifiers for unbalanced classification problems: A study on the performance scores," 2016.

[6] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, "RUSBoost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 40, pp. 185–197, IEEE, 2010.

[7] G. M. Weiss and F. Provost, "Learning when training data are costly: The effect of class distribution on tree induction," *Journal of Artificial Intelligence Research 19*, pp. 315–354, AAAI Press, 2003.

[8] T. Jo and N. Japkowicz, "Class imbalances versus small disjuncts," *SIGKDD Explorations Newsletter*, vol. 6, pp. 40–49, ACM, 2004.

[9] N. Japkowicz, "Class imbalance: Are we focusing on the right issue?" *Notes from the ICML Workshop on Learning from Imbalanced Data Sets*, 2003.

[10] N. Tomašev and D. Mladenić, "Class imbalance and the curse of minority hubs," *Knowledge-Based Systems*, vol. 53, pp. 157–172, Elsevier Science Publishers B. V., 2013.

[11] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, pp. 113–141, Elsevier, 2013.

[12] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, pp. 429–449, IOS Press, 2002.

[13] M. Denil and T. Trappenberg, "Overlap versus imbalance," *Advances in Artificial Intelligence*, pp. 220–231, Springer Berlin Heidelberg, 2010.

[14] M. Wasikowski and X. w. Chen, "Combating the small sample class imbalance problem using feature selection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1388–1400, IEEE, 2010.

[15] M. Khun, "A short introduction to the caret package," 2016. [Online].
Available: https://cran.r-project.org/web/packages/caret/vignettes/caret.html .

[16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, AI Access Foundation, 2002.

[17] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," *Advances in Intelligent Computing*, pp. 878–887, Springer Berlin Heidelberg, 2005.

[18] H. He and E. Garcia, "Learning from imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, pp. 1263–1284, IEEE, 2009.

[19] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," *In Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179–186, Morgan

Kaufmann, 1997.

[20] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD Explorations Newsletter*, vol. 6, pp. 20–29, ACM, 2004.

[21] G. Weiss, "Foundations of imbalance learning," *Imbalanced Learning Foundations, Algorithms, and Applications*, pp. 13–43, Wiley-IEEE Press, 2013.

[22] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, pp. 463–484, IEEE, 2012.

[23] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, pp. 3358–3378, Elsevier, 2007.

[24] X. Y. Liu, J. Wu, and Z. H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, pp. 539–550, IEEE, 2009.

[25] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, pp. 119–139, Academic Press, Inc., 1997.

[26] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 324–331, IEEE, 2009.

[27] K. M. Ting, "An instance-weighting method to induce cost-sensitive trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, pp. 659–665, IEEE, 2002.

[28] M. Maloof, "Learning when data sets are imbalanced and when costs are unequal and unknown," *Notes from the ICML Workshop on Learning from Imbalanced Data Sets*, 2003.

[29] P. Domingos, "MetaCost: A general method for making classifiers cost-sensitive," *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 155–164, ACM, 1999.

[30] C. Elkan, "The foundations of cost-sensitive learning," *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, pp. 973–978, Morgan Kaufmann Publishers Inc., 2001.

[31] F. Provost and P. Domingos, "Tree induction for probability-based ranking," *Machine Learning*, vol. 52, pp. 199–215, Springer, 2003.

[32] R. Vilalta and D. Oblinger, "A quantification of distance bias between evaluation metrics in classification," *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 1087–1094, Morgan Kaufmann Publishers Inc., 2000.

[33] P. A. Flach, "The geometry of roc space: Understanding machine learning metrics through roc isometrics," *in Proceedings of the Twentieth International Conference on Machine Learning*, pp. 194–201, AAAI Press, 2003.

[34] D. A. Cieslak and N. V. Chawla, "Learning decision trees for unbalanced data," *Machine Learning and Knowledge Discovery in Databases*, pp. 241–256, Springer Berlin Heidelberg, 2008.

[35] A. A. Juana, I. Pascuala, D. Guimaransb, and B. Barriosa, "Combining biased randomization with iterated local search for solving the multidepot vehicle routing problem," *International Transactions in Operational Research*, pp. 647–667, Wiley, 2014.

[36] M. J. Flores and J. A. Gamez, "Impact on bayesian networks classifiers when learning from imbalanced datasets," *International Conference on Agents and Artificial Intelligence*, vol. 2, pp. 382–389, SciTePress,

2015.

[37] D. Heckerman, "A tutorial on learning with bayesian networks," *Innovations in Bayesian Networks: Theory and Applications*, pp. 33–82, Springer Berlin Heidelberg, 2008.

[38] "UCI machine learning repository." [Online].
Available: https://archive.ics.uci.edu/ml/index.php .

[39] V. Sigillito, "Pima indians diabetes data set," 1990. [Online].
Available: https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes .

[40] Wolberg, Street, and Mangasarian, "Breast cancer wisconsin (diagnostic) data set," 1995. [Online].
Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) .

[41] Nash, Sellers, and Talbot, "Abalone data set," 1994. [Online].
Available: https://archive.ics.uci.edu/ml/datasets/abalone .

[42] K. Nakai, "Ecoli data set," 1996. [Online].
Available: https://archive.ics.uci.edu/ml/datasets/ecoli .

[43] R. Quinlan, "Thyroid disease data set," 1986. [Online].
Available: http://archive.ics.uci.edu/ml/datasets/thyroid+disease .

[44] K. Nakai, "Yeast data set," 1996. [Online].
Available: https://archive.ics.uci.edu/ml/datasets/Yeast .

[45] L. Nanni, C. Fantozzi, and N. Lazzarini, "Coupling different methods for overcoming the class imbalance problem," *Neurocomputing*, vol. 158, pp. 48–61, Elsevier, 2015.

[46] R. Fisher, "Iris data set," 1936. [Online].
Available: https://archive.ics.uci.edu/ml/datasets/iris .

[47] Michalski and Chilausky, "Soybean data set," 1980. [Online].
Available: https://archive.ics.uci.edu/ml/datasets/Soybean+(Large) .