UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

José da Silva

Modelo para teses e dissertações em LATEX utilizando o Pacote USPSC para o IFSC

São Carlos

2021

José da Silva

Modelo para teses e dissertações em LaTeX utilizando o Pacote USPSC para o IFSC

Thesis presented to the Graduate Program in Physics at the Instituto de Física de São Carlos da Universidade de São Paulo, to obtain the degree of Doctor in Science.

Concentration area: Applied Physics

Advisor: Profa. Dra. Elisa Gonçalves Rodrigues

Versão original

São Carlos

2021

É possível elaborar a ficha catalográfica em LaTeX ou incluir a fornecida pela Biblioteca. Para tanto observe a programação contida nos arquivos USPSC-modelo.tex e fichacatalografica.tex e/ou gere o arquivo fichacatalografica.pdf.

A biblioteca da sua Unidade lhe fornecerá um arquivo PDF com a ficha catalográfica definitiva, que deverá ser salvo como fichacatalografica.pdf no diretório do seu projeto.

ERRATA

A errata é um elemento opcional, que consiste de uma lista de erros da obra, precedidos pelas folhas e linhas onde eles ocorrem e seguidos pelas correções correspondentes. Deve ser inserida logo após a folha de rosto e conter a referência do trabalho para facilitar sua identificação, conforme a ABNT NBR 14724 (**?**).

Modelo de Errata:

SILVA, J. **Modelo para teses e dissertações em LaTeX utilizando o Pacote USPSC para o IFSC**. 2021. **??**p. Tese (Doutorado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2021.

ERRATA

Folha	**Linha**	**Onde se lê**	**Leia-se**
1	10	auto-conclavo	autoconclavo

Este trabalho é dedicado aos alunos da USP, como uma contribuição das Bibliotecas do Campus USP de São Carlos para o desenvolvimento e disseminação da pesquisa científica da Universidade.

ACKNOWLEDGEMENTS

Primeira frase do agradecimento ....

Segunda frase ....

Outras frases ....

Última frase ....

*"O estudo, a busca da verdade e da beleza são domínios
em que nos é consentido sermos crianças por toda a vida."*
Albert Einstein

ABSTRACT

SILVA, J. **Model for thesis and dissertations in LaTeX using the USPSC Package to the IFSC**. 2021. **??**p. Thesis (Doctor in Science) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2021.

This is the english abstract.

Keywords: LaTeX. USPSC class. Thesis. Dissertation. Conclusion course paper.

RESUMO

SILVA, J. **Modelo para teses e dissertações em LaTeX utilizando o Pacote USPSC para o IFSC**. 2021. **??**p. Tese (Doutorado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2021.

O resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (. . .) Salientamos que algumas Unidades exigem o titulo dos trabalhos acadêmicos em inglês, tornando necessário a inclusão das referências nos resumos e abstracts, o que foi adotado no **Modelo para TCC em LaTeX utilizando a classe USPSC** e no **Modelo para teses e dissertações em LaTeX utilizando a classe USPSC**. As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto (**?**).

Palavras-chave: LaTeX. Classe USPSC. Tese. Dissertação. Trabalho de conclusão de curso (TCC).

CONTENTS

1 INTRODUCTION

Bipartite interaction data is a common representation for a multitude of phenomena. It consists of two separate groups of instances often representing two classes of distinct nature of objects. Each object from one group may interact with any of the objects from the other, so that each possible pair of objects from distinct groups holds a set of attributes describing their interaction. Examples of such grouped instances are drugs and proteins, microRNAs and messenger RNAs or even videos and users on a media streaming platform. Thus, bipartite interactions also naturally encompasses the data format targeted by the broadly-known recommender systems ().

The interaction attributes may be of any dimensionality, and may as well be unknown for some (often many) instance pairs. When binary interactions are considered (pairs either do or do not interact in any specified way) we frequently find ourselves in a Positive-Unlabeled (PU) scenario (), where we can only confidently measure the presence of a given phenomena, not its absence, and hence, the instance pairs' interactions can only be said to be positive (actually happening) or unknown.

Furthermore, as the number of interactions grows with the product of the numbers of interacting instances in each bipartite group, taking all possible interactions into consideration may become unfeasible for larger datasets using standard machine learning algorithms. As a result, many workaround techniques are usually employed to generate negative interaction data, such as considering a random subset of unlabeled data as negative(**?**, **?**, **?**), selecting the most reliably-non interacting pairs (which depends on estimating the interaction likelihood with semi-supervised methods such as self-learning)() or even artificially creating new dataset instances when very specific factors are known to be needed for an interaction to occur (namely the chemical-structural characteristics of an enzyme's active site) ().

Despite even using sophisticated deep learning algorithms, these approaches thus fail to take all possible drug-target pairs into consideration.

Predictive Bi-Clustering Trees (PBCT) were proposed in 2018 by (**?**) to address some of these issues, introducing a new method for growing decision tree-based models from bipartite interaction data. With this method and further optimizations, millions of interactions can be considered in reasonable computation time.

Decision Trees work by recursively partitioning the dataset in chunks with progressively similar labels(**?**, **?**). They do so by consecutively searching for decision rules in each partition that would split the partition in two. For example, a specific numeric characteristic of our instances being less or greater than a threshold value, or if an instance has one of a specific set of values of a categorical variable. For this study, we mainly focus on numerical instance features, so that each tree node represents a binary split designated by an instance attribute an a threshold value.

The main idea behind biclustering trees was to separately search for a split attribute and value on each of the two instance groups, considering all possible thresholds among row instances first (e.g. proteins), and only then processing the column instances attributes (e.g. target drugs features).

In 2020, the authors expanded on this concept, building ExtraTrees ensembles of PBCTs (**?**) and reporting solid boosts on prediction performance. The authors, however, did not explore other forms of tree ensembles, including the so popular Random Forests proposed by (**?**), despite the latter being oftentimes regarded one of the best tree ensemble techniques(**?**, **?**, **?**, **?**). Hence, in this study we demonstrate how DTI prediction improvements can be achieved with the use of Random Forests of Predictive Bi-Clustering Trees, that we name Biclustering Random Forests, and provide an optimized implementation based on scikit-learn (**?**), one the most standard libraries for machine learning applications using the Python(**?**) programming language.

1.1 Introduction

The effect a drug molecule has on our organism is tightly associated with the specific microscopic structures it physically or chemically interacts with. The regulatory roles of microRNAs are often mediated by their binding to a defined set of messenger RNAs. The activity of a transcription factor is determined by which genes it is capable of regulating. The success of a recommendation system depends on its ability to predict a user's preference for a set of items in a catalog. All those scenarios share the same underlying structure: there are two distinct domains of objects that interact with one another, forming a heterogeneous *bipartite network*. As such, the role of a machine learning model in this setting is to receive a feature vector of the heterogeneous pair, often composed of independent representations of each interacting entity, and characterize their interaction through, namely, a binary label, affinity score or predicted user-item rating.

Frequently, great importance lies in computationally describing such relationships. For instance, are known to be spent in drug development every year. A large portion of that cost derives from the large *in vitro* screening experiments, extensive clinical trials and . Therefore, being able to predict beforehand the interactions in a large dataset of drugs and protein targets holds a remarkable potential to accelerate drug development and reduce its costs, enabling the concentration of financial resources and human expertise to a set of most promising candidates. On the other hand, being able to characterize on a large scale the relationships between microRNAs and genes or other biomolecular entities can greatly aid the understanding of regulatory mechanisms of gene expression, a valuable step towards the discovery and development of new therapies.

Despite their importance, interaction prediction problems pose unique challenges to machine learning approaches:

1. **Heterogeneous data:** The input data is often composed of two very dissimilar types of objects, each with its own set of features. As a result, estimators may be required to simultaneously deal with descriptors of very different natures.

2. **High dimensionality:** Due to the intrinsic combinatory nature of this type of problem, the interactions to be processed are usually very numerous, even if the number of training instances in each domain is not that expressive. Specifically, for each new object introduced to the training set, a new interaction could be considered for each training sample in the other domain, rendering the number of interactions to grow quadratically with the number of entities of each type.

3. **Knowledge sparsity:** As a consequence of the previous point, no amount of research efforts in characterizing new interactions can keep up with the rate at which possible relationships appear, resulting in a fundamental sparsity of confidently verified interactions.

4. **Lack of negative validation:** In many cases, the absence of a relationship between two objects is much harder to validate than its presence. For instance, negative results in biochemical essays asserting molecular interactions can often be caused by a plethora of external experimental factors other than the actual lack of interaction under optimal conditions, unlike positive results that usually imply a successful assay (). Furthermore, since positive results are often much more informative, negative results in such interaction characterization settings are naturally less valued in the scientific community, and therefore less likely to be reported. These factors result in a general lack of high-quality negative data in interaction datasets, requiring special considerations from machine learning pipelines that are often overlooked in the literature.

These issues highlight intrinsic differences between interaction prediction problems and more general supervised tasks, stemming from the fundamental way interaction problems are defined as predicting combinations (numerous by nature). Therefore, acknowledging such defining aspects and limitations and incorporating the bipartite nature of the problem from the ground up in the algorithm design process can be a crucial step in developing successful and scalable machine learning models for interaction prediction.

If, in a given bipartite interaction problem, the feature vector characterizing a pair of possibly interacting entities is a simple combination of independent representations of each entity, we can see the training set as two separate design matrices rather than the usual single one, disregarding their combination as a data preprocessing step and leaving the treatment of both in conjunction as a matter of estimator design. This view defines a different machine learning paradigm, in which what we call "instances" or "samples" are not the dyads themselves, but rather each individual interacting object. While *link prediction* and *dyadic prediction* are common terms in the literature referring to similar concepts, they are not specific to the presence of two distinct groups of objects nor make clear the presence of describing attributes for both of them (often called *side-features* or *side-information* in the context of recommendation systems). While *interaction prediction* seems a closer alternative, usually assuming side-features, it is also not concerned with the heterogeneity of the network. As an abuse of terminology, we thus refer to this "heterogeneous bipartite interaction prediction" paradigm as *bipartite learning*.

Interestingly, on the highest level, the variety of general frameworks and strategies specifically built under the bipartite paradigm is not staggering. The most common approach is to work with simple combinations of the feature vectors describing the components of each pair (by concatenating them, for example), effectively resetting the problem around the dyad as the primary atomic entity to be labeled. Transformed into a more traditional format, usual machine learning algorithms can then be applied to the problem. This approach, referred to as *global single output* (GSO) by (), is particularly impacted by the aforementioned scalability issues resulting from the large number of possible interactions, which leads many authors towards undersampling negative interactions, likely at the expense of increasing false positives. Furthermore, inadvertently sampling out test or validation sets directly from the GSO-transformed data results in feature vectors for individual instances to be shared with the training set, even if dyads themselves are disjoint, raising considerations about the resulting estimates of model generalization pahikalla2015.

Other strategies, termed *local multi-output* (LMO) involve building separate traditional models for each bipartite domain, considering instances in the unselected domain as featureless outputs to be modeled. While faster to train in general, the application of this approach in online settings is severely hampered by the need to train secondary models for each previously unseen instance.

Some interesting proposals modify estimators in a more fundamental way, diverging from the aforementioned strategies by not depending on organizing the bipartite dataset to be processed by a generic traditional algorithm. () demonstrates that a linear regression model can be built as if a dyad-dyad kernel matrix was used (which would be prohibitively large), by instead separately eigendecomposing the two intra-domain kernel matrices. However, while remarkably efficient, linear relationships frequently are too simplistic to capture the complexity of the underlying interactions, displaying limited predictive power. Under the name of *Predictive Bi-Clustering Trees* (PBCT) (**?**), proposes decision tree-based models to directly tackle bipartite datasets, by localizing to each domain not entire models, as in LMO, but rather the split search procedure occurring at each tree node. Nevertheless, no improvement in training complexity is observed relative to GSO-adapted decision trees.

Matrix factorization methods are an outcast in this general taxonomy of bipartite algorithms. Common in the context of recommendation systems, they are native to link prediction settings, being already successfully applied to bipartite problems. This class of algorithms is based on determining latent feature matrices for each domain, in a way that their matrix product approximates the adjacency matrix formed from the training labels. While originally agnostic to side-information and thus incapable of labeling new instances (what is sometimes referred to as the *cold start* problem), ingenious adaptations have been proposed liu to circumvent this issue by encouraging intra-domain pairwise distances to be a transferable property from the original to the latent feature space .

While deep learning strategies have been gaining popularity in the last years for problems such as the prediction of drug-target affinities or microRNA-gene interactions , they often rely on the same GSO transformation we mentioned before, suffering from the scalability issues we mentioned and often not reporting predictive performance for new instances, only new combinations. Furthermore, the particular lack of interpretability of these models is a long-discussed matter in the scientific literature , and more transparent reasoning supporting the obtained predictions could be an invaluable asset in the advancement of human knowledge, potentially uncovering new insights about underlying molecular dynamics or genetic regulatory mechanisms of various interaction phenomena.

On the other hand, decision tree-based methods are widely known for their transparency, flexibility and straightforward usage (), having a relatively small number of hyperparameters and being successfully applied to a wide range of learning problems. Hence, in this study, we tackle the presented challenges by proposing improvements to the concept of bipartite trees developed by (**?**). We introduce below.

- **Faster training:** By employing a global impurity metric and slightly modifying the training procedure of PBCTs, we are able to achieve a $\log n$ speedup in asymptotic training time while generating models identical to GSO-adapted trees built on all possible dyads in the training set. Theoretical complexity analysis and empirical results on artificial datasets showing high statistical significance solidly support this result, paving a promising path for the application of bipartite trees to naturally high-dimensional interaction problems.

- **Weighted neighbors prototypes:** Results from the original PBCT paper (**?**) suggest that when one of the entities composing a pair being predicted is present in the training set, restricting the prototype calculation in the final leaf to the labels of the known instance improves predictive performance. We employ this procedure for the first time in forests of bipartite trees and generalize the idea by exploring several similarity-weighted averages for prototype calculation, showing that quadratic similarity weighting, in particular, significantly outperforms the remaining options when new entities are presented.

- **Semi-supervised impurities:** Taking the intrinsic large amount of unlabeled data of interaction problems into account, we are the first to explore the use of semi-supervised impurity metrics in the training of bipartite forests, while also proposing an efficient impurity function for kernel matrices, once again acknowledging the scalability concerns of bipartite learning. We show that this modification can significantly improve the model's resilience to unknown interactions by measuring their performance under random masking of positive labels in the training set.

1.2 Related work

While early proposals of bipartite algorithms for drug-target interaction prediction already make use of the concept of local models, (**?**) seem to be the first to formally define the ideas of the GSO and LMO adaptations of traditional machine learning algorithms. In their study, (**?**) analyses the performance of Random Forests breiman2001random (RF) and Extremely Randomized Trees geurts2006extremely (ExtraTrees) adapted in both ways to be applied to DTI prediction, suggesting a greater potential to generalization of ExtraTrees in comparison to RF.

In **??**, (**?**) presents the PBCT algorithm, an original adaptation of decision trees to bipartite scenarios, and demonstrates improved results over the previous, more superficial, adaptations. Although directly considering the bipartite data format, with separate design matrices for each domain, their proposal is still limited by the same asymptotic complexity of GSO-adapted trees. The authors later extend the PBCT algorithm to ensembles, building bipartite versions of Random Forests and ExtraTrees (**?**). Later still, they explore the use of Neighborhood-Regularized Logistic Matrix Factorization (NRLMF) liu2016neighborhood as a pre-training step to bipartite ExtraTrees, generating continuous pseudo-labels for negative interactions to be inputted to the forest estimator. They demonstrate significant improvements in predictive performance resulting from this procedure (**?**), reinforcing the hypothesized potential of semi-supervised techniques. Nevertheless, the authors refrain from experimenting with random forests, and their bipartite models are still subject to the same training complexity.

(**?**) and independently propose a way to consider semi-supervised assumptions on the level of a decision tree's split search procedure rather than the whole model. They do so by penalizing the selection of split points that separate close training instances, so that instances within the same tree node, that tend to yield the same output, also will now tend to have similar feature vectors. Under the assumption that neighboring instances are more likely to have similar labels (the *smoothness* assumption), delegating part of the split's influence to the feature space directly is expected to reduce the impact of missing or incorrect labels.

More recently, (**?**) applied a more computationally efficient variant of this idea to bipartite trees, using a semi-supervised evaluation not in every possible split but to choose between the two best supervised splits of each domain at each tree node. The authors, however, do not explore ensembles of bipartite trees or the more expensive semi-supervised evaluation of every candidate split.

In the present work, we thus demonstrate how bipartite trees can be grown faster with a simple modification of their training procedure. We also apply the original proposal of a semi-supervised split choice criterion to bipartite trees and compare it to the more efficient variant proposed by (**?**). We also propose an intermediate-complexity approach for kernel features, enabling semi-supervised evaluation of every candidate split while still being faster than the original idea.

1.3 Main objectives

1.4 Main contributions

2 DEVELOPMENT

2.1 Definitions

2.1.1 Mathematical notation

For any given matrix M, we denote by $M^{[ij]}$ its element on the i-th row and j-th column ($i, j \in \mathbb{N}^*$). Analogously, we represent by $M^{[i\cdot]}$ the vector containing M's i-th row so that $M^{[i\cdot][j]} = M^{[ij]}$ and by $M^{[\cdot j]}$ the column vector $((M^\intercal)^{[j\cdot]})^\intercal$ referring to the j-th column of M so that $M^{[\cdot j][i1]} = M^{[ij]}$. Defining the index notations as superscripts frees the subscripts to be used only as indentifiers, naming the matrix or vector as a whole and not in an element-wise fashion. Indices are also always represented by a single letter, to dispense the use of separators between them.

Inspired by the usual notation $|\cdot|$ for the cardinality of a set, we write the total number of M's rows as $|M|_i$ and its number of columns as $|M|_j$. The total number of elements in M is written $|M| = |M|_j|M|_i$, not to be confused with the determinant of M.

We display filtered matrices or vectors by writing the condition as the index, optionally enclosed by parentheses when necessary or improving readability (Eq. **??**).

$$M^{[(i<3)j]} \equiv \{M^{[kj]} \mid k < 3\}$$

When summing over all indices in a given dimension, we took the freedom of omitting the start and end positions (**??**).

$$\sum_i M^{[ij]} \equiv \sum_{i=1}^{|M|_i} M^{[ij]}$$

We also made the choice of representing averages in a more concise way, optionally pondered by sets of w_1 and w_2 weights in each respective axis (**??**).

$$\begin{aligned} M^{\langle i\rangle[j]} &\equiv \frac{\sum_i w_1^{[i]} M^{[ij]}}{\sum_i w_1^{[i]}} \\ M^{[i]\langle j\rangle} &\equiv \frac{\sum_j w_2^{[j]} M^{[ij]}}{\sum_j w_2^{[j]}} \\ M^{\langle ij\rangle} &\equiv \frac{\sum_j \sum_i w_2^{[j]} w_1^{[i]} M^{[ij]}}{\sum_j \sum_i w_2^{[j]} w_i^{[i]}} \end{aligned} \tag{2.1}$$

The enclosing of indices within brackets also allows for the omission of parentheses when concomitantly using exponents, as exemplified by Eq. **??**, and we additionally reserve ourselves the freedom of representing each index only once, which in the last two shown cases requires preemptively defining that i and j respectively represent rows and columns. Notice that dispensing parentheses makes important the order in which the exponent and averaged indices (those within $\langle\cdot\rangle$) appear. The position of indices within $[\cdot]$ is however facultative, and is here chosen to be as close as M as possible in order to avoid confusion with $M^2 = MM$. Also notice that the indices within $[\cdot]$ will be the indices of the resulting matrix or vector.

$$\begin{aligned} M^{[ij]2} &= ((M^{[ij]})^2)^{[ij]} \\ M^{\langle ij\rangle 2} &= (M^{\langle ij\rangle})^2 \\ M^{2\langle ij\rangle} &= (M^{[ij]2})^{\langle ij\rangle} \\ M^{[i]2\langle j\rangle} &= M^{[ij]2\langle j\rangle} \\ M^{[j]2\langle i\rangle} &= M^{[ij]2\langle i\rangle} \end{aligned} \tag{2.2}$$

2.1.2 Problem statement

The supervised machine learning applications focus on modeling a function $f\colon \mathbb{R}^{n_f} \to \mathbb{R}^{n_o}$ whose exact underlying mechanism is unknown or costly to implement. As a result, the only information available about such mapping is a set of inputs $\{x_i \in \mathbb{R}^n_f\}$ and their corresponding outputs $\{y_i \in \mathbb{R}^n_o\}$ of that given function. The goal is therefore to build an *in silico* model (or estimator) $\tilde{f}$ that approximates f, yielding as similar as possible outputs for the same given input, even and especially for outputs not utilized in the process of building $\tilde{f}$.

The known input vectors are usually organized as rows of an X matrix so that $X^{[ij]} = x_i^{[j]}$, and we refer as *feature* or *attribute* to each specific horizontal position j of $x^{[j]}$, which corresponds to a column of X. Likewise, a Y matrix is built with their corresponding outputs ($Y^{[ik]} = y_i^{[k]}$). Commonly referred to as "targets" in the context of regression learning, we here call the known outputs of the modeled process by *labels*, as in classification, even if real-valued, to avoid confusion when referring to the protein targets of a drug.

In the present setting, we concentrate on problems involving the interaction of two domains of instances (also called sample groups). As such, each sample domain forms a different X matrix, that we term X_1 and X_2. Only inter-domain interactions are allowed, that is, instances are restricted from interacting with others in the same sample group, so that the interaction network constitutes an undirected bipartite graph.

The output, in our case, is any scalar piece of information describing the interaction between a given instance pair, such as the rating of a movie given by a user or a kinetic parameter of an enzyme-substrate reaction. The labels are then disposed in a $|X_1|_i$ by $|X_2|_i$ adjacency matrix Y (also called interaction matrix) so that the function to be modeled can now be formulated as mapping the pair's vector representations to the interaction label $f\colon (X_1^{[i\cdot]}, X_2^{[k\cdot]}) \mapsto Y^{[ik]}$.

Since each sample in X_1 refers to a *row* of Y and each sample in X_2 corresponds to a *column* of Y, we sometimes refer to the sample domains of X_1 and X_2 as *row samples* and *column samples*, respectively. We call these datasets *bipartite*, to differentiate from the more common *monopartite* problems, in which a single X matrix is utilized.

2.2 Stablished algorithms for interaction prediction

2.2.1 Linear models

One of the simplest approaches to learning problems in general is to assume a linear relationship between the input features and output labels. Formally, for the non-bipartite case, one assumes that the training output matrix Y can be approximated as follows:

$$\hat{Y} = XW \tag{2.3}$$

in which W is a matrix representing the set of parameters to be learned. To determine W, the mean squared error (MSE) is usually defined as the loss function to be minimized, to which we add an extra regularization term controlled by the hyperparameter α:

$$\mathcal{J} = \frac{1}{2}\|Y - \hat{Y}\|^2 + \frac{\alpha}{2}\|W\|^2 = \frac{1}{2}\|Y - XW\|^2 + \frac{\alpha}{2}\|W\|^2 \tag{2.4}$$

An analytical solution for W can be obtained by taking the derivative of $\mathcal{J}$ with respect to W and setting it to zero:

$$\frac{\partial \mathcal{J}}{\partial W} = 0 = X^\intercal(XW - Y) + \alpha W \implies W = (X^\intercal X + \alpha I)^{-1} X^\intercal Y \tag{2.5}$$

There are scenarios, however, where the specific values of X are less interesting than the pairwise similarities between them. In those settings, while X may not be directly available, we do have access to similarity matrices S (also called *kernel* matrices) in which $S^{[ij]}$ designates a similarity score between $X^{[i]}$ and $X^{[j]}$. Rather than simply considering S in the same way we would treat X in linear regression, we could instead employ the *kernel trick* (**?**): replacing the $XX^\intercal$ terms in the above equations by S. In this case, we are assuming that similarities $S^{[ij]}$ represent the internal product of the vectors $X^{[i]}$ and $X^{[j]}$ in some feature space, which is based on the intuition that the internal product by itself can be regarded as a similarity metric. We also define W slightly differently, ommiting the $X^\intercal$ factor as $W = (S + \alpha I)^{-1} Y$, so that the final prediction is given by

$$\hat{Y} = SW = (S + \alpha I)^{-1} S \tag{2.6}$$

For the bipartite interaction prediction case, besides standard adaptations as described in **??**, a unique formulation is presented by vanlaarhoven2011gaussian. Similar in concept to the standard global single output procedure (**??**), the authors consider each interaction pair as a unitary instance. The authors then propose building a kernel matrix relating each pair of instances to another pair, and not each interacting entity to another of the same domain. If $S_1 \in \mathbb{R}^{n_1 \times n_1}$ and $S_2 \in \mathbb{R}^{n_2 \times n_2}$ are the intra-domain similarity matrices, the global kernel matrix S is defined as

$$S^{[(i_1 n_2 + i_2)(j_1 n_2 + j_2)]} = S_1^{[i_1 j_1]} S_2^{[i_2 j_2]} \tag{2.7}$$

or, more succinctly, as the Kronecker product of S_1 and S_2:

$$S = S_1 \otimes S_2 \tag{2.8}$$

Each entry on S thus represents the similarity between the pair $X_1^{[i_1]}$-$X_2^{[j_1]}$ and another pair $X_1^{[i_2]}$-$X_2^{[j_2]}$ by the product of the similarities between $X_1^{[i_1]}$ and $X_1^{[j_1]}$ and between $X_2^{[i_2]}$ and $X_2^{[j_2]}$.

The bipartite linear regression is then framed on the vectorized Y, denoted vec(Y), built by concatenating the columns of Y into a single $|Y|$ by 1 column vector. Purely for notation purposes, we organize the weight parameters as the vectorized version of a matrix W with the same dimensions of Y.

$$\text{vec}(\hat{Y}) = S\,\text{vec}(W) \approx \text{vec}(Y) \tag{2.9}$$

$$\text{vec}(W) = (S + \alpha I)^{-1}\text{vec}(Y) \tag{2.10}$$

As the reader may imagine, S gets prohibitively large for big datasets (it's a $n_1 n_2$-sized square matrix!), both in terms of memory usage and the time needed to perform the matrix inversion. The authors, however, provide a clever way of circumventing this issue by decomposing each of the S_1 and S_2 kernel matrices separately and exploiting the properties of the Kronecker product.

Given that S_1 and S_2 are symmetric square matrices, it follows from the spectral theorem () that they can be decomposed as follows:

$$S_1 = U_1 \Lambda_1 U_1^\intercal$$
$$S_2 = U_2 \Lambda_2 U_2^\intercal$$

where, if λ_1 represents the vector of eigenvalues of S_1, Λ_1 is the diagonal matrix of those eigenvalues ($\Lambda_1 = \text{diag}(\lambda_1)$), with U_1 columns representing their corresponding eigenvectors $U^{\intercal [i]}$ for each $\lambda_1^{[i]}$. The symmetry of the similarity matrices also implies that U_1 and U_2 are orthogonal, i.e. $U_1^\intercal U_1 = U_1 U_1^\intercal = \mathbb{I}$ and $U_2^\intercal U_2 = U_2 U_2^\intercal = \mathbb{I}$, or, equivalently, $U_1^{-1} = U_1^\intercal$ and $U_2^{-1} = U_2^\intercal$. Utilizing the fact that $(AB) \otimes (CD) = (A \otimes C)(B \otimes D)$ (), the Kronecker product of S_1 and S_2 can be written as

$$S = S_1 \otimes S_2 = (U_1 \Lambda_1 U_1^\intercal) \otimes (U_2 \Lambda_2 U_2^\intercal) = (U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(U_1 \otimes U_2)^\intercal = U \Lambda U^\intercal \tag{2.11}$$

in which we denote $U = U_1 \otimes U_2$ and $\Lambda = \Lambda_1 \otimes \Lambda_2$. W now becomes

$$\text{vec}(W) = (U \Lambda U^\intercal + \alpha \mathbb{I})^{-1}\text{vec}(Y)$$

Further exploring the orthogonality of U, $UU^\intercal = U\mathbb{I}U^\intercal = \mathbb{I}$, so that

$$\text{vec}(W) = U(\Lambda + \alpha \mathbb{I})^{-1} U^\intercal \text{vec}(Y)$$

The most crucial property of the Kronecker product for our application is its relationship with the vectorization operator (): $(A \otimes B)\text{vec}(C) = \text{vec}(BCA^\intercal)$, which allows us to write

$$\text{vec}(W) = U(\Lambda + \alpha \mathbb{I})^{-1}(U_1^\intercal \otimes U_2^\intercal)\text{vec}(Y) = U(\Lambda + \alpha \mathbb{I})^{-1}\text{vec}(U_2^\intercal Y U_1)$$

Since $(\Lambda + \alpha\mathbb{I})^{-1}$ is diagonal, its multiplication by the vector $\text{vec}(U_2^\intercal Y U_1)^\intercal$ can be expressed as a Hadamard product (element-wise multiplication, denoted by $\odot$) between two vectors. Acting element-wise, the Hadamard product is unaffected by vectorization, so that we can simplify the above expression by employing the matrix

$$(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1[ij]} = \frac{1}{\alpha + \lambda_1^{[i]} \lambda_2^{[j]}} \tag{2.12}$$

In this context, $\lambda_1 \otimes \lambda_2^\intercal$ represents the n_1 by n_2 matrix resulting from the *outer product* of the vectors containing the eigenvalues of S_1 and S_2, respectively, while $A^{\circ -1}$ denotes the Hadamard inverse, corresponding to the matrix formed by taking the reciprocal of each element in A. Thus,

$$\begin{aligned}\text{vec}(W) = U\text{diag}(\Lambda + \alpha^{-1}\mathbb{I}) \odot \text{vec}(U_2^\intercal Y U_1) = \\ = (U_1 \otimes U_2)\text{vec}((\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)) = \\ = \text{vec}(U_2[(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)]U_1^\intercal)\end{aligned}$$

Which yields

$$W = U_2[(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)]U_1^\intercal \tag{2.13}$$

Finally, predictions for a new group of instances in the test set are obtained as follows from the similarities with the train instances ($S_{1,\text{ test}}^{[ij]}$ specifies the similarity between $X_{1,\text{ test}}^{[i]}$ and $X_{1,\text{ train}}^{[j]}$).

$$\text{vec}(\hat{Y}_{\text{test}}) = (S_{1,\text{ test}} \otimes S_{2,\text{ test}})\text{vec}(W) = \text{vec}(S_{2,\text{ test}} W S_{1,\text{ test}}^\intercal) \tag{2.14}$$

which summarizes to

$$\hat{Y}_{\text{test}} = S_{2,\text{ test}} U_2[(\alpha + \lambda_1 \otimes \lambda_2^\intercal)^{\circ -1} \odot (U_2^\intercal Y U_1)]U_1^\intercal S_{1,\text{ test}}^\intercal \tag{2.15}$$

2.2.2 Neighborhood-Regularized Matrix Factorization

2.2.2.1 Traditional matrix factorization

Several algorithm proposals have been made throughout the last decades to tackle the problem of bipartite interaction prediction.

The idea behind matrix factorization is to find an approximation of the interaction matrix Y by decomposing it into two matrices U and V of lower dimensions, such that

$$\hat{Y} = UV^T \approx Y \tag{2.16}$$

The rows of U and V can be seen as learned representations in a new vector space (usually called *latent* space) of each sample in the row and column domains, so that $U^{[i]}$ represents $X_1^{[i]}$ and $V^{[j]}$ represents $X_2^{[j]}$. Notice that the number of latent features is constrained by **??** to be the same for both instance domains: $|U|_j = |V|_j$, being an arbitrary hyperparameter to be defined by the user. Usually, the number of latent features is set to be much smaller than the number of original features ($|U|_j \ll |X_1|_j$ and $|V|_j \ll |X_2|_j$), requiring less computational labor and generating models less susceptible to overfitting.

The learning procedure of matrix factorization algorithms thus consists of obtaining such latent feature matrices U and V so to approximate Y as best as possible. The most common approach is to define a loss function that penalizes the difference between the predicted and the true values of Y and to employ gradient descent techniques to gradually change U and V in the direction that minimizes such loss.

As can be deduced from **??**, the dot product of each row of U and V approximates the corresponding element of Y:

$$\hat{Y}^{[ij]} = U^{[i]} \cdot V^{[j]} \tag{2.17}$$

Logistic matrix factorization (LMF) slightly redefines the problem by introducing one more step to obtain $\hat{Y}$ from U and V (**?**): it assumes that the interaction matrix Y is the result of the logistic function applied to $UV^\intercal$ and not only $UV^\intercal$ anymore.

$$\hat{Y}^{[ij]} = \text{logistic}(U^{[i]} \cdot V^{[j]}) = \frac{\exp(U^{[i]} \cdot V^{[j]})}{1 + \exp(U^{[i]} \cdot V^{[j]})} \tag{2.18}$$

where $\mathbf{a} \cdot \mathbf{b}$ represents the dot product between the vectors $\mathbf{a}$ and $\mathbf{b}$. When applied to a matrix, we assume that $\log$ and $\exp$ functions operate element-wise:

$$(\log M)^{[ij]} = \log M^{[ij]}$$

If $\hat{Y}$ is interpreted as the probability of being positive as predicted by the model ($\hat{Y}^{[ij]} = P(Y^{[ij]} = 1)$ and $1 - \hat{Y}^{[ij]} = P(Y^{[ij]} = 0)$), the optimization objective is usually based on maximizing the joint probability of correctly guessing all interactions in Y:

$$P(\hat{Y} = Y) = \prod_{ij} |\hat{Y}^{[ij]} + Y^{[ij]} - 1| \tag{2.19}$$

in which $|a|$ represents the absolute value of a. A few modifications are then further made:

1. The logarithm of the objective function is taken to simplify the expression without affecting the optimization problem, since $\log$ is a monotonic function;

2. Since positive interactions are usually less numerous but more important in matrix completion problems, a factor α is introduced to prioritize them, multiplying the terms corresponding to $Y^{[ij]} = 1$ in the objective function. It results as if $alpha$ copies of each positive interaction are present in the training set;

3. To discourage overfitting and avoid U and V being arbitrarily large, quadratic regularization terms are added, penalizing the magnitude of the elements of U and V.

4. Similarity information between samples is incorporated by NRLMF,

To simplify notation, we define matrices J whose combined sum of all elements corresponds to the objective function $\mathcal{J}$:

$$\mathcal{J} = \sum J_{\text{labels}} + \sum J_{1,\text{ regularization}} + \sum J_{2,\text{ regularization}} + \sum J_{1,\text{ neighborhood}} + \sum J_{2,\text{ neighborhood}} \tag{2.20}$$

where by $\sum J$ we denote $\sum_i^{|J|_i} \sum_j^{|J|_j} J$. The sums must be performed individually due to the J matrices having different dimensions.

Applying the logarithm to **??** we have our first term of $\mathcal{J}$:

$$J_{\text{labels}} = \log |\hat{Y} + Y - 1| = Y \odot \log \hat{Y} + (1 - Y) \odot \log(1 - \hat{Y})$$

where we separate the cases in which $Y^{[ij]} = 1$ and $Y^{[ij]} = 0$. $\odot$ represents the Hadamard product (element-wise multiplication) between matrices:

$$(A \odot B)^{[ij]} = A^{[ij]} B^{[ij]}$$

Adding the aforementioned positive importance factor α and expanding $\hat{Y}$ according to **??** we have

$$\begin{aligned} J_{\text{labels}} &= \alpha Y \odot \{UV^\intercal - \log[1 + \exp(UV^\intercal)]\} - (1 - Y) \odot \log[1 + \exp(UV^\intercal)] = \\ &= \alpha Y \odot UV^\intercal + [(1 - \alpha)Y - 1] \odot \log[1 + \exp(UV^\intercal)] \end{aligned} \tag{2.21}$$

To discourage large values in U and V, we consider quadratic regularization terms weighted by hyperparameters λ_1 and λ_2:

$$J_{1,\text{ regularization}} = -\frac{\lambda_1}{2} U \odot U = -\frac{\lambda_1}{2} \mathbb{I} \odot (UU^\intercal) \tag{2.22}$$

$$J_{2,\text{ regularization}} = -\frac{\lambda_2}{2} V \odot V = -\frac{\lambda_2}{2} \mathbb{I} \odot (VV^\intercal) \tag{2.23}$$

If the initial objective proposal can be interpreted as maximizing the probability of guessing all labels correctly given specific U and V ($\sum J_{\text{labels}} = \log P(\hat{Y} = Y \mid U, V)$), adding the regularization terms is equivalent to introduce prior assumptions about U and V distributions and define a slightly different objective: maximizing the posterior probability to obtain the current U and V given that $\hat{Y} = Y$. Applying Bayes' theorem and assuming $P(Y) = 1$ we have

$$P(U, V \mid \lambda_1, \lambda_2, Y) = P(Y \mid U, V)\; P(U \mid \lambda_1)\; P(V \mid \lambda_2) \tag{2.24}$$

Under the assumption that the values in U and V follow zero-centered spherical gaussian distributions with variances given by $\frac{1}{\lambda}$, that is, $U^{[ij]} \sim \mathcal{N}(0, \lambda_1^{-1}\mathbf{I})$ and $V^{[ij]} \sim \mathcal{N}(0, \lambda_2^{-1}\mathbf{I})$, we recover the regularized objective function of **??** (**?**).

$$\begin{aligned} &\log P(U, V \mid \lambda_1, \lambda_2, Y) = \\ &\quad = \log P(Y \mid U, V) + \sum \log(\exp(-\frac{\lambda_1}{2} U \odot U)) + \sum \log(\exp(-\frac{\lambda_2}{2} V \odot V)) = \\ &\quad = \sum J_{\text{labels}} - \sum \frac{\lambda_1}{2} U \odot U - \sum \frac{\lambda_2}{2} V \odot V \end{aligned} \tag{2.25}$$

Therefore, if multiple values of U and V possibly generate the same $\hat{Y} = Y$, applying the regularization can be understood as not only finding one of such combinations but, between those U and V that satisfy $\hat{Y} = Y$, finding the U and V that are most likely to be randomly sampled. If $P(U, V \mid \lambda_1, \lambda_2, Y)$ continuously varies as a function of U and V, pooling U and V from a region of maximal $P(U, V \mid \lambda_1, \lambda_2, Y)$ should improve generality, arguably ensuring that stochastic deviations of U and V would still result in high $P(\hat{Y} = Y)$ and justifying the use of regularization as a way to avoid overfitting.

One may have noticed that the original feature matrices X_1 and X_2 were not considered in any regard when describing matrix factorization and detailing the objective functions. Born in the context of recommendation systems where the relationship labels are usually the only information available, matrix factorization algorithms in general encounter a significant issue when brought to our current scenario of bipartite interaction prediction: in its canonic formulation, they do not take sample-level features into account, often called *side information* or *side features* in the recommendation field (), possibly overlooking valuable data. As a consequence, they are unable to provide predictions for new samples that were not present in the training set, since no information about them is available to be inputted to the model. This is commonly regarded as the *cold-start problem* (). As such, matrix factorization usage is usually restricted to the task of matrix completion, in which the goal is to predict the missing values of a matrix given the values of the remaining elements, without receiving completely new rows or columns during model evaluation ().

2.2.2.2 Neighborhood regularization

Targeting these issues, liu2016neighborhood,liu2017lpinrlmf,liu2020predicting proposes a modification to LMF that incorporates side information into the model, successfully applying their method to interaction problems such as drug-target interaction prediction (**?**), long non-coding RNA interactions (**?**) and microRNA interactions (**?**). The core idea of their technique lies in adding one more term to the objective function, penalizing instances regarded as close when considering the original features but were separated by U and V, placed far from each other in the latent space. As such, the algorithm is called *Neighborhood-Regularized Logistic Matrix Factorization* (NRLMF). To precisely define it, let's consider similarity-weighted adjacency matrices A_1 and A_2 referring to each sample domain that specifies neighborhood relationships between samples. If $S_1^{[ij]}$ denotes a similarity score between $X_1^{[i]}$ and $X_1^{[j]}$, $A_1^{[ij]}$ is set to this similarity value if $X_1^{[j]}$ belongs to the neighborhood of $X_1^{[i]}$, denoted $N(X_1^{[i]})$, and 0 otherwise:

$$A_1^{[ij]} = \begin{cases} S_1^{[ij]} & \text{if } X_1^{[j]} \in N(X_1^{[i]}) \\ 0 & \text{otherwise} \end{cases} \tag{2.26}$$

Multiple options are available for the definition of neighborhoods, such as considering all samples within a certain radius of each other or only the k nearest neighbors of each sample. In this work, following the original proposal of NRLMF (**?**), we will consider the latter, defining A_1

and A_2 as the adjacency matrices of the k-nearest neighbors graphs of X_1 and X_2, respectively. In other words, $N(X_1^{[i]})$ is the set formed by the k rows $X_1^{[j]}$ with the k highest s_ij. In the interaction prediction problems we analyze, similarities are precalculated so that the X matrices directly provide the distance metric over which the nearest neighbors are selected. That is, X_1 and X_2 themselves already constitute pairwise similarity matrices: $S_1^{[ij]} = X_1^{[ij]}$. In general, however, one may need to define a kernel matrix S as a preprocessing step, choosing a distance metric over the original features to be used in the nearest neighbors search such as the Euclidean distance or a radial basis function (**?**). In any case, notice that A is a function of X alone for NRLMF. While Y may also be considered in similar scenarios (as will be discussed ahead), A does not depend on U or V and can be built as a single pre-training step.

The loss term proposed by NRLMF is then given by the sum of the Euclidean distances in the latent space between samples in the same neighborhood, weighted by their similarities:

$$\mathcal{J}_{1,\,\text{neighborhood}} = -\sum_{ij} A_1^{[ij]} \|U^{[i]} - U^{[j]}\|^2 \tag{2.27}$$

$$\mathcal{J}_{2,\,\text{neighborhood}} = -\sum_{ij} A_2^{[ij]} \|V^{[i]} - V^{[j]}\|^2 \tag{2.28}$$

in which $\|\mathbf{v}\|$ represents the Euclidean norm of a vector $\mathbf{v}$. Concentrating on the row instances and expanding the last definition we have

$$\sum_{ij} A_1^{[ij]} \|U^{[i]} - U^{[j]}\|^2 = \sum_{ij} A^{[ij]} \left(U^{[i]} \cdot U^{[i]} + U^{[j]} \cdot U^{[j]} - 2U^{[i]} \cdot U^{[j]}\right) =$$

$$= \sum_i \left(\sum_j A_1^{[ij]}\right) U^{[i]} \cdot U^{[i]} + \sum_j \left(\sum_i A_1^{[ij]}\right) U^{[j]} \cdot U^{[i]}$$

The terms in which U appears with the same index ($U^{[i]} \cdot U^{[i]}$ and $U^{[j]} \cdot U^{[j]}$) can be rewritten to include both by multiplying them by the identity matrix $\mathbb{I}$. Essentially, we consider $\sum_i U^{[i]} \cdot U^{[i]} = \text{trace}(UU^\intercal) = \sum_{ij} (\mathbb{I} \odot UU^\intercal)^{[ij]}$.

$$\sum_i \left(\sum_j A_1^{[ij]}\right) U^{[i]} \cdot U^{[i]} = \sum_i \left(\sum_j A_1^{[ij]}\right) \sum_k \mathbb{I}^{[ik]} U^{[i]} \cdot U^{[k]} = \sum_{ij} \left(\sum_k A_1^{[ik]}\right) \mathbb{I}^{[ij]} U^{[i]} \cdot U^{[j]}$$

This allows us to write

$$\sum_{ij} A_1^{[ij]} \|U^{[i]} - U^{[j]}\|^2 =$$

$$= \sum_{ij} \left[\left(\sum_k A_1^{[ik]} + \sum_l A_1^{[lj]} \right) \mathbb{I}^{[ij]} - 2A_1^{[ij]} \right] U^{[i]} \cdot U^{[j]} =$$

$$= \sum L_1 \odot (UU^\intercal)$$

in which we define

$$L_1^{[ij]} = \left(\sum_k A_1^{[ik]} + \sum_l A_1^{[lj]} \right) \mathbb{I}^{[ij]} - 2A_1^{[ij]} = \left(\sum_k A_1^{[ik]} + A_1^{[kj]} \right) \mathbb{I}^{[ij]} - 2A_1^{[ij]} \tag{2.29}$$

Notice that taking $A_1^\intercal$ instead of A_1 has no effect on the final result, since $\sum A_1 UU^\intercal = \sum A_1^\intercal UU^\intercal$. We could then work with a symmetrized version of A_1 from the start:

$$\tilde{A}_1 = A_1 + A_1^\intercal$$

yielding

$$L_1 = \left(\sum_k \tilde{A}_1^{[ik]}\right)\mathbb{I}^{[ij]} - \tilde{A}_1^{[ij]}$$

We can see that the first term of L_1 acts in a similar way to the quadratic regularization terms presented by **??**, multiplying the main diagonal of $UU^\intercal$ and thus penalizing the model for latent vectors with large Euclidean norms (the diagonal of $UU^\intercal$ holds the squared norms $U^{[i]} \cdot U^{[i]}$). The amount of regularization is however pondered by the weighted number of neighbors of each sample in this case: $\sum_k A_1^{[ik]}$ represents the sum of similarities of $X_1^{[i]}$ with all its neighbors, also called the *degree* of a sample. This results in samples with larger and more compact neighborhoods being more heavily penalized for having large norms in the latent space. The second term of **??**, on the other hand, rewards the model for placing close neighbors colinear to each other in the latent space, summing over $S^{[ij]}U^{[i]} \cdot U^{[j]}$ terms between each sample and its neighbors ($A_1^{[ij]}$ is 0 if $U^{[i]}$ and $U^{[j]}$ are not neighbors).

Finally, the neighborhood regularization terms are written as

$$J_{1,\text{ neighborhood}} = -\frac{\beta_1}{2} L_1 \odot (UU^\intercal) \tag{2.30}$$

$$J_{2,\text{ neighborhood}} = -\frac{\beta_2}{2} L_2 \odot (VV^\intercal) \tag{2.31}$$

Combining the matrix terms as in **??**, NRLMF's objective function is given by

$$\begin{aligned}\mathcal{J} = &\sum \alpha Y \odot UV^\intercal + [(1-\alpha)Y - 1] \odot \log\left[1 + \exp(UV^\intercal)\right] \\ &- \sum \frac{1}{2}(\lambda_1 \mathbb{I} + \beta_1 L_1) \odot (UU^\intercal) \\ &- \sum \frac{1}{2}(\lambda_2 \mathbb{I} + \beta_2 L_2) \odot (VV^\intercal)\end{aligned} \tag{2.32}$$

and the derivatives of the objective function with respect to U and V to be used in the gradient descent procedure are given by

$$G_U = \frac{\partial \mathcal{J}}{\partial U} = \{[(1-\alpha)Y - 1] \odot \hat{Y} + \alpha Y\}V - (\lambda_1 \mathbb{I} + \beta_1 L_1)U \tag{2.33}$$

$$G_V = \frac{\partial \mathcal{J}}{\partial V} = \{[(1-\alpha)Y - 1] \odot \hat{Y} + \alpha Y\}^\intercal U - (\lambda_2 \mathbb{I} + \beta_2 L_2)V \tag{2.34}$$

The training procedure of NRLMF is presented by **??**, and consists of alternated updates on U and V in the gradient's direction until certain stop criteria are satisfied. Common choices for stopping conditions are a maximum number of iterations or a minimum change in the objective function between iterations.

Since faster convergence is reported by the original authors (**?**), we follow johnsonlogistic,liu2016neighborhooda by implementing the AdaGrad procedure (**?**), in which the length of each gradient step is divided by the square-root sum of squared previous steps:

$$U_{t+1} = U_t + \frac{\eta G_{U,t}}{\sqrt{\sum_{t'=0}^{t} G_{U,t'}^2}} \tag{2.35}$$

where $G_{U,t}$ is the partial derivative $\frac{\partial \mathcal{J}}{\partial U}$ of the objective function with respect to U at step t, and η is the user-defined learning rate. The same is done for V.

Función TrainNRLMF(Y, S_1, S_2, α, λ_1, λ_2, β_1, β_2, η): Train an NRLMF model.

```
    Input: $Y$, the training labels matrix to be approximated.
    Input: $S_1$, $S_2$, Similarity matrices among instances of each axis.
    Input: $\alpha$, the positive importance factor.
    Input: $\lambda_1$, $\lambda_2$, quadratic regularization factors.
    Input: $\beta_1$, $\beta_2$, neighborhood regularization factors.
    Input: $\eta$, the learning rate.
    Output: $U$, $V$, the resulting latent feature matrices.
 1  Optionally precompute constant factors of the gradient (Equations ?? and ??), such
      as $(\lambda_2 \mathbb{I} + \beta_2 L_2)$, $[(1-\alpha)Y - 1]$ or $\alpha Y$;
 2  Initialize $U$ and $V$ with normally-distributed random values;
 3  $T_U, T_V \leftarrow \mathbf{0}$;            // Initialize gradient accumulators
 4  while Stop conditions are not met do
        // Update U
 5      Obtain $G_U$ through ??;
 6      $T_U \leftarrow T_U + G_U^2$;
 7      $U \leftarrow U + \eta \frac{G_U}{\sqrt{T_U}}$;
        // Update V
 8      Obtain $G_V$ through ??;
 9      $T_V \leftarrow T_V + G_V^2$;
10      $V \leftarrow V + \eta \frac{G_V}{\sqrt{T_V}}$;
11  end
12  return $U$, $V$
```

The main importance of NRLMF however lies in the inference phase. As mentioned, matrix factorization methods are not designed to deal with new input samples, that are not present in the training set. Specifically, traditional matrix factorization is incapable of generating latent vectors for the unseen samples to be used for label prediction. An idea that may seem natural at first glance is to delay training to the arrival of new instances, including them in the training set with zero-only labels before performing the optimization. But even then, using an objective function based only on Y as is traditionally done, no new information is brought by the new instances and adding new zeroed rows or columns to Y will mainly introduce noise to the training data and likely degrade the model's predictive performance.

NRLMF, however, leverages proximity information encoded by X to remarkably enable determining latent feature vectors for completely new instances. The neighborhood regularization terms in the objective function now reveal their full importance: they support proximity as a transferable property between the original and the latent spaces. By encouraging that neighbors in X_1 and X_2 remain close in U and V, we can infer latent features of new instances based on their neighborhood.

Consider the test similarity matrices $S_{1,\,\text{test}}$ and $S_{2,\,\text{test}}$ respectively derived from $X_{1,\,\text{test}}$ and $X_{2,\,\text{test}}$, relating the new instances to the known training samples. For instance, $S^{[ij]}_{1,\,\text{test}}$ represents the similarity between $X^{[i]}_{1,\,\text{test}}$ and $X^{[j]}_{1,\,\text{train}}$. If $A_{1,\,\text{test}}$, like before in **??**, accordingly restricts the similarity matrix to the neighborhood of each sample,

$$A^{[ij]}_{1,\,\text{test}} = \begin{cases} S^{[ij]}_{1,\,\text{test}} & \text{if } X^{[j]}_{1,\,\text{train}} \in N(X^{[i]}_{1,\,\text{test}}) \\ 0 & \text{otherwise} \end{cases} \tag{2.36}$$

the latent feature vector of a new instance is simply estimated as the weighted average of its neighbors' latent representations:

$$U^{[i]}_{\text{test}} = \frac{A^{[i]}_{1,\,\text{test}} U_{\text{train}}}{\sum A^{[i]}_{1,\,\text{test}}} \tag{2.37}$$

the analogous being held for V, so that new predictions are made as usual with **??**, where U_{test}, U_{train}, V_{test} and V_{train} can be used in accordance with the prediction task under study (see **??** for details on the different prediction scenarios):

$$\hat{Y}_{\text{TT}} = \frac{\exp(U_{\text{test}} V^{\intercal}_{\text{test}})}{1 + \exp(U_{\text{test}} V^{\intercal}_{\text{test}})} \quad \hat{Y}_{\text{TL}} = \frac{\exp(U_{\text{test}} V^{\intercal}_{\text{train}})}{1 + \exp(U_{\text{test}} V^{\intercal}_{\text{train}})} \quad \hat{Y}_{\text{LT}} = \frac{\exp(U_{\text{train}} V^{\intercal}_{\text{test}})}{1 + \exp(U_{\text{train}} V^{\intercal}_{\text{test}})} \tag{2.38}$$

2.2.2.3 Further developments on NRLMF

2.2.3 Applying traditional estimators to bipartite data

As previously stated in **??**, bipartite interaction problems fundamentally differ from the usual machine learning paradigm, in which input data represents a single entity to be labeled. Interaction tasks are instead concerned with labeling a relationship between two entities, and as such, each prediction is made upon a pair of feature vectors, each vector being specific to each of the two sample domains.

Such subtlety is most often bypassed by reformulating a bipartite dataset into the traditional machine learning setting (**?**). These strategies can be encompassed under two general approaches, initially termed *global* and *local* by vert2008reconstruction and later adopted and extended by sschrynemackers2015. Aiming to enhance clarity and generalization, we further refine these categories by defining *global* and *local* as general properties of bipartite estimators, rather than specific training procedures:

- ***Global*** estimators are those aware of both instance domains during the training procedure (X_1 and X_2).
- ***Local*** estimators are those which only have access to feature information from one of the two instance domains during training (either X_1 or X_2). As such, they are often employed in compositions, combining the predictions of several local models to produce the final output.

Furthermore, to be consistent with pliakos2018,pliakos2019,pliakos2020, in our current context of bipartite interactions we assume that:

- ***Single-output*** estimators are those which consider all labels, i.e. all $Y^{[ij]}$ elements, irrespectively of the column j or row i they are in. They are all regarded as a single type of output[*].
- ***Multi-output*** estimators, on the other hand, are those which consider each instance, each row or column of Y, as a separate task, for example by defining a composite loss function formed by the combination of losses over each row or column.

Finally, the two most common ways of adapting monopartite estimators to bipartite data are then named *global single-output* (GSO) and *local multi-output* (LMO), as proposed by pliakos2018,pliakos2019,pliakos2020. We further denote them standard, to clearly distinguish them from new adaptation proposals that could share the globality, locality, or outputness properties, but work in an entirely different way.

Specific definitions and shortcomings of these procedures are now presented.

2.2.3.1 The standard global single-output adaptation

Arguably the most straightforward and commonly used strategy of formatting bipartite data to be inputted into conventional monopartite estimators is through presenting concatenated pairs of row-column samples, labeled by each element of Y. The interacting pair itself is what we abstract as a sample in this case, with its attributes being its component objects' attributes combined. This is usually done by converting the two X matrices and the interaction matrix Y to a single design matrix we call X_{SGSO} and a column-vector of labels we refer to as Y_{SGSO}.

One way of doing so is by choosing indices as described by **??**, where $\begin{bmatrix} x_1 & x_2 \end{bmatrix}$ denotes concatenation and all i_1 and i_2 combinations are explored (see **??**).

[*] Notice that the label matrix Y could be still represented in two dimensions even if the model is single-output in this sense, contrary to the usual case where bidimensionality of Y is a defining characteristic of a multi-output problem.

$$\begin{aligned} i_g &= (i_1 - 1)|X_2|_i + i_2 - 1 \\ X_{\text{SGSO}}^{[i_g\cdot]} &= \begin{bmatrix} X_1^{[i_1\cdot]} & X_2^{[i_2\cdot]} \end{bmatrix} \\ Y_{\text{SGSO}}^{[i_g 1]} &= Y^{[i_1 i_2]} \end{aligned} \tag{2.39}$$

Notice that, in order to consider the interactions of all possible pairs, X_{SGSO} would have $|X_1|_i|X_2|_i$ rows and $|X_1|_j + |X_2|_j$ columns, with Y_{SGSO} having a single column but all the same $|X_1|_i|X_2|_i$ elements as Y. Thus, while reformatting Y has no impact in memory usage, X_{SGSO} as defined by **??** is highly redundant. As a result, naively dealing with such a large X_{SGSO} matrix is impeditive in many cases, both in terms of memory usage and computation time. Therefore, a commonly used workaround is to subsample the 0-labeled interactions, yielding a dataset with equal amounts of interacting (1-labeled) and unknown (0-labeled) pairs (). Although unknown, the 0-labeled pairs are usually far more numerous than 1-labeled and much more likely to be truly negative interactions than false negatives (see Section **??**), justifying the negatives-undersampling strategy. Despite such reasoning, we demonstrate in **??** that taking all negative samples into consideration instead has significant benefits, and we propose new model optimizations to enable it (**??**).

2.2.3.2 The standard local multi-output adaptation

The local approaches, in contrast to global methods, propose training different models on X_1 and X_2, so that each single model only has access to information regarding either row samples or column samples.

As such, multiple monopartite models need to be used in conjunction to predict interactions between new row samples and new column samples. The standard local multi-output (SLMO) approach uses four monopartite models, two for each axis, that we here refer to as *primary* and *secondary* rows (or columns) estimators. In general, they must be multi-output estimators, each being able to receive X_{train} and Y_{train} bi-dimensional matrices in the training step, receive an X_{new} in the prediction step such that $|X_{\text{new}}|_j = |X_{\text{train}}|_j$, and outputting Y_{pred} with $|Y_{\text{pred}}|_i = |X_{\text{new}}|_i$ newly predicted rows and $|Y_{\text{pred}}|_j = |Y_{\text{train}}|_j$ output columns.

The procedure for training the estimators in an SLMO setting consists of simply training the primary estimators, the primary rows estimator being trained on X_1 and Y, and the primary columns estimator on X_2 and $Y^\intercal$ (**??**). The prediction step is however more complicated, involving the training of the secondary models on the newly predicted labels by the primary estimators, optionally combined with the original training data. The procedure is described in details by **??** and illustrated by **??**. The `combine` function used in line **??** of the **??** procedure can be arbitrarily chosen, and is usually defined as the simple element-wise average of both matrices (`combine(`Y_1`, `Y_2`)` $= \frac{1}{2}(Y_1 + Y_2)$).

```
Función TrainLocalModel($X$, $Y$)
    Input: The bipartite training dataset.
    Output: A bipartite local model.
    train(primaryRowsEstimator, $X_1$, $Y$);
    train(primaryColumnsEstimator, $X_2$, $Y^\intercal$);
    return primaryRowsEstimator, primaryColumnsEstimator
```

```
Función PredictLocalModel(primary models, $X_{\text{new}}$)
    Input: The trained primary models and the unseen sample matrices $X_{\text{new}}$ for both
           axes.
    Output: $Y_{\text{pred}}$ predictions for each interaction provided.
    $Y_{\text{new rows}} \leftarrow$ predict(primaryRowsEstimator, $X_{1new}$)
    $Y_{\text{new cols}} \leftarrow$ predict(primaryColumnsEstimator, $X_{2new}$)
    if Secondary estimators consider label dependencies then
        // Concatenate known rows and columns labels to the
           primary predictions
        $Y_{\text{new cols}} \leftarrow \begin{bmatrix} Y^\intercal \\ Y_{\text{new cols}} \end{bmatrix}$;
        $Y_{\text{new rows}} \leftarrow \begin{bmatrix} Y \\ Y_{\text{new rows}} \end{bmatrix}$;
        // Otherwise, if label columns are considered
           independently, this step is not necessary
    end
    train(secondaryRowsEstimator, $X_1$, $Y^\intercal_{new\ cols}$);
    train(secondaryColumnsEstimator, $X_2$, $Y^\intercal_{new\ rows}$);
    $Y_{\text{pred rows}} \leftarrow$ predict(secondaryRowsEstimator, $X_{1new}$);
    $Y_{\text{pred cols}} \leftarrow$ predict(secondaryColumnsEstimator, $X_{2new}$);
    if Secondary estimators consider label dependencies then
        // Skip predictions not referring to $X_{1\text{new}}$ and $X_{2\text{new}}$
        $Y_{\text{pred rows}} \leftarrow Y_{\text{pred rows}}^{[\cdot j > |X_1|_i]}$;
        $Y_{\text{pred cols}} \leftarrow Y_{\text{pred cols}}^{[\cdot j > |X_2|_i]}$;
    end
    return combine($Y_{pred\ rows}$, $Y^\intercal_{pred\ cols}$)
```

Notice that, if the secondary multi-output estimators treat each label independently, including the Y_{train} labels in their training will make no difference, and one should use only the predictions from the primary estimators.

A specific case of a model with multiple independent-outputs occurs when a collection of single-output models is utilized as a unified entity, each being trained on each column of

Y_{train}. This setup was present in the first proposal of a local model (), and enables a wider range of learning algorithms, not just multi-output strategies, to be employed in interaction prediction.

If, however, the secondary estimator is indeed able to take advantage of relationships between outputs, one might consider concatenating the primary estimators' predictions to Y_{train} and use both to train the secondary estimators (see lines **??** of Function **??**). This setting would enable the secondary models to explore the output relationships involving the original training set, which are arguably more reliable than those between the primary predictions alone.

That said, another consideration regarding the use of dependent-outputs secondary estimators is whether or not to provide the whole $X_{1\text{new}}$ and $X_{2\text{new}}$ at once, since doing so would increase the amount of primarily-predicted data used to train the secondary estimators, and it may be desirable to have more columns coming directly from Y_{train} rather than inferred by the primary models. The ideal scenario then would be to run **??** once for every $X_{1\text{new}}$ and $X_{2\text{new}}$ row combination, in a total of $|X_{1\text{new}}|_i|X_{2\text{new}}|_i$ iterations, with possible performance drawbacks for most learning algorithms. The natural intermediate idea would be to provide $X_{1\text{new}}$'s and $X_{2\text{new}}$'s rows in batches, possibly increasing the prediction time but ensuring the $|X_{a\text{new}}|_i/|X_{a\text{train}}|_i$ ratio is not detrimentally high. Additionally, some algorithms allow for output weights to be used in the training procedure, enabling us to assign lower importance to the Y_{new} columns inferred by the primary estimators.

In any case, contrary to what is usually observed, the amount of test data and specific combinations of test instances provided to the SLMO ensemble clearly affect the resulting predictions when the secondary models have inter-dependent outputs. This characteristic should thus be taken into consideration when developing, evaluating and comparing bipartite estimators under the SLMO configuration, although seldomly addressed by previous authors in our experience.

Due to each monopartite model being provided with a much lower number of instances in comparison to the SGSO procedure, SLMO models tends to be naturally faster to train than SGSO models. However, a striking limitation of the SLMO procedure is caused by its inference phase, which requires traing of the secondary models whenever new instances are inputted. The resulting large prediction times hinders the application of SLMO models on online learning scenarios.

2.3 Bipartite decision forests

2.3.1 Interaction matrix approximation methods

The first we call *matrix reconstruction approaches* and includes the usage of matrix factorization or other procedures that convert the sparse label matrix Y into a dense representation, employing sample attributes, sample similarities or network characteristics to replace 0-valued unknown interactions by more meaningful, often real-valued numbers. These new values can be used directly as a probability of interaction, of can serve as input to downstream learning methods in the pipeline.

An important consideration is that these techniques are often stateless estimators, meaning that all calculations must be redone for each new sample that is presented to have its label predicted. To avoid this, some methods include the test instances in the training set, but substituting their labels by 0, so that the only their numerical features are available. Examples are (). We argue that this approach, even if not abrupt, can still pose some data leakage effect, since the estimator is not refrained from the extra information about the feature space. An estimator in this setting, for instance could give more attention to achieving better results in denser regions of the sample space, information which would be clearly affected by the proposed approach. As such, the estimator performance metrics on new data could be ever so slightly biased.

Another idea to circumvent this limitation is through the use of other, traditionally monopartite, estimators in specific steps. () and (), for instance, after learning the U and V latent vectors in a matrix factorization procedure, utilize nearest neighbors estimators to compute new samples' latent vectors as a linear combination of the vectors generated for the training set, optionally weighting neighbors with similarity metrics.

Still, without monopartite models, these matrix reconstruction approaches cannot deal with new samples without retraining the model on the whole training data plus the new instances. We thus consider these techniques more inclined to the preprocessing realm rather than constituting machine learning models by themselves.

2.3.2 Bipartite forests

A *sui generis* learning algorithm adaptation was proposed by pliakos2018 to deal with bipartite data, without the need for dataset reformatting as in SGSO, or compositions of multiple estimators and secondary training steps as SLMO.

Named Predictive Bi-Clustering Tree (PBCT) by the authors, it tunes the usual decision tree-growing algorithm to directly operate on bipartite interactions, building a single tree model directly on bipartite formatted datasets (using X_1, X_2 and Y). Importantly, their proposed algorithm inherits all the benefits of tree-based estimators, such as their well-known interpretability and remarkably low amount of hyperparameters (**?**)).

Intriguingly, PBCTs were only explored in a scarce number of previous studies (**?**, **?**, **?**), to the best of our knowledge. A possible explanation is that no improvement in computational complexity of training times is observed with respect to SGSO-adapted decision trees (**?**), even if drastically less memory is used by a PBCT in comparison to a naive implementation of SGSO. Furthermore, no implementations of PBCTs are provided in sufficiently accessible and extensible formats, which could also have hindered its adoption by the scientific community.

We thus turn our attention onto such tree-based algorithms, proposing optimizations proven to reduce asymptotic training times of bipartite trees by a $\log |X|_i$ factor, enabling larger bipartite datasets to have all its unknown (0-labeled) interactions considered in model training and bringing unprecedented scalability to tree and forest estimators on interaction prediction tasks.

A more thorough description of traditional decision trees is now presented, as a theoretical foundation for the upcoming formal definition of bipartite decision trees.

2.3.3 Traditional decision trees

Let's consider the scenario where a single protein of interest is selected, and we receive the task of determining which drug molecules will likely affect its physical structure or catalytic function, for example. We wish to find a systematic procedure to decide whether a given drug molecule x_i will interact with our protein or not. To develop such a procedure, consider we have at our disposal a set of n_f known drug molecules, whose degree of interaction with our protein of interest was previously experimentally determined. We can then describe a drug molecule x_i in general by how similar it is to each of our n_f known molecules, organizing this information as a vector $x_i = \begin{bmatrix} x_i^{[1]} & x_i^{[2]} & x_i^{[3]} & \cdots & x_i^{[n_f]} \end{bmatrix}$ so that $x_i^{[j]}$ represents the similarity score between the drug x_i and the j-th of our n_f known drugs.

The hypothetical decision procedure we intend to determine could then be structured as a path with consecutive bifurcations. We always start at the same first bifurcation, and at each forking of the path a question is asked about the drug x_i in hands. The questions are in a standard format, exemplified by "Is x_i more than 60% similar to the 3rd known drug?", or $x_i^{[j]} > t$, for a general known drug j and similarity threshold t. The answer to the question in each bifurcation determines which of the two possible paths we should follow. No cycles are allowed in the path, and eventually, all routes reach final locations instead of bifurcations, where a final decision is made about the drug x_i's effect on our protein of interest.

Such a decision procedure, structured as a binary tree path, is what is commonly referred to as a *decision tree* (DT) model, illustrated by **??**. The rules leading to each output being clearly defined along the tree structure is what results in the well-known transparency of decision trees, an attractive characteristic in fields such as drug discovery or regulatory network inference, where insights into the underlying processes are greatly valued.

The described binary-tree structure is by far the most utilized (), being ubiquitous to all tree-based estimators in the present study. The main challenge lies in the building process of such models, in how to determine the rules that define each fork and the stopping criteria for a final decision to be yielded.

As presented by **??**, to build the decision tree we are given a training set X composed of $|X|_i$ entities (drug molecules in the previous example) so that $X^{[i\cdot]} = x_i$ represents the i-th entity. Each entity is described by $|X|_j$ numeric descriptors (similarities to known drugs in the previous example), and to each is assigned one or more numeric labels $Y^{[i\cdot]}$ describing the known target prediction results. In analogy to the last example, the labels would denote interaction or not with one or more proteins of interest.

If we were to execute the prediction process of a decision tree for each training instance, going through the aforementioned branched path, each bifurcation would divide the training instances between those who answer the question affirmatively and those who answer otherwise. The procedure for building a DT thus consists of determining features f and threshold values t that recursively split the dataset in two parts, named *left* and *right* partitions, as defined by **??**.

$$\begin{aligned} Y_{\text{left}} &= \{Y^{[k\cdot]}\ \forall\ k \mid X^{[kf]} \leq t\} \\ Y_{\text{right}} &= \{Y^{[k\cdot]}\ \forall\ k \mid X^{[kf]} > t\} \\ X_{\text{left}} &= \{X^{[kf]}\ \forall\ k \mid X^{[kf]} \leq t\} \\ X_{\text{right}} &= \{X^{[kf]}\ \forall\ k \mid X^{[kf]} > t\} \end{aligned} \tag{2.40}$$

Each bifurcation of a DT, more commonly referred to as a *node*, then represents one of such splits, defined by a selected feature f and a threshold value t, and each of its two children receives one of the data partitions generated by its parent (see **??**). Under specific predefined circumstances, a node stops generating descendant nodes, having no children and taking record of the dataset partition it received from its parent. Among possible stopping criteria are a maximum depth of the tree or a minimum number of samples in a node, for instance. These terminal nodes are called *leaves*.

Under these definitions, the prediction step for a new sample x_{new} as described in the introductory example consists of transversing the tree from the first (or *root*) node until a leaf, following each node's test $x_{\text{new}}^{[f_{\text{node}}]} > t_{\text{node}}$ and selecting the corresponding child node as given by the partitioning rule of **??**. Once a leaf is reached, the tree returns a prototype value calculated over the partition of the training data corresponding to that leaf. The average label ($Y_{\text{leaf}}^{\langle i\rangle[j]}$) of each output is a common choice for this prototype.

As previously stated, the focus of the present work is however the training procedure of a decision tree, the method to determine the f and t parameters of each node in order to effectively estimate the labels of new samples. Most commonly, a top-down induction algorithm (from the root node to the leaves) is followed, and all possible partitions of the given training set are evaluated. The exhaustive evaluation of partitions in each node can be done by, for each feature column $X^{[\cdot f]}$, sorting $X^{[\cdot f]}$ and considering a threshold t between each two consecutive values in it. As seen by **??** and illustrated by **??**, any threshold value between the same two consecutive $X_{\text{sorted}}^{[\cdot f]}$ elements will result in the exactly same partitioning of the training set. The common practice is thus to take the average between the two neighboring feature values.

A greedy procedure is then followed for the overall tree growing, selecting at each node the best t and f to represent the split, according to a predefined quality criteria we further discuss in **??**. The exhaustive split search procedure is detailed by the **??**. The algorithm **??** then describes its application on growing a traditional decision tree (stablished by breiman1984classification), while **??** formally details how predictions are made given a model built by **??** and a new data instance.

Función Predict(RootNode, x): Compute a Decision Tree's prediction.

```
Input: A new interaction sample to be evaluated and the root node of a Decision
       Tree.
Output: The Decision Tree's predicted value for the given sample attributes.
1 Node ← RootNode;
2 while Node is not a leaf do
3     if $x^{[Node.feature]} > Node.threshold$ then
4         Node ← Node.childRight
5     else
6         Node ← Node.childLeft
7     end
8 end
9 return prototype (Node.X, Node.Y)
```

Función BuildTree(X, Y): Recursively build a Decision Tree

```
Input: The training data for the current node.
Output: Current node, with all information of subsequent splits.
1 $Q^*$, $f^*$, $t^*$ ← FindSplit ($X$, $Y$, $\tilde{n}_f$);
  // Many stopping criteria are possible
2 if DecideToStop ($Q^*$, $f^*$, $t^*$, $X$, $Y$) then
3     return NodeObject {
4         isLeaf ← True
5         $X_{\text{leaf}}$ ← $X$
6         $Y_{\text{leaf}}$ ← $Y$
7     };
8 else
9     Get $X_l$, $Y_l$, $X_r$, $Y_r$ from $f^*$ and $t^*$ (Eq. ??);
10    return NodeObject {
11        isLeaf ← False
12        childLeft ← BuildTree ($X_l$, $Y_l$)
13        childRight ← BuildTree ($X_r$, $Y_r$)
14        feature ← $f^*$
15        threshold ← $t^*$
16    };
17 end
```

An alternative to avoid considering all partitioning options and greatly reduce the amount of operations is to draw a random threshold t between the minimum and maximum value of each feature, thus evaluating only $|X|_j$ splits when choosing the best. Although degrading the performance of a single tree, this procedure, described by **??**, is an interesting option when

Función FindSplitBest(X, Y)

```
Input: A partition of the training data in a given node.
Output: The highest quality score $Q^*$ found among all splits evaluated, with its
        corresponding feature column $f^*$ and threshold value $t^*$.
1  Initialize $S_r$ and $S_l$ as a $|Y|_j$-sized vectors;
2  $Q^*$, $f^*$, $t^* \leftarrow \mathbf{0}$;
3  Draw $\tilde{n}_f$ columns (features) of $X$ without replacement;
4  foreach feature index $f$ of the $\tilde{n}_f$ drawn features do
5      $n_l \leftarrow 0$;                                    // Holds $|Y_l|_i$
6      $S_r \leftarrow \sum_i Y^{[ij]}$;                      // Holds $\sum_i Y_r^{[ij]}$
7      $S_l \leftarrow \mathbf{0}$;                           // Holds $\sum_i Y_l^{[ij]}$
8      Get the permutation $P$ that sorts $X^{[\cdot f]}$;
9      Apply $P$ to $Y$'s and $X^{[\cdot f]}$'s rows:
10         $Y_P$, $X_P \leftarrow P(Y)$, $P(X^{[\cdot f]})$;
11     foreach row index $\hat{\imath}$ of $Y_P$ do
12         $n_l \leftarrow n_l + 1$;
13         foreach column index $\hat{\jmath}$ of $Y_P$ do
14             $S_r^{[\hat{\jmath}]} \leftarrow S_r^{[\hat{\jmath}]} - Y_P^{[\hat{\imath}\hat{\jmath}]}$;
15             $S_l^{[\hat{\jmath}]} \leftarrow S_l^{[\hat{\jmath}]} + Y_P^{[\hat{\imath}\hat{\jmath}]}$;
16         end
17         Use $S_l$, $S_r$ and $n_l$ to calculate $Q$ (Eq. ??). Notice that other node-specific
             constants might be needed;
18         if $Q > Q^*$ then
19             $Q^* \leftarrow Q$;
20             $f^* \leftarrow f$;
21             $t^* \leftarrow \frac{1}{2}(X_P^{[\hat{\imath}]} + X_P^{[\hat{\imath}+1]})$;
22         end
23     end
24 end
25 return $Q^*$, $f^*$, $t^*$;
```

building tree ensembles (**??**), being the core idea behind the extremely randomized trees (ERT) algorithm (**?**).

2.3.4 Split quality criteria and impurity metrics

In order to compare and select the best splitting rules at each node, a split quality criterion must be defined. The quality ΔI of a split is commonly framed as the decrease of an impurity metric calculated over the subsets of training labels (**??**). This decrease is taken for the combined impurities of the generated children nodes (**??**). All impurities are normalized by the size of each partition relative to the total number of training samples ($|Y_{\text{root}}|_i$), restricting the

```
Función FindSplit_random($(X, Y)$)
    Input: A partition of the training data in a given node.
    Output: The highest quality score $Q^*$ found among all splits evaluated, with its
            corresponding feature column $f^*$ and threshold value $t^*$.
1   $Q^*, f^*, t^* \leftarrow \mathbf{0}$;
2   Draw $\tilde{n}_f$ columns (features) of $X$ without replacement;
3   foreach feature index $f$ of the $\tilde{n}_f$ drawn features do
4       Find min($X^{[\cdot f]}$) and max($X^{[\cdot f]}$);
5       Draw a random threshold value $t \in \mathbb{R}$ so that min($X^{[\cdot f]}$) $< t <$max($X^{[\cdot f]}$);
6       Calculate $Q$ for the drawn $t$ (Eq. ??);                    // $O(|Y|)$
7       if $Q > Q^*$ then
8           $Q^* \leftarrow Q$;
9           $f^* \leftarrow f$;
10          $t^* \leftarrow \frac{1}{2}(X_P^{[\hat{\imath}]} + X_P^{[\hat{\imath}+1]})$;
11      end
12  end
13  return $Q^*, f^*, t^*$ 1FindSplit_random(1)FindSplit_random
```

effect of nodes with less data that could introduce spurious variations of impurity. Notice that $|Y_{\text{node}}| = |Y_{\text{left}}| + |Y_{\text{right}}|$.

$$\Delta I(Y,t,f) = \frac{|Y_{\text{node}}|}{|Y_{\text{root}}|} I(Y_{\text{node}}) - \frac{|Y_{\text{left}}|}{|Y_{\text{root}}|} I(Y_{\text{left}}) - \frac{|Y_{\text{right}}|}{|Y_{\text{root}}|} I(Y_{\text{right}}) \tag{2.41}$$

Several metrics can be chosen as the impurity function $I(\cdot)$, such as the Gini impurity, the Shannon entropy or the Poisson loss (). In this study we utilize the variance of each output column, averaged over all outputs (**??**). Since the prediction values we return, resulting from the prototype function, are the column averages of a leaf's partition $Y_{\text{partition}}$ of the training labels ($Y_{\text{partition}}^{\langle i\rangle[j]}$), the column variances correspond to the *mean squared error* (MSE) for the training data in the node, as if the node holding $Y_{\text{partition}}$ were to become a leaf.

$$I_{\text{MSE}}(Y) = (Y^{[ij]} - Y^{\langle i\rangle[j]})^{2\langle ij\rangle} = Y^{2\langle ij\rangle} - Y^{\langle i\rangle 2\langle j\rangle} \tag{2.42}$$

Also notice that I_{MSE} is equivalent to the Gini impurity if Y contains only binary values. That can be quickly shown by noticing that $Y_{\text{bin}}^{[ij]2} = Y_{\text{bin}}^{[ij]}$ for binary labels, so that $Y_{\text{bin}}^{2\langle i\rangle[j]} = Y_{\text{bin}}^{\langle i\rangle[j]} \equiv p^{[j]}$ which yields **??** and culminates in the usual form of the average of Gini impurities across all outputs.

$$I_{\text{MSE}}(Y_{\text{bin}}) = Y_{\text{bin}}^{2\langle ij\rangle} - Y_{\text{bin}}^{\langle i\rangle 2\langle j\rangle} = (p^{[j]} - p^{\langle j\rangle 2})^{\langle j\rangle} = [p^{[j]}(1 - p^{[j]})]^{\langle j\rangle} = I_{\text{Gini}}(Y) \tag{2.43}$$

Referring back to bipartite scenarios, we now describe how intuitive adaptations of the decision tree algorithm are developed to deal with interaction data.

2.3.5 Bipartite decision trees

As presented by **??**, pliakos2018global introduces an ingenious strategy to build a decision tree directly on bipartite-formatted datasets. The main concept behind it borrows the idea of the SLMO approach described in **??**, in that the training procedure is divided into two steps, separately considering each axis of the bipartite data. However, instead of training two separate models as in SLMO, two candidate versions of each node are generated, one for each axis, and the best between both is selected to integrate the final model.

Formally speaking, given a general procedure `FindSplit` for finding a split threshold in a traditional multioutput decision tree (see **??**), the PBCT algorithm (**?**) consists of applying `FindSplit` twice at each node: once over X_1 and Y, in which case each Y *column* is considered a different output; and once over X_2 and $Y^\intercal$, so that each Y *row* is now interpreted as an output. Finally, the best overall split is chosen between the best horizontal split and the best vertical split, according to the quality criterion of **??**.

Although each node locally performs the partitioning search on each axis, the resulting tree in its entirety is termed *Global MultiOutput* (GMO) by the original authors, and its training procedure corresponds to using `FindSplit` = **??** in **??**.

Interestingly, contrary to the other adaptation approaches of traditional machine learning algorithms (**??**), the PBCT procedure does not require using multiple monopartite estimators and training steps (such as SLMO, **??**) or converting bipartite datasets to the memory-expensive global format such as SGSO (**??**). Nevertheless, considering each row or column of Y as a separate output results that growing a PBCT still occurs in the same asymptotic time complexity of a decision tree under the SGSO adaptation (derived in **??**).

Remarkably, we show that leveraging the single-output assumption (**??**) in the tree impurity calculation can simplify the bipartite local split searching procedure introduced by pliakos2018global, yielding a new PBCT-based algorithm with expressive gains in training time efficiency. A formal description of our algorithm proposal is now presented.

2.3.6 Bipartite global single-output trees

Notice that the split rules as defined by **??** are agnostic to the specific arrangement of the bipartite data, being seamlessly applicable to the (X_{SGSO}, Y_{SGSO}) format employed by the SGSO adaptation (**??**) or directly to the X_1, X_2 and Y matrices. For a decision tree growing under the SGSO framework, the impurity of each node, calculated as $I(Y_{\text{SGSO}})$, in many cases could be easily translated to the bipartite format. **??** describes such a translation of the MSE impurity presented by **??**.

$$I_{\text{MSE}}(Y_{\text{SGSO}}) = Y_{\text{SGSO}}^{2\langle ij\rangle} - Y_{\text{SGSO}}^{\langle i\rangle 2\langle j\rangle} = Y^{2\langle ij\rangle} - Y^{\langle ij\rangle^2} \tag{2.44}$$

```
Función FindBipartiteSplit($X$, $Y$)
    Input: A partition of the bipartite training data in a given node. $X$ encodes one
           design matrix for each axis, $X_1$ and $X_2$.
    Output: The highest quality score $Q^*$ found among all splits evaluated in both row
            and column directions, with its corresponding feature column $f^*$ and
            threshold value $t^*$.
 1  if adapterStrategy is GSO (??) then
        // Build $Y$ proxies $\tilde{Y}_1$ and $\tilde{Y}_2$ (??)
 2      $\tilde{Y}_1 \leftarrow Y^{[\cdot]\langle j\rangle}$;
 3      $\tilde{Y}_2 \leftarrow Y^{\langle i\rangle[\cdot]}$;
 4  else
        // Using GMO strategy, no proxies are used
 5      $\tilde{Y}_1 \leftarrow Y$;
 6      $\tilde{Y}_2 \leftarrow Y^\intercal$;
 7  end
    // Generate a split in each axis. Get each split's
       position, feature and quality score
 8  $Q_r^*, f_r^*, t_r^* \leftarrow$ FindSplit($X_1$, $\tilde{Y}_1$);
 9  $Q_c^*, f_c^*, t_c^* \leftarrow$ FindSplit($X_2$, $\tilde{Y}_2$);
10  if $Q_r^* > Q_c^*$ then
11      return $Q_r^*$, $f_r^*$, $t_r^*$
12  else
        // $f_c^*$ value lets clear its $X_2$ ownership
13      return $Q_c^*$, $f_c^*$, $t_c^*$
14  end
```

If we then define

$$I_{\text{GMSE}}(Y) \equiv I_{\text{MSE}}(Y_{\text{SGSO}}) = Y^{2\langle ij\rangle} - Y^{\langle ij\rangle^2}, \tag{2.45}$$

the exact same SGSO-adapted decision tree, with its node structure and split rules, can be grown by applying the PBCT procedure (**??**) with the I_{GMSE} impurity instead of the original I_{MSE}. This again follows from the generality of the split rules defined in **??**, invariant under the SGSO data rearrangement (**??**).

Notably, this new impurity poses a simple yet sound advantage over the original. Having only total averages in **??**, always involving both i and j indices simultaneously, exempts the training function from storing column-wise label averages during split search, enabling us to pre-compute averages of each row and column of Y_{node} and to iterate over one-dimensional $\tilde{Y}_{\text{node}}$ proxies (**??**) instead of the bi-dimensional matrix when evaluating splits (**??** of **??**). This property can be explored to build a more efficient training procedure for bipartite GSO decision trees, as demonstrated in the asymptotic complexity analysis developed in **??**.

$$\begin{aligned}\tilde{Y}_1^{[i]} &= Y^{[i]\langle j\rangle}\\ \tilde{Y}_2^{[j]} &= Y^{\langle i\rangle[j]}\end{aligned} \tag{2.46}$$

2.3.7 Prototype functions of bipartite decision trees

When a leaf is reached during the prediction step of a decision tree, the `prototype` function is called to determine the output value to be returned (**??** of **??**). With monopartite datasets, the `prototype` function of traditional decision trees most often simply returns, for each output, the average label of the leaf's partition (**??**) (**?**).

$$\texttt{prototype}\,(Y_{leaf})^{\,[j]} = Y_{\text{leaf}}^{\langle i\rangle[j]} \tag{2.47}$$

As an extension, the most natural approach to use on single-output bipartite trees is the analogous average of the whole partition of the interaction matrix (**??**).

$$\texttt{prototype}\,(Y_{leaf}) = Y_{\text{SGSO, leaf}}^{\langle i\rangle[1]} = Y_{\text{leaf}}^{\langle ij\rangle} \tag{2.48}$$

Nevertheless, some considerations are possible when dealing with bipartite data, since there are cases in which one of the entities of the interaction being predicted is already known from the training set. As introduced by pliakos2018global, if a row or column instance is in the training set, we have the option of averaging only the column or row (respectively) of Y_{leaf} corresponding to its known outputs. Specifically, when predicting the interaction between a sample pair $x_{1,\,\text{new}}$ and $x_{2,\,\text{new}}$, we can set `prototype` as in **??**.

$$\texttt{prototype}\,(Y_{leaf}) = \begin{cases} Y_{\text{leaf}}^{[k]\langle j\rangle} & \text{if } \exists\ k \mid x_{1,\,\text{new}} = X_{1,\,\text{leaf}}^{[k\cdot]}\\ Y_{\text{leaf}}^{\langle i\rangle[k]} & \text{if } \exists\ k \mid x_{2,\,\text{new}} = X_{2,\,\text{leaf}}^{[k\cdot]}\\ Y_{\text{leaf}}^{\langle ij\rangle} & \text{otherwise.}\end{cases} \tag{2.49}$$

A drawback of this approach, especially when working with very imbalanced interaction matrices and sufficiently small leaf partitions, is a possibly greater susceptibility to random fluctuations, since the label averages in the prediction step are taken over a much smaller sample size (a single row or column of Y_{leaf} instead of the whole Y_{leaf}). Given we are predominantly working with similarity scores as sample attributes, we propose an intermediate approach: to weight the rows and columns of Y_{leaf} by the similarity values in the form $w_{s1}^{[i]} \equiv \text{similarity}(x_1^{[i]},\, X_1^{[i\cdot]})$ between x_{new} and the training samples in the leaf node (**??**).

$$\texttt{prototype}\,(Y_{leaf}) = \frac{\sum_{i\in\text{leaf}} w_{s1}^{[i]} Y_{\text{leaf}}^{[i]\langle j\rangle}}{2\sum_{i\in\text{leaf}} w_{s1}^{[i]}} + \frac{\sum_{j\in\text{leaf}} w_{s2}^{[j]} Y_{\text{leaf}}^{\langle i\rangle[j]}}{2\sum_{j\in\text{leaf}} w_{s2}^{[j]}} \tag{2.50}$$

Since we are dealing with precomputed pairwise similarities, X_1 and X_2 are square matrices in which $X_a^{[i_1 i_2]} = \text{similarity}(X_a^{[i_1\cdot]},\, X_a^{[i_2\cdot]})$. We explore three different cases:

1. $w_s = x^{[i]}$
2. $w_s = (x^{[i]})^2$
3. $w_s = e^{x^{[i]}}$

2.3.8 Asymptotic complexity analysis

From the algorithm description, one can infer that **??**'s complexity will be given by

$$O(alg: \mathit{find\_best\_split}) = O(\tilde{n}_f S(|Y|_i) + \tilde{n}_f |Y|_i |Y|_j) = O(\tilde{n}_f |Y|_i \log |Y|_i + \tilde{n}_f |Y|_i |Y|_j) = O(\tilde{n}_f |Y|_i (\log |Y|_i \tag{2.51}$$

where $\tilde{n}_f$ is the number of features being considered in each node and $S(n)$ is the complexity of the chosen sorting algorithm of when operating on n values. $\tilde{n}_f$ is a hyperparameter of random forests (**??**) usually defined as a function of the total number of features $|X|_j$. We further employ $\tilde{n}_{f1}$ and $\tilde{n}_{f2}$ to refer to the number of features to be selected from X_1 and X_2, respectively, in each node.

The most effective sorting algorithms currently known for real-valued data, such as Quick Sort or Merge Sort (), have $O(n \log n)$ asymptotic complexity. However, since the sorting of multiple subsets of the same X values will be performed, it is often effective to spend $O(|X|_j |X|_i \log |X|_i)$ time previously obtaining ranks for each column of X and for each axis, so that subsequent partition sorting steps can be performed in linear time with integer-specific algorithms such as Radix Sort (). That said, considering $S(n) = O(n \log n)$ does not affect $O($**??**$)$ under the assumption $|Y|_j >> \log |Y|_i$, in which case the second term in the result of **??** dominates in the asymptotic regime.

When applied on X_{SGSO} and Y_{SGSO} as in the naive GSO approach, we have **??**.

$$O(alg: \mathit{find\_best\_split}) = O((\tilde{n}_{f1} + \tilde{n}_{f2}) |Y_{\text{SGSO}}|_i | (\log Y_{\text{SGSO}}|_i + |Y_{\text{SGSO}}|_j)) = O((\tilde{n}_{f1} + \tilde{n}_{f2}) |Y_{\text{SGSO}}|_i | \log Y_{\text{SGSO}}|_i) \tag{2.52}$$

For **??** employing the GMO approach (the original PBCT procedure (**?**)), the time complexity is given by **??**, which renders it equivalent to the SGSO strategy. $\tilde{n}_{f1}$ and $\tilde{n}_{f2}$ stand for the numbers of row and column features to be drawn at each node, respectively.

$$= O(\tilde{n}_{f1} |Y|_i (\log |Y|_i + |Y|_j) + \tilde{n}_{f2} |Y|_j (\log |Y|_j + |Y|_i)) = O((\tilde{n}_{f1} + \tilde{n}_{f2}) |Y|_i ||Y|_j) \tag{2.53}$$

When considering the BGSO approach however, $\tilde{Y}_1$ and $\tilde{Y}_2$ column vectors are used instead of Y, effectively eliminating the `for` loops in **??** and **??** of **??**. Considering that building the $\tilde{Y}$ proxies takes $O(|Y|_i |Y|_j)$, the split search procedure in this strategy has its complexity described by **??**.

$$O(alg: \mathit{find\_bipartite\_split}_{\text{BGSO}}) = O(|Y|_i |Y|_j + \tilde{n}_{f1} |\tilde{Y}_1|_i \log |\tilde{Y}_1|_i + \tilde{n}_{f2} |\tilde{Y}_2|_i \log |\tilde{Y}_2|_i) = O(|Y|_i |Y|_j + \tilde{n}_{f1} |Y|_i \tag{2.54}$$

For the whole tree-building process, considering $|Y|_i \propto |Y|_j = n$ and given any constant $k \in \mathbb{N}$ so that $O(\text{FindSplit}) = n^k$ we analyze the best case of a balanced decision tree, where the number of samples in each node ($|Y| = n^2$) halves with each level added:

$$|Y_{\text{child node}}| \approx |Y_{\text{parent node}}/2| = n^2_{\text{parent node}}/2 \tag{2.55}$$

As a consequence, the number of nodes in the $(l+1)$-th level is given by 2^l, while the number of samples to process at each node is estimated by $n^2/2^l$ and the total number of levels is given by $L = \log_2 n^2 = 2\log_2 n$.

$$\begin{aligned} O(alg : buildtree) &= O\left(\sum_{l=0}^{L-1} 2^l \text{FindSplit}(n^2/2^l)\right) = O\left(\sum_{l=0}^{L-1} 2^l n^{2k}/2^{kl}\right) = O\left(n^{2k}\sum_{l=0}^{L-1} 2^{l(1-k)}\right) \\ &= \begin{cases} O(n^{2(k+1)}) & \text{if } k < 1 \\ O(n^2 \log n) & \text{if } k = 1 \\ O(n^{2k}) & \text{if } k > 1 \end{cases} \end{aligned} \tag{2.56}$$

We can observe that our proposed optimization for the GSO strategy reduces the tree-building complexity by a factor of

$$\frac{O(\text{BGSO})}{O(\text{GMO})} = \frac{\log n + \tilde{f}}{\tilde{f}\log n} = \frac{1}{\tilde{f}} + \frac{1}{\log n} \tag{2.57}$$

in comparison to GMO, a major improvement especially for datasets with a large number of features. Remarkably, when kernel or similarity matrices are used, as the datasets explored in this work do, or in general when $\tilde{f} \propto n$, BGSO is $\log n$ times faster than GMO in the asymptotic regime, yielding unseen scalability of decision trees for interaction prediction problems.

Regarding the performance of **??** as a substitute for **??**, although considerable amounts of operations are saved, the procedure for generating a split point for each feature column still requires finding minimum and maximum values, occurring in the same linear time complexity as the greedy approach. As a result, the asymptotic computational time needed by the algorithm is expected to grow with no different ratio relative to the number of samples, not resulting in any improvements in complexity relative to **??**.

n$_{f1}$

2.3.9 Decision forests

While single decision trees are valuable tools for comprehending the learning problem at hand, they often fall short in effectively modeling intricate relationships within the data, displaying limited generalization capabilities and high susceptibility to overfitting (). Their most

Strategy	Split search	If Y is square	Tree building	$\tilde{n}_f \propto n$
SGSO	$O((\tilde{n}_{f1} + \tilde{n}_{f2})n_1 n_2(\log n_1 + \log n_2))$	$O(\tilde{n}_f n^2 \log n)$	$O(\tilde{n}_f n^2 \log n)$	$O(n^3 \log n)$
GMO	$O(\tilde{n}_{f1} n_1 \lvert(\log n_1 + n_2) + \tilde{n}_{f2} n_2 \lvert(\log n_2 + n_1))$	$O(\tilde{n}_f n^2)$	$O(\tilde{n}_f n^2 \log n)$	$O(n^3 \log n)$
BGSO	$O(n_1 n_2 + \tilde{n}_{f1} n_1 \log n_1 + \tilde{n}_{f2} n_2 \log n_2)$	$O(n^2 + \tilde{n}_f n \log n)$	$O(\tilde{n}_f n^2)$	$O(n^3)$

Table 1 – Comparison between asymptotic time complexities of bipartite decision tree-building procedures. n_1 and n_2 respectively designate the number of samples in X_1 and X_2, while n is defined so that $n_1 \propto n_2 \propto n$. Similarly, $\tilde{n}_{f1}$ and $\tilde{n}_{f2}$ represent the number of features from each axis to be considered for split search in each node, while $\tilde{n}_f$ is used to illustrate the cases where $\tilde{n}_{f1} \propto \tilde{n}_{f2} \propto \tilde{n}_f$. The last column refers to the case where the number of features considered in each node is proportional to the number of samples in each axis ($n_f \propto n$). This scenario could arise, for instance, if one is dealing with pairwise similarities or kernel matrices as the model's input data.

significant impact on machine learning applications is observed when committees of such models are utilized instead, in which the final output values are most commonly computed by averaging or summing the predictions of all trees.

These compositions of estimators are usually referred to as *ensembles*. Being studied in the context of machine learning since the 1970s, they are based on the intuitive idea that combining multiple opinions from a diverse set of experts frequently results in better decision taking.

In fact, it has been extensively demonstrated both empirically and theoretically that the predictive perfomance a group of learners always surpasses that of its individual components if and only if the individual estimators are sufficiently accurate and diverse (**?**).Importantly, requiring diversity means that the individual estimators must ideally commit errors on different instances for the composition to succeed (**?**).

dietterich2000ensemble,polikar2006ensemble describe three ways in which combining diverse estimators could benefit the ensemble's performance.

1. **Statistical:** Building each estimator can be seen as finding a hipothesis that explains the prediction problem as well as possible. If multiple different hypotheses are found, the likelihood that at least one of them is close to the true underlying function is increased. Furthermore, averaging multiple hypotheses can contribute to alleviate the influence of each hipothesis' variance, resulting in a more accurate approximation and reduced propensity to overfitting.

2. **Computational:** Finding the globally optimal decision tree or neural network is known tobe an NP-complete problem (). Thus, finding and combining multiple approximate hypotheses, that may represent local maxima of the objetive function, is often much cheaper than expending time on the search for a global solution.

3. **Representational:** Sometimes, a single estimator is not complex enough to represent the

intricacies of the probem at hand. However, combination of models can be utilized to enhance the representational power of the ensemble, building a more general decision function as a combination of the simpler decision boundaries of each estimator.

The balance between diversity and individual strength is thus of central role when designing ensemble models. Both these properties act in the direction of increasing the final ensemble performance, so that mechanisms of introducing model heterogeneity, and not only strategies for boosting individual gains, must be carefully considered.

Even if hampering single component performance to some extent, promoting diversity between components, so to reduce as far as possible the correlation between their outputs, is often proved to result in performance improvements for the whole ensemble (**?**). This is the core idea of decision forests. Even simple estimators such as decision trees, traditionally prone to overfitting, can be leveraged to compose powerful ensemble models, generically called *decision forests*, as long as proper diversification is employed. We briefly describe some popular strategies for introducing heterogeneity in the decision tree growing procedure.

1. **Instance sampling:** A new version of the training set is built by randomly drawing instances from it, with or without replacement (selecting rows of X). If drawing with replacement, one can optionally draw the same number of instances as the original set, a procedure known as bootstrapping.

2. **Feature sampling:** A random subset of features is selected to be used (selecting columns of X). The number of features $\tilde{n}_f$ selected in each node is commonly defined as a function of the total number of features $|X|_j$. Usual choices are $\tilde{n}_f = \lceil \sqrt{|X|_j} \rceil$ or $\tilde{n}_f = \lceil \log |X|_j \rceil$.

3. **Split threshold randomization:** Instead of searching for the best split threshold for each feature as in **??**, a random threshold value is drawn between the minimum and maximum values of each feature column.

Both instance and feature undersampling can be performed node-wise, occurring before the split search procedure of each node, or tree-wise, occurring once for each tree in the ensemble. Both can also be performed with or without replacement, but notice that sampling features with replacement only makes sense if randomization of split threshold is also employed. In such a case, a random candidate split threshold is selected for each repetition of a feature, whereas the same split threshold would be selected for all duplicates if the greedy approach was used, spending more time with no different result than if omitting the repeated features.

The concept of sampling instances with replacement to create a different training set for each estimator in the ensemble was first proposed by breiman1996bagging under the name of bootstrap aggregation or *bagging*.

The idea of selecting a subset of features was introduced independently by amit1997shape and tinkamho1998random. While amit1997shape explored feature sampling as a remedy for a shape recognition problem with impeditively large sample sets, tinkamho1998random was mainly focused on overfitting-prevention for decision forests.

It was breiman2001random who first combined the two ideas, proposing and greatly popularizing the Random Forest algorithm, with over 115 thousand citations according to Google Scholar as of september 2023.

geurts2006extremely later introduced the randomized split threshold concept, presenting the Extremely Randomized Trees algorithm (Extra-Trees) and showcasing its competitive prediction scores and clear superiority in terms of training speed compared to bagging and Random Forests.

The two main ensemble-building strategies we explore in the current work can now be defined as follows:

- **Random Forests (RF):** Tree-wise instance sampling with replacement and node-wise feature undersampling without replacement are employed.
- **Extremely Randomized Trees (ERT):** Split threshold randomization and node-wise feature undersampling without replacement are employed. No resampling of instances is performed in the original proposal.

One of the notable features of decision forests that contribute to their expressive popularity is their low need for data preprocessing, since the order of the values among each feature is the main factor for determining the best split points, and not their scale. Additonally, the relative scaling of different features is not a concern, since the split search procedure is performed separately for each feature. These factors, combined with their strikingly small set of hyperparameters, crown decision forests as remarkable "plug-and-play", easily configurable, models, especially suited for unstructured tabular data (**?**).

We refer the reader to sagi2018ensemble and fawagreh2014random for a more in-depth description of prominent decision forest strategies and previous work in the field. amasyali2011comparison provides an experimental comparison of the most popular ensemble methods.

Regarding interaction problems, schrynemackers2015classifying explores the use of decision forests under the standard global multi-output and local single-output adaptations presented by **??**.

However, the same tree-diversification and forest building techniques discussed in this section can also be applied to bipartite decision trees, with very small modifications regarding the data sampling procedures: instance sampling and feature sampling must now occurr on both domains of the interaction dataset. Under this assumption, the adptation to bipartite problems can thus occur at the tree level, not at the ensemble level as would be the case with SLMO and SGSO strategies.

(**?**) explores these ideas, using bipartite global multi-output decision trees to build both Random Forests and Extra-Trees ensembles. We hereafter refer to these forests as Bipartite Random Forests (BRF) and Bipartite Extra-Trees (BXT), respectively. Their study suggests superior performance of BXT in comparison to SLMO- and SGSO-adapted forests, as well as in comparison to previously proposed algorithms, although no significance level or test statistic is used to describe the comparisons.

The authors later extended their work to include a self-learning step for predicting drug-target interactions (**?**). In that study, their GMO BXT model is not directly built upon the bipartite dataset but rather on a reconstructed version of the interaction matrix obtained through applying the Neighborhood-Regularized Logistic Matrix Factorization technique (NRLMF, liu2016neighborho

In the present work, we refine such comparisons and inroduce ensembles of our newly proposed bipartite global single-output decision trees (**??**), demonstrating competitive prediction scores under an asymptotically more efficient training framework.

2.3.10 Implementation details

2.3.11 Incorporating semi-supervision in decision trees

The tree algorithms presented up until this point disregard the positive-unlabeled characteristic of our datasets of interest, either ignoring the possibility of abscence of labels in the training set, or, equivalently, considering all non-occurring interactions as unknown interactions. In other words, negative labels represent both the interactions known to not occur and the interactions about which no information is available.

Therefore, the machine learning paradigm under which the discussed decision trees are proposed is still *supervised*, in the sense that all training instances are associated with a label (for each sample in X there is a corresponding expected output in Y).

As several previous authors (**?**, **?**), we argue that taking into account the partial availability of information that is often intrinsic to interaction prediction problems could thus improve prediction performance of bipartite models and reduce the necessity for labeled data.

When the learning problem at hand does not involve predefined labels, the taks is termed *unsupervised*. The most common example is clustering problems, where the goal is to group similar instances together based solely on their numerical attributes, without any *a priori* information regarding hypothetical classes to which each instance could belong. In such problems, only the X matrix is utilized, there is not an associated label matrix Y.

In summary, while supervised learning problems are concerned with the relationship between X and Y, aiming to model how a label is determined by the descriptive features of each instance, unsupervised learning problems target the relationship between instances themselves (between rows of X), describing how similar they are and how they can be taxonomically organized.

Some problems, however, lie in the intersection of the supervised-unsupervised spectrum. They usually arise when the available labels are scarce or unreliable, so that both supervised and unsupervised objectives are of interest. Take for instance the case of an image gallery phone applicative capable of grouping all photos of a person into a folder with their name. While the name must be asked once to the user, subsequent photos of the same person are automatically grouped under the same directory. The algorithm is both concerned with labeling each photo with the correct name and with grouping similar photos together. Another example can occur when diagnostic data of a large collection of patients is employed to build an automated diagnosis system to assess multiple diseases simultaneously. Almost certainly, not all labels for the occurrence of each medical condition will be available for all patients, so that the algorithm must be able to deal with considerable amounts of missing data. One possible strategy is to infer missing labels from the labels of similar instances, so that the concept of grouping is again into play. Still in the same example, the diagnosis for each disease may be presented in different (unknown) degrees of certainty, resulting from diagnostic procedures of different natures. In this case, an even more refined strategy can be employed, in which the amount of importance given by the model to correctly classify a given label may be adjusted to allow prioritizing information resulting from clustering rather than the label itself, essentially regulating the relative importance between the supervised and unsupervised objectives.

While in the first example the known labels are usually much more scarce than in the diagnostics example, both problems deal with the same underlying issue: the lack of reliable labels for all instances. As such, both label inference (that characterizes supervised problems) and clustering (that characterizes unsupervised problems) are closely intertwined as the model's objective. The goal is to simultaneously learn to correctly classify the instances with known labels in the training set while also considering similarities to infer the labels of the unlabeled instances (or to rectify labels on which one cannot fully rely). The class of such hybrid learning paradigms is called *semi-supervised learning* ().

Since missing labels are a defining characteristic of interaction prediction tasks, it is commonly suggested (**?**, **?**) that applying semi-supervised concepts to our problems could significantly improve the performance of bipartite models.

For decision trees specifically, there are straightforward ways to incorporate semi-supervised assumptions. Notice that the growing of decision trees unavoidably induces groupings of samples in the training set. Specifically, the structure of each tree represents a hierarchical clustering of the training samples, in which each tree node represents a partition of the training set composed by the training instances that reach that node. However, this clustering procedure is usually preformed under the objective of grouping instances with similar *labels*, which not necessarily means that instances in the same group will have similar *features*. This results from the definition of the impurity function I governing the split search procedure: I is usually chosen as to minimize the divergence of labels within each partition (see **??**). As such, I commonly depends on Y_{node} alone. With that in mind, decision trees can be naturally adapted to unsupervised or semi-supervised tasks by redefining the impurity function to consider the feature matrix X_{node} instead of only the label matrix Y_{node}.

In a purely unsupervised context, an example would be to utilize the average of column variances in X_{node} as the impurity function (**??**). This would result in a tree that groups instances with similar features together, regardless of their labels. This idea is the main concept behind the CLUS algorithm (**?**).

$$I_{\text{unsupervised MSE}}(X) = \text{MSE}(X) = \left(X^{[ij]} - X^{\langle i\rangle[j]}\right)^{\langle i\rangle 2\langle j\rangle} \tag{2.58}$$

To address semi-supervised scenarios, we can consider both supervised and unsupervised objectives simultaneously. This can be done by using a linear combination of unsupervised and supervised impurities to guide the split selection, taking into account both the similarities between features and between labels to build a semi-supervised decision tree. **??** defines such a hybrid impurity function.

$$I_{ss}(X_{\text{node}}, Y_{\text{node}}) = (1-\sigma)\frac{I_u(X_{\text{node}})}{I_u(X_{\text{root}})} + \sigma\frac{I_s(Y_{\text{node}})}{I_s(Y_{\text{root}})} \tag{2.59}$$

We divide each term by the corresponding impurities on the root node, thus calculated over the entire training set. The reason is to avoid the influence of possible differences in scale between the values in X and in Y. It also compensates relative scale differences that could arise when choosing different functions for I_u and I_s. The parameter $\sigma \in [0,1]$, that we call supervision balance, weights the relative importance given to each objective, with $\sigma = 0$ corresponding to a fully unsupervised tree and $\sigma = 1$ to a fully supervised tree. Strategies for adjusting σ are discussed in **??**.

For semi-supervised tasks in general, where we have confirmed positive and confirmed negative annotations alongside missing entries, notice how I_s can only be calculated over the non-missing annotations, while I_u can always utilize both labeled and unlabeled instances. Nevertheless, please notice that all confirmed annotations are positive in our present scenario of positive-unlabeled learning. Thus, we must still consider the missing labels as zero entries for the calculation of I_s. Even so, we argue that decreasing the relative importance of the supervised objective can compensate the label uncertainty and improve generalization. The idea is to encourage the tree to appreciate the structures in the feature space. For example, it should select splits that isolate very compact groups of instances, even if the labels in a given group are not satisfactorily homogeneous.

A caveat of using a semi-supervised impurity as in **??** is that trees could continue to find splits even if all instances in a given node have the same label. This is because we could keep reducing the unsupervised impurity by further splitting even if the supervised impurity is already zero. However, the output value of each node is still calculated over Y_{node} alone, so all nodes descending from a homogeneous node would yield the same output. We avoid such redundant splits by forcefully stopping the split search for a node when $I_s(Y_{\text{node}}) = 0$.

Another important notice is that features must be normalized before training the tree: the impurity function I_u (**??**) is most often sensitive to the relative scale of the different columns of X. This is normally not a requirement for decision trees, since the split search procedure is performed separately for each feature and usually only depends on the order of the values, not on their specific magnitude.

2.3.11.1 Heuristics for σ

Determining the ideal value of σ is not a trivial task. In fact, it is not even clear whether a single constant σ for the whole tree is enough or it should be adjusted for each node.

alves2023semisupervised argues that the unsupervised impurity should be more important for nodes with a larger number of unlabeled instances, since the supervised information would be more likely to be unreliable. Thus, they propose updating σ according to the label density of each node, as defined by **??**.

$$\sigma = 0.1 + 0.9 \cdot Y_{\text{node}}^{\langle ij \rangle} \tag{2.60}$$

However, we must recall that the prototype value of each node is only dependent on its partition of the label matrix. Therefore, if Y_{node} is close to being homogeneous, a new split is unable to cause a big overall change in the outputs for the instances involved. Specifically, the most drastic prototype change possible occurs if we perfectly separate the instances with positive labels from the instances with negative labels. When the node partition is very imbalanced, even this ideal separation will only greatly affect the very few instances with the minority label. Thus, using the heuristic of **??** prioritizes the unsupervised objective only in cases where no significant change in the output values is expected. Additionally, few new splits are possible in near-homogeneuous nodes, and we are more likely to achieve label homogeneity (and thus stop the split search) right after we achieve the cases in which I_u is highly prioritized. In summary, the heuristic of **??** is likely to undermine the effect of the unsupervised objective.

To test this hypothesis, we propose another heuristic for σ that prioritizes the unsupervised objective in the earliest stages of the tree growing process and the supervised objective in nodes closer to the leaves (**??**).

$$\sigma = 1 - \frac{|Y_{\text{node}}|}{|Y_{\text{root}}|} \tag{2.61}$$

This way, we take advantage of the unsupervised impurity prior to when the label partitions are already homogeneous. Essentially, we start by performing clustering in the feature space, and then gradually move towards separating instances based on their labels. This process should be similar to a two step procedure: we first identify large structures in the feature space (when a large number of instances is still available) and then apply the label clustering separately to each of these structures. The difference is that each of our semi-supervised trees represents a gradual transition between the two steps.

We also speculate that the observed benefit of a dynamic σ could emerge from the diversity it promotes in tree learners. As discussed in **??**, the diversity of the individual estimators is a key factor for the success of ensemble models. In this sense, ensuring variety of the σ parameter could even be more important than estimating its "correct" value for each partition. Therefore, we also evaluate the strategy of selecting a random σ at each node, drawn from a uniform distribution in the interval $[0, 1]$ (**??**).

$$\sigma \sim \mathcal{U}(0, 1) \tag{2.62}$$

2.3.11.2 Unsupervised impurities

The initial proposal of a semi-supervised impurity function (**?**) employed the variance of the feature matrix X as the unsupervised term (**??**). The main concern with this strategy is that it considerably slows the training procedure of bipartite trees. This results from the fact that the number of features to be considered in each node is the same ...

Exploiting the fact that we are working with kernel matrices as input features, we propose an efficient unsupervised impurity function based on the mean pairwise distance between the samples in the tree node (equation **??**). s_{node} denotes the set of indices representing the samples in the tree node, and since distances from samples outside the node are disregarded, the number of operations required drops more steeply with respect to the node size in comparison to $I_{\text{MSE GMO}}$ or similar impurities, which is especially beneficial in our case of large X matrices.

$$I_{\text{MeanDistance}}(X) = \frac{1}{|\mathrm{s}_{\text{node}}|} \sum_{j \in \mathrm{s}_{\text{node}}} \sum_{i \in \mathrm{s}_{\text{node}}} (1 - X^{[ij]}) \tag{2.63}$$

2.4 Assessing the performance of bipartite models

2.4.1 Datasets

We gathered ten publicly available interaction datasets to evaluate the performance of the proposed models. Quantitative information about the datasets is presented in table 2, and more detailed descriptions are provided below.

These datasets comprise drug-protein interactions for four distinct classes of proteins: enzymes, GPCRs, ion channels, and nuclear receptors, respectively. Drug similarities were computed using the SIMCOMP metric, while protein similarities were computed as normalized scores of Smith-Waterman pairwise alignments (**?**).

The datasets represent interactions between genes and transcription factors in *E. coli* and *S. cerevisiae*, respectively. Gene and TF features are initially composed of experimentally measured expression levels and, in SRN, gene motif features (**?**, **?**). We compute the RBF kernel of such values to obtain the final similarity matrices.

The DAVIS dataset contains experimentally measured drug-kinase dissociation constants (**?**). The dataset was binarized by considering interactions with dissociation constants $\leq 30nM$ as the positive ones, as suggested by (**?**). Drug similarities were computed using the Extended Connectivity Fingerprints (ECFP4) (**?**, **?**) while protein similarities were taken as the normalized Smith-Waterman score (**?**, **?**).

The KIBA dataset was initially built by **??**(**?**) and contains experimentally verified affinity scores between kinase and kinase inhibitors.

(**?**) further processed the dataset by removing all drugs and targets with less than 10 observations. In alignment with (**?**, **?**), we consider positive interactions as those with log_10 KIBA-scores ≤ 3.0 to reframe the task as binary classification.

The utilized version of the dataset with corresponding amino acid sequences and SMILES representations were provided by (**?**). From them, we generated the protein similarity matrix using the same procedure employed in the preprocessing of NPInter proteins. The drug similarities were computed similarly to how (**?**) processed the DAVIS dataset, using the Tanimoto distances of ECFP4 fingerprints (**?**, **?**). The Python library `rdkit` (**?**) was used to this calculation.

The mirTarBase dataset contains experimentally validated microRNA-messengerRNA interactions. MicroRNA sequences were obtained from miRBase (**?**) while transcript sequences were obtained from GENCODE (**?**). The longest transcript for each gene was selected and the 3' UTR exonic sequences were recovered from the genome and annotation files provided by GENCODE. The similarity matrices were then built from the normalized Smith-Waterman (**?**) alignment scores among microRNAs and among the genes' 3' UTRs. The alignments were performed using the BLASTN substitution matrix and no gap penalty, with the help of the Biopython package (**?**).

Each miRNA was required to have at least 10 interactions in the dataset, and each gene was required to have at least 100 interactions.

Interactions between long non-coding RNAs (lncRNA) and proteins were recovered from NPInter (**?**, **?**). The lncRNA sequences were obtained from NONCODE (**?**) and the protein sequences were obtained from UniProt (**?**). The similarity matrices were built from the normalized Smith-Waterman (**?**) alignment scores among lncRNAs and among the proteins. Similarly to the preprocessing of mirTarBase, we leveraged the Biopython package (**?**) to perform the alignments. using the BLASTN and BLOSUM62 substitution matrices for the lncRNA and protein alignments, respectively, and no gap penalty in both cases.

Each lncRNA was required to interact with 50 proteins or more in the dataset, and each protein was required to have at least 2 interactions.

2.4.2 Evaluating bipartite models

2.4.3 Model validation

To evaluate machine learning models, the standard procedure consists of separating a subset of data samples not to be used in the training process. These samples are subsequently inputted to the trained model and its known labels are compared to the model's predictions in order to estimate the algorithm performance. The hold-out samples are collectively called the *test set* while the remaining ones used for model building are called the *training set*.

Since bipartite interaction datasets present two distinct categories of instances and the model's input is a pair of them, one from each group, additionally to a traditional "unknown test set" there are two mixed training/test folds possible: we could test our model performance when predicting interactions between instances from X_1 that are present in the training set and instances from X_2 present in the test set, and vice-versa. Similarly to **??**, we name those settings *LT*, after "learned X_1, test X_2", and *TL*, after "test X_1, learned X_2". The usual cross-validation setting with completely new test pairs is then called *TT*, and the training set could alternatively be called the *LL* set.

In the present work, we make use of an adapted k-fold cross-validation procedure to evaluate our models' performance. With customary datasets formatted as X_{SGSO} and Y_{SGSO}, k-fold cross-validation consists in equally and randomly dividing both X_{SGSO} and Y_{SGSO} together in k non-overlapping partitions (or folds). The model is then evaluated k times, each time selecting a fold as the test set and the remaining ones as the training set (Figure **??**).

In the interaction setting though, with a two-dimensional interaction matrix, fold division can be done in each of the two axis, corresponding to each of the two X_a sample groups. Each of the k_1 "axis-folds" of X_1 can be combined with one of the k_2 axis-folds of X_2 to make up a Y fold and split the dataset in the corresponding four LL, LT, TL and TT subsets. If all axis-fold combinations are explored, a k_1 by k_2 two-dimensional cross-validation naturally has a total of $k_1 k_2$ folds.

However, an argument can be made about not sharing axis-folds between Y-folds, to ensure all folds are completely independent and no information is shared between models built on each fold. For instance, if a particular X_1 axis-fold happens by chance to be unrepresentative of the remaining instances in X_1, all k_2 folds that include this axis-fold are expected to yield poor prediction scores. A statistical test comparing two of such score populations then would be biased towards considering those k_2 anomalously distributed points as a significant difference, while in reality they come from a single stochastic event, not k_2 events as could be apparent.

To achieve fold-independence, each fold must be built from a completely different pair of axis-folds, which can be simply done by selecting $k = k_1 = k_2$ and pairing each X_1 axis-fold with a single X_2 axis-fold, yielding a total of k folds, not k^2 as when all axis-fold combinations are used (Figure **??**). While $k_1 \neq k_2$ is still theoretically possible, the total number of folds will always be equal to the least k_a value, and the axis corresponding to the greater k_a would have unexploited axis-folds when creating the test sets.

We refer to the aforementioned two-dimensional cross-validation procedure built from a one-to-one mapping of k X_1 axis-folds to k X_2 axis-folds as k-fold *diagonal* cross-validation.

In order to maximize the amount of training data in each fold, several studies **??** perform LT and TL validation separately from the TT validation, employing 1 by k and k by 1 cross-validation procedures respectively for LT and TL settings. Nevertheless, this requires performing cross-validation three times for each estimator, while TT cross-validation already unavoidably generates LT and TL partitions that could be used for scoring. Furthermore, using separate LT, TL and TT validation procedures hinders score comparison between LT and TT and between TL and TT, since different amounts of training data would be used for validating TT in comparison to validating the partially-learned test sets.

2.4.4 Prediction scoring metrics

This section is concerned with defining the two metrics used throughout this work to evaluate the predictive performance of an estimator and enable comparison between them.

Consider a test set of N interaction labels to be inferred by a classifier (we are not concerned with the shape of Y in this section, and $N = |Y|$). Let the classifier's predictions then be represented by a matrix $\hat{Y}$ of the same shape as Y, with $\hat{Y}^{[ij]}$ being the predicted value for the ground-truth label $Y^{[ij]}$. Since Y and $\hat{Y}$ are both binary matrices, there are four possible outcomes when a prediction is made, traditionally quantified () as follows:

- **True Positives (TP):** the number of positive labels correctly predicted, where both the predicted and actual labels are 1.
$$TP = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 1 \text{ and } \hat{Y}^{[ij]} = 1) \tag{2.64}$$
- **True Negatives (TN):** the number of negative labels correctly predicted, where both the predicted and actual labels are 0.
$$TN = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 0 \text{ and } \hat{Y}^{[ij]} = 0) \tag{2.65}$$
- **False Positives (FP):** the number of instances where the predicted label is positive (1), but the actual label is negative (0).
$$FP = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 0 \text{ and } \hat{Y}^{[ij]} = 1) \tag{2.66}$$
- **False Negatives (FN):** the number of instances where the predicted label is negative (0), but the actual label is positive (1).
$$FN = \sum_{i,j} \mathbb{I}(Y^{[ij]} = 1 \text{ and } \hat{Y}^{[ij]} = 0) \tag{2.67}$$

where $\mathbb{I}(A)$ is the indicator function that equals 1 if statement A is true and 0 otherwise:

$$\mathbb{I} = \begin{cases} 1 & \text{if } A \\ 0 & \text{otherwise} \end{cases} \tag{2.68}$$

Notice that the sum of TP, TN, FP and FN is equal to the total number of instances T, and we also define P $= TP + FN$, the total number of *a priori* positive labels in the test set, and N $= TN + FP$, the total number of *a priori* negative labels. The total number of predicted positives is termed PP $= TP + FP$ and the remaining predicted negatives are called PN $= TN + FN$.

Those quantities can be organized in the so-called *confusion matrix*, illustrated by **??**. Naturally, one wants their estimator to maximize TP and TN while minimizing FP and FN.

Most commonly, we do not use these metrics directly, but instead normalize them to the interval [0, 1] in numerous ways. This enables score comparisons across datasets with different numbers of samples and different densities of positive annotations. Below we list the most common normalized scoring metrics for binary classification problems, from which the metrics used in this work are derived.

- **True positive rate (TPR)** or **recall:** the ratio of correctly predicted positive labels to the total number of positive labels.

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{2.69}$$

- **True negative rate (TNR):** the ratio of correctly predicted negative labels to the total number of negative labels.

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{2.70}$$

- **False positive rate (FPR):** the ratio of incorrectly predicted positive labels to the total number of negative labels.

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR \tag{2.71}$$

- **False negative rate (FNR):** the ratio of incorrectly predicted negative labels to the total number of positive labels.

$$FNR = \frac{FN}{P} = \frac{FN}{FN + TP} = 1 - TPR \tag{2.72}$$

- **Precision:** the ratio of correctly predicted positive labels to the total number of predicted positive labels.

$$Pr = \frac{TP}{PP} = \frac{TP}{TP + FP} \tag{2.73}$$

These metrics are better visualized in **??**.

An optimal binary classifier will thus present high TPR, TNR and precision while minimizing FPR and FNR.

Notice that each of these metrics still allows trivial solutions: if a classifier outputs positive labels for all instances irrespectively of the input features, it will achieve perfect TPR, FNR and Pr, but its TNR and FNR will be null. On the other hand, if a negative label is outputted every time, the opposite will happen. Thus, we will consider pairs of these metrics simultaneously, such as TPR and TNR. The selected pair must include at least three of TP, TN, FP, or FN to ensure the whole confusion matrix is taken into account.

Minimizing all the four components of the confusion matrix is not always possible. Most often, the learning algorithms are subject to a tradeoff between the ability to correctly infer positive annotations and the ability to correctly infer negative annotations, that we express as a balance between TPR and TNR. When evaluating models, we must be aware of the relative importances being assigned to each of these tendencies. This choice is highly application-specific. For instance, in the process of diagnosing medical conditions, TPR is often prioritized (), favoring the identification of all true cases at the expense of some misleading positive results. In this case, the cost of missing a positive result is usually far greater than that of a false positive. In other scenarios such as spam email filtering (), TNR might be favored, minimizing the number of legitimate emails marked as spam even if some spam emails go undetected.

Considering this balance across a variety of learning tasks is a challenging factor to be taken into account, especially in cases where the estimators can be easily adjusted to favor each class. Many estimators, such as those employed in the present study, do not directly output a binary label but instead provide us with a continuous decision value (such as a probability of interaction) that additionally depends on a threshold parameter to establish the final predicted classes. Formally, the predicted labels $\hat{Y}$ are obtained from a threshold t applied to the decision values $\tilde{Y}$:

$$\hat{Y}^{[ij]} = \mathbb{I}(\tilde{Y}^{[ij]} > t) \tag{2.74}$$

The selection of t directly affects the propensity to positive or negative outputs, so that a single model can yield multiple different results with varying levels of TPR and TNR depending on the chosen thresholds. A common practice is then to consider TPR and TNR for all t, avoiding the influence of threshold selection on the estimator comparisons.

Notice that, since a finite set of outputs is considered for evaluation (the test set), a finite set of thresholds will cover all possible classification results of a model. These results can be easily displayed in a two-dimensional plot, using a point for each considered threshold so that its corresponding TPR and TNR are each indicated by an axis. Conventionally, the TPR is plotted in the y axis while the FPR values (representing the TNR, $FPR = 1 - TNR$) are presented as x coordinates, which results in the traditional *receiver operating characteristic* (ROC) curve, exemplified by **??**. An ideal threshold of an ideal estimator would then be close to the top-left corner of the plot, where TPR and TNR are both 1. On the other hand, consider a completely random classifier outputting uniformly random values $\hat{Y}$_proba$^{[ij]}$ in the 0-1 interval. We would have $\hat{Y}^{[ij]} = \mathbb{I}(\hat{Y}^{[ij]}_{\text{proba}} > t)$ for a given threshold t. This implies that the number of correctly guessed positive labels is $1-t$ (the probability of yielding 1) times the total number of positive labels: $TP = (1-t)(TP+FN)$, which results in $TPR = 1-t$ from the definition. Similarly, $FPR = TPR = 1-t$, so that the ROC curve of a random classifier is a diagonal line from the bottom-left to the top-right corners.

To summarize a classifier's performance across all thresholds, the area under the ROC curve (AUC ROC) is often employed, with values ranging from 0.5 to 1, where 1 represents a perfect classifier and 0.5 represents a random classifier. Although theoretically possible, values below 0.5 would signify the opposite label is being consistently predicted, most likely indicating misconfiguration of the estimator. If AUROC < 0.5, the result can be easily converted to a value greater than 0.5 by simply inverting the predicted labels (turning 0s into 1s and vice-versa).

Despite considerably common, the use of ROC curves requires additional considerations when dealing with heavily imbalanced classification datasets (where some classes are greatly overrepresented) (**?**, **?**, **?**). For instance,consider the case where $NEG \gg POS$. Since the denominator of TNR is far greater than the denominator of TPR, a change in TN (for example, missing one more negative label) will have a much smaller impact on TNR than a change in TP (missing a positive label) will have on TPR. Specifically, an increase of k in TN will cause an increase in TNR $\frac{NEG}{POS}$ times greater than the increase in TPR caused by the same k increase in TP.

Given that ROC equally considers both metrics, classifiers that are more sensitive to positive labels will arguably be favored over those prioritizing negative outputs. By the same logic, ROC will also be more lenient towards false positives (that decrease TNR) rather than false negatives (that decrease TPR) ().

To address this issue, it is commonly suggested the usage of precision-recall (PR) curves instead (**?**, **?**), where the precision is plotted as the vertical coordinate while the *recall* (another name for TPR) is represented horizontally. We can see the FP term in **??** as the proxy for the true negatives (FP = N - TN). Notice that FP in the definition of precision (**??**) is not divided by the total number of negative labels, as TN is in TNR. Thus, it is argued that AUPR is less likely than AUROC to prioritize the minority class in imbalanced scenarios (**?**, **?**).

We explore these claims in further detail in the following section, formally defining the ROC and PR curves in terms of ideal label probability distributions.

2.4.4.1 Ideal descriptions of AUROC and AUPRC

Consider a general estimator outputting a decision value $s \in \mathbb{R}$ for each input instance. The final class to be assigned is still to be defined by a threshold s^* so that $\hat{Y}^{[ij]} = \mathbb{I}(s^{[ij]} > s^*)$. For the ideal case of an infinite number of test samples, the possible scoring results of the estimator would be fully determined by the two theoretical distributions of s given the true label $Y^{[ij]}$, i.e. the probability density functions $P(s \mid Y^{[ij]} = 1)$ and $P(s \mid Y^{[ij]} = 0)$. Similar to hand2009measuring, we thus define the probability density functions f_k and their corresponding cumulative distribution functions F_k for each of the two classes $k \in \{0, 1\}$:

$$f_0(s) = P(s \mid Y^{[ij]} = 0) \tag{2.75}$$

$$f_1(s) = P(s \mid Y^{[ij]} = 1) \tag{2.76}$$

$$F_0(s) = \int_{-\infty}^{s} f_0(u)\, du \tag{2.77}$$

$$F_1(s) = \int_{-\infty}^{s} f_1(u)\, du \tag{2.78}$$

We further define $p = \frac{P}{T} = P(Y_{\text{test}}^{[ij]} = 1)$ and $n = \frac{N}{T} = P(Y_{\text{test}}^{[ij]} = 0)$, the fractions of positive and negative labels in the test set, respectively. We can then express the expected values of the confusion matrix as functions of a given threshold s^* for the decision value s:

$$\text{TP}(s^*) = p(1 - F_1(s^*)) \tag{2.79}$$

$$\text{TN}(s^*) = nF_0(s^*) \tag{2.80}$$

$$\text{FP}(s^*) = n(1 - F_0(s^*)) \tag{2.81}$$

$$\text{FN}(s^*) = pF_1(s^*) \tag{2.82}$$

which, in turn, yields

$$\text{TPR}(s^*) = 1 - F_1(s^*) \tag{2.83}$$

$$\text{TNR}(s^*) = F_0(s^*) \tag{2.84}$$

$$\text{FPR}(s^*) = 1 - F_0(s^*) \tag{2.85}$$

$$\text{Pr}(s^*) = \frac{p(1 - F_1(s^*))}{p(1 - F_1(s^*)) + n(1 - F_0(s^*))} \tag{2.86}$$

The area under the ROC curve can now be expressed as

$$A_{\text{ROC}} = \int_0^1 \text{TPR}(\text{FPR})\, d\text{FPR} = \int_{\infty}^{-\infty} \text{TPR}(s) \frac{d\text{FPR}(s)}{ds}\, ds = \int_{-\infty}^{\infty} (1 - F_1(s)) f_0(s)\, ds \tag{2.87}$$

The integration limits arise from the fact that FPR is maximal when all instances are classified as positives and minimal when all instances are classified as negatives, which respective corresponds to $s^* \to -\infty$ and $s^* \to \infty$. This formulation of AUROC by **??** leads to the most common intuition behind the metric. The first factor $(1 - F(s^*))$, represents the fraction of positive instances that are ranked higher than the threshold s^*, alternatively expressed as $\int_{s^*}^{\infty} f_1(s)\ ds$. f_0, as defined by **??**, represents the probability of finding a negative instance within s^* and $s^* + ds$. Hence, the product of both factors represents the joint probability of having a negative instance between s^* and $s^* + ds$ while also finding a positive instance with $s > s^*$. After integration over all possible thresholds, we conclude that the AUROC score represents the overall probability of randomly selecting a positive instance ranked higher than a randomly selected negative instance ().

The baseline score 0.5 can be derived as follows. A random classifier is defined as an estimator incapable of distinguishing between the true label distributions of each class. That is, a classifier is a random classifier if and only if $f_0(s) = f_1(s) \forall s$. As a consequence,

$$A_{\text{ROC}} = \int_{-\infty}^{\infty} (1 - F_0(s)) f_0(s)\ ds = -\int_{-\infty}^{\infty} (1 - F_0)\ d(F_0) = \left[\frac{(1 - F_0)^2}{2}\right]_{-\infty}^{\infty} = \frac{1}{2} \tag{2.88}$$

?? also shows a characteristic of AUROC discussed in the previous section: the AUROC score is independent of the relative prevalence of each class in the test set. There is no influence of p or n and only the dependency on the learned decision value distributions.

The area under the PR curve (AUPRC) is defined as

$$\begin{aligned} A_{\text{PRC}} = \int_0^1 \text{Pr(TPR)}\ d\text{TPR} = \int_{\infty}^{-\infty} \text{Pr}(s) \frac{d\text{TPR}(s)}{ds}\ ds = \\ = \int_{-\infty}^{\infty} \text{Pr}(s) f_1(s)\ ds = \int_{-\infty}^{\infty} \frac{p(1 - F_1(s)) f_1(s)\ ds}{p(1 - F_1(s)) + n(1 - F_0(s))} \end{aligned} \tag{2.89}$$

From **??**, AUPRC can be interpreted as the average precision weighted by the distribution of positive instances. Analogously, it corresponds to collecting the decision values attributed to each positive instance, and then calculating the average precision considering only these values as classification thresholds. Furthermore, unlike AUROC, the true label cumulative distributions (F_0 and F_1) appear each weighted by their respective class prevalences (p and n).

2.4.4.2 AUPR and AUROC in terms of ranked decision values

When computing AUROC or AUPR for a given estimator on a test set, the values of the decision function s are usually not directly considered. Instead, the decision values outputted for each test instance are used to rank them from lowest to highest, and from these ranked test labels the curves and respective areas are obtained. The specific values of s are thus indifferent to the scoring process, as long as the ranking is preserved. If the percentile rank of each test instance is denoted by $r \in [0, 1]$ and each test instance is associated with a decision value, there is a one-to-one monotonic correspondence between r and s in the limit of an infinite number of test samples. Formally, we have

$$r(s) = pF_1(s) + nF_0(s) \tag{2.90}$$

$$dr = [pf_1(s) + nf_0(s)]\ ds \tag{2.91}$$

We also take the liberty to represent $g(r) = g(s(r))$, so that

$$F_1(r) = \int_0^r \frac{f_1(r)}{pf_1(r) + nf_0(r)}\,dr \tag{2.92}$$

$$\frac{dF_1(r)}{dr} = \frac{f_1(r)}{pf_1(r) + nf_0(r)} \tag{2.93}$$

AUPRC as defined in **??** can now be written as

$$\begin{aligned}
A_{\text{PRC}} = p\int_0^1 \frac{1 - F_1(r)}{(p+n) - [pF_1(r) + nF_0(r)]}\,\frac{f_1(r)}{[pf_1(s) + nf_0(s)]}\,dr &= \\
= p\int_0^1 \frac{[1 - F_1(r)]}{[1-r]}\frac{dF_1(r)}{dr}\,dr = -\frac{p}{2}\int_0^1 \frac{1}{1-r}\,\frac{d[1-F_1(r)]^2}{dr}\,dr &= \\
= -\frac{p}{2}\left|\frac{[1-F_1(r)]^2}{1-r}\right|_0^1 + \frac{p}{2}\int_0^1\left[\frac{1-F_1(r)}{1-r}\right]^2 dr &= \\
= \frac{p}{2}\left\{1 + \int_0^1\left[\frac{1-F_1(r)}{1-r}\right]^2 dr\right\} = \frac{p}{2}\left\{1 + \int_0^1 [\text{Pr}(r)]^2 dr\right\}
\end{aligned} \tag{2.94}$$

in which the upper boundary term is determined by using L'Hôpital's rule and noticing that $F_1(r=1) = 1$ and $\frac{dF_1(r)}{dr} \leq \frac{1}{p}$.

$$\lim_{r\to 1}\frac{[1-F_1(r)]^2}{1-r} = 2\cdot\lim_{r\to 1}[1-F_1(r)]\,\frac{dF_1(r)}{dr} = 0$$

?? reveals that AUPRC is closely related to the average squared precision across all ranks.

We can also express the AUROC in terms of the percentile ranks:

$$\begin{aligned}
A_{\text{ROC}} = \frac{1}{n}\int_0^1 (1-F_1(r))\left(\frac{nf_0(r)}{nf_0(r) + pf_1(r)}\right)\,dr &= \\
= \frac{p}{n}\int_0^1 (1-F_1(r))\left(1 - \frac{pf_1(r)}{nf_0(r) + pf_1(r)}\right)\,dr &= \\
= \frac{1}{n}\int_0^1 (1-F_1(r))\left(1 - p\frac{dF_1(r)}{dr}\right)\,dr &= \\
= \frac{1}{n}\int_0^1 (1-F_1(r))\,dr - \frac{p}{n}\int_0^1 (1-F_1(r))\,d(F_1(r)) &= \\
= \frac{1}{n}\int_0^1 (1-F_1(r))\,dr + \frac{p}{n}\left[\frac{(1-F_1(r))^2}{2}\right]_0^1 &= \\
= \frac{1}{n}\left\{\int_0^1 (1-F_1(r))\,dr - \frac{p}{2}\right\}
\end{aligned}$$

Equations **??** and **??** put AUPR and AUROC in a similar format, better delineating the differences between the two metrics. Consider expressing both now in terms of the TPR.

$$A_{\text{ROC}} = \frac{1}{n}\left\{\int_0^1 \text{TPR}(r)\,dr - \frac{p}{2}\right\} \tag{2.95}$$

$$A_{\text{PR}} = \frac{p}{2}\left\{1 + \int_0^1\left[\frac{\text{TPR}(r)}{1-r}\right]^2 dr\right\} \tag{2.96}$$

Ignoring constant terms and factors, both metrics are centered on integrating the TPR over all possible ranks, each rank representing a classification threshold. The crucial difference is that AUROC equally considers all TPR values, while AUPR weights each TPR value by the inverse of the reversed ranks. Therefore, the AUPR metric emphasizes the effect of the highest ranked instances. Additionally, the square exponent in AUPR's integrand further prioritizes high values of $\frac{\text{TPR}(r)}{1-r}$, likely reinforcing the effect. In other words, AUPR's penalty for having false positives is much higher when the rank is also high (TPR $= 1 -$ FPR).

On the other side, AUROC is more concerned with the overall ranking, being more likely to allow a few highly-ranked false positives. Using AUPR is then more suited when one is interested in selecting a restricted number of top-ranked instances from a pool of predictions, such as in recommendation systems or drug discovery tasks. On the other hand, AUROC should be favored when the goal is to rank a large batch of interactions. Examples would be modeling genetic interactions in a genome-wide fashion or building interaction databases.

We also argue that AUROC could be preferable for comparing models under the PU assumption, at least in purely theoretical settings. For PU datasets, we naturally expect some negative-labeled instances to be very highly ranked since they could be, in fact, unannotated positives. AUPR would strictly penalize such predictions, favoring models that consider the labeling mechanism itself rather than only the underlying interaction mechanism. AUPR could thus undermine the model's potential to discover new interactions, likely failing to gauge the generalization capabilities of algorithms in a PU context. In fact, we show in the next section that AUROC is closely related to the Mean Percentile Rank metric, which has been suggested for PU learning contexts of interaction prediction and recommendation systems (**?**, **?**). Conversely, the importance of AUPR lies in applied scenarios where selecting false positives could be costly or have significant negative impacts. For instance, when selecting a small number of drug candidates for further testing.

2.4.4.3 AUROC is the normalized mean percentile ranks

From **??**, we can express AUROC as

$$
\begin{aligned}
A_{\text{ROC}} &= \frac{1}{n}\left\{\int_0^1 (1 - F_1(r))\, dr - \frac{p}{2}\right\} = \\
&= \frac{1}{n}\left\{1 - \int_0^1 F_1(r)\, dr - \frac{p}{2}\right\} = \frac{1}{n}\left\{1 - \int_0^1 \frac{d(r)}{dr} F_1(r)\, dr - \frac{p}{2}\right\} = \\
&= \frac{1}{n}\left\{1 - [rF_1(r)]_0^1 + \int_0^1 r\frac{dF_1(r)}{dr}\, dr - \frac{p}{2}\right\} = \frac{1}{n}\left[\int_0^1 r\frac{dF_1(r)}{dr}\, dr - \frac{p}{2}\right] \qquad (2.97)
\end{aligned}
$$

which offers a different perspective on AUC ROC. The term $\int_0^1 r\, c(r)\, dr$ represents the average percentile ranking of positive labels, or the average distance of a positive label from the origin in the plot of $c(r)$. This quantity by itself, the mean percentile ranking (MPR), is a common metric in the realm of collaborative filtering for recommendation systems (**?**, **?**) and has also been proposed to bipartite interaction prediction (**?**, **?**, **?**).

$$\text{MPR} = \int_0^1 r\, c(r)\; dr \tag{2.98}$$

At first glance, MPR seems to offer some advantages over AUROC: besides simpler in formulation, MPR also seemingly does not depend on p or n, the relative amounts of postive and negative labels in the test set, making comparisons across different test sets more direct. However, consider the maximum and minimum values of MPR, achieved for the ideal $c(r)$ distributions:

$$c_{\text{max}}(r) = \frac{\mathbb{I}(r > n)}{p} \tag{2.99}$$

$$c_{\text{min}}(r) = \frac{\mathbb{I}(r < p)}{p} \tag{2.100}$$

The extrema of MPR do in fact depend on n:

$$\text{MPR}_{\text{max}} = \frac{1}{p}\int_n^1 r\; dr = \frac{1-n^2}{2p} = \frac{1+n}{2} \tag{2.101}$$

$$\text{MPR}_{\text{min}} = \frac{1}{p}\int_0^p r\; dr = \frac{p^2}{2p} = \frac{p}{2} = \frac{1-n}{2} \tag{2.102}$$

If we then try to remedy this issue by normalizing MPR to the $[0,1]$ interval, we obtain

$$\text{MPR}_{\text{norm}} = \frac{\text{MPR} - \text{MPR}_{\text{min}}}{\text{MPR}_{\text{max}} - \text{MPR}_{\text{min}}} = \frac{\text{MPR} - \frac{1-n}{2}}{n} = \frac{1}{2} + \frac{1}{n}\left(\text{MPR} - \frac{1}{2}\right) = A_{\text{ROC}} \tag{2.103}$$

precisely the expression for the AUROC (**??**). To the best of our knowledge, we thus show for the first time that employing MPR as a metric is equivalent to using AUROC normalized to the $[0,1]$ interval. Since the statistics for model comparison employed in this study consider only the order of the estimators based on their scores and not the score values themselves, MPR results would not differ in any way from AUROC and hence will not be further considered in this work.

$$\begin{aligned} A_{\text{PR}} &= \int_0^1 pr\; d(tpr) = \int_{tpr(0)}^{tpr(1)} pr\frac{d(tpr)}{dr}\; dr = \\ &= \int_0^1 \left(p\frac{1 - \int c(r)\; dr}{1-r}\right)(-c(r))\; dr = -p\int_0^1 \frac{1-C(r)}{1-r}C'(r)\; dr \end{aligned} \tag{2.104}$$

$$\begin{aligned} -p\int_0^1 \frac{1-C(r)}{1-r}C'(r)\; dr &= -n\int_0^1 \frac{1}{n}\frac{r-pC(r)}{r}\frac{1}{n}\left(1 - p\,c(r)\right)\; dr = \\ &= -\int_0^1 \frac{r-pC(r)}{r}\left(1 - p\,c(r)\right)\; dr = -p\int_0^1 \left(\frac{1}{p} - \frac{C(r)}{r}\right)\left(\frac{1}{p} - c(r)\right)\; dr = \\ &= -\int_0^1 \frac{1}{p} - \frac{C(r)}{r}\; dr - p\int_0^1 \left(\frac{1}{p} - \frac{C(r)}{r}\right)(-c(r))\; dr \end{aligned} \tag{2.105}$$

$$-p\int_0^1 \frac{1-C(r)}{1-r}C'(r)\,dr = -p\left[\left[\frac{1-C(r)}{1-r}C(r)\right]_0^1 - \int_0^1 \left(\frac{1-C(r)}{1-r}\right)' C(r)\,dr\right] =$$
$$= -p\left[\left[\frac{1-C(r)}{1-r}C(r)\right]_0^1 - \int_0^1 \left(\frac{-C'(r)C(r)}{1-r} + \frac{(1-C(r))C(r)}{(1-r)^2}\right) dr\right] =$$
$$= -p\left[\left[\frac{1-C(r)}{2(1-r)}C(r)\right]_0^1 - \int_0^1 \frac{1}{2}\left(\frac{(1-C(r))C(r)}{(1-r)^2}\right) dr\right] \quad (2.106)$$

$$\int_0^1 \frac{C(r)C'(r)}{1-r}\,dr = \left[\frac{(C(r))^2}{1-r}\right]_0^1 - \int_0^1 \left(\frac{C(r)}{1-r}\right)' C(r)\,dr =$$
$$= \left[\frac{(C(r))^2}{1-r}\right]_0^1 - \int_0^1 \left(\frac{C'(r)C(r)}{1-r} + \frac{C(r)^2}{(1-r)^2}\right) dr =$$
$$= \frac{1}{2}\left[\frac{(C(r))^2}{1-r}\right]_0^1 - \frac{1}{2}\int_0^1 \frac{C(r)^2}{(1-r)^2}\,dr \quad (2.107)$$

$$\int_0^1 \frac{C'(r)}{1-r}\,dr = \left[\frac{C(r)}{1-r}\right]_0^1 - \int_0^1 C(r)\left(\frac{1}{1-r}\right)' dr =$$
$$= \left[\frac{C(r)}{1-r}\right]_0^1 - \int_0^1 \frac{C(r)}{(1-r)^2}\,dr \quad (2.108)$$

2.4.5 General experimental settings

2.5 Experiments

2.5.1 Empirical time complexity analysis

To empirically assess the training time complexity of the tree models under study, we artificially generate a series of bipartite datasets by filling three n by n matrices with pseudo-random values, representing the two X matrices and the y matrix on each interaction. Values were taken uniformly from the interval $[0, 1]$ for the feature matrices and from the interval $[0, 100]$ for the target matrix. We thus represent interactions between n drugs and n proteins, each being described by n features.

We then train the GMO and the optimized GSO versions of a single bipartite decision tree (BDT) and a single bipartite ExtraTree (BXT) on each of the generated datasets, measuring their training duration in seconds. The results are shown in Figure **??**. From the least squares linear regression on the log-log plot, we see that the estimated training time complexities closely follow the theoretical expectations developed under Section **??**, with slopes referring to the GSO models (predicted to be $O(n^3)$) approaching 3 while the GMO models (predicted to be $O(n^3 \log(n))$) produce slope between 3 and 4.

Statistical testing further shows that the empirical time complexity of the proposed GSO algorithms are indeed significantly lower than that of their GMO counterparts (see the caption for Figure **??**).

The slightly lower slopes for the ExtraTrees in comparison both with the BDTs and with the theoretical complexities are also expected, since the bottleneck calculation for these models in the asymptotic regime is the search for the minimum and maximum values of each feature in each node (lines **??**), which can be done much faster than the search for the best split employed by the greedy decision trees (**??**), even though both procedures have the same order of asymptotic complexity. As such, much larger datasets would be required to observe the asymptotic behavior of the ExtraTrees. In spite of that, the empirical complexity of `bdt_gso` is still observed to be highly significantly lower than that of `bxt_gmo`, validating once more the prediction that `bdt_gso` should present faster training times than `bxt_gmo` on sufficiently large datasets.

2.5.2 Comparison between GSO models

To assess the impact of global single-output optimizations in bipartite decision tree growing, we compare three slightly different training methods for BXT and BRF models.

- **ngso**: Naive global single output implementation (Section **??**);
- **ngsous**: Naive global single output implementation with undersampling of the non-interacting pairs to yield a balanced training set (Section **??**);
- **gso**: Optimized implementation of global single output trees (Section **??**).

While no significant divergence was measured among the GSO models using the entirety of the training data, undersampling revealed to significantly degrade the predictive performance of both forests in terms of AUPR and MCC (Figure **??**), even though it is arguably the most common procedure when dealing with this kind of data ().

On the other hand, AUROC is significantly improved by the undersampling procedure, which is most likely an artifact of the highly imbalanced nature of the present data, as explained as follows. The models grown on the undersampled datasets are naturally the most likely to assign positive labels to new interactions in general, improving TPR at the expense of also increasing FPR. However, since negative labels greatly outnumbers positive labels in the test sets of our current scenario, an increase in FPR impacts a much larger number of predictions than the same increase in TPR. In spite of that, AUROC equally treats TPR and FPR, so that the impact of a high FPR is underestimated. As such, AUROC results could be deemed as unrepresentative of model performance in this setup.

When comparing training times, the common choice for undersampling in previous works is justified, as an expressive reduction of training time is observed for both forests (Table

??) relative to naive GSO training. Nevertheless, it is remarkable that the optimized implementation of GSO forests achieves similar training times in comparison to undersampled GSO without the AUPR and MCC burden of undersampling, keeping the higher scores resulting from employing the entirety of the dataset. For larger and less imbalanced datasets, the optimized implementation of GSO forests is expected to be even more advantageous, in agreement with the theoretical time complexity analysis (Section **??**).

In conclusion, the proposed approach confidently enables the use of the entire training data in a much shorter time frame than naive implementations without the need for data undersampling, which is statistically expected to yield better prediction scores for forest predictors.

2.5.3 Comparison between GMO prediction weights

In this work we propose a different prototyping strategy to determine the output value of each leaf in a GMO decision tree, taking the similarity matrices of our use cases into consideration (**??**). We here compare such strategies, building BXT and BRF models for every option. The minimum rows per leaf and minimum columns per leaf were both set to 5, ensuring that at least 5 samples of each domain are considered when calculating the prototype values. To observe the effect of this early-stopping criterion by itself, we also include forests of fully-grown trees in the comparison. The compared models are described below.

- **`gmosa`:** proposed by pliakos2018global, the output of each leaf is the average of the labels of the learned samples that reach that leaf (**??**).
- **`uniform`:** also proposed by pliakos2018global, the only difference from the `gmosa` strategy occurs in TL or LT test settings, when the average is taken only among the labels of the known sample of the pair being predicted (**??**).
- **`precomputed`:** introduced by us (as also `square` and `softmax`), the labels in each leaf are weighted by the similarities between the learned samples and the pair being predicted (**??**).
- **`square`:** the labels in each leaf are weighted by the squared similarities between the learned samples and the pair being predicted (**??**).
- **`softmax`:** the labels in each leaf are weighted by the exponential of the similarities between the learned samples and the pair being predicted (**??**).
- **`full`:** the trees are grown until a single interaction remains in each leaf.

The results for BRF (**??**) and BXT (**??**) were similar. In all cases except LT+TL average precision, using the square of the similarities to weight the labels in each leaf resulted in in the best scores. For the LT+TL average precision score, growing the decision tree to its maximal size was the best strategy, followed by `uniform`, the original proposal by pliakos2018global of averaging only the outputs of the learned samples (known from the training set) in each leaf. With the exception of the LT+TL AUROC metric for BXT, the mentioned winning models were statistically distinguished from all the remaining estimators. For LT+TL AUROC, the superiority of `square` could not be attested when compared to the `uniform` strategy.

The fully-grown versions of both forest algorithms are shown to be especially advantageous when considering the LT+TL average precision. This suggests that building trees to their maximum depth is the best strategy for learning tasks in which

1. one of the domains is fixed, with the final goal being to model how new instances will bind to this known set of entities and
2. controlling the number of false positives is equally important to correctly ranking the positive interactions.

Examples that satisfy the first condition are drug repositioning () or dyadic prediction ().

We propose that this behaviour results from the ability of a fully-grown tree to independently consider the labels of each learned instance when calculating the prototype, while the methods using weighted averages ivariably mix the labels of a pool of neighbors in each leaf. This hypothesis is supported by the fact that the second-best model regarding LT+TL average precision is the `uniform` strategy, which also uses the labels of individual learned samples to generate the predictions. The hypothesis alone, however, do not explain the superiority of the deeper trees over the `uniform` weights. In this case, the larger tree depth is likely beneficial through i) an increase in the predictive power of each individual tree, and/or ii) an increase in tree diversity, both of which would improve the ensemble's performance as discussed in **??**. We let to future work the more specific investigation of these effects.

On the other hand, `full` and `uniform` show notable inferior performance in the TT evaluation settings, suggesting that the weighting strategy used by the other prototype functions could be an important technique to improve generalization.

We then select the squared weighting strategy and the fully grown trees to be further investigated in the downstream analyses.

2.5.4 Comparison between adaptation strategies

We now compare each of the described approaches for adapting decision forests to bipartite data (**??**, **??** and **??**). The suffixes in the model names of this section indicating the employed bipartite adaptations are briefly described below.

- **`lmo`:** implements the standard local multi-output approach (SLMO; **??**) by training four separate multioutput models, two for each domain. First explored by schrynemackers2015classifying.

- **`lso`:** also implements the SLMO approach (**??**), but each multioutput model is instead a composition of several local single-output (LSO) models, i.e. one model is trained for each row or column of the interaction matrix. This setting is similar to the early proposal by bleakley2009supervised, but employing decision forests as the base algorithm.
- **`gmosa`:** a global multi-output forest with single-label averaging (GMOSA; **??**), as explored by pliakos2019network under the name eBICT.
- **`gmo`:** a global multi-output (GMO; **??**) forest with minimum leaf dimensions of 5 by 5, implementing our proposed squared-similarities weighting for the prototype function (**??**). Apart from the new prototype, this model is based on the original GMO trees proposed by pliakos2018global. Despite their original results suggesting advantage over GMOSA, to the best of our knowledge, this is the first time the GMO trees are employed in building decision forests.
- **`gso`:** a bipartite global single-output (BGSO; **??**) forest, as initially explored by schrynemackers2015classifying but now implementing our proposed algorithm with improved computational complexity.
- **`sgso_us`:** implements the standard global single-output (SGSO; **??**) adaptation, additionally employing undersampling of the non-interacting pairs to yield a balanced training set (**??**).

All the global models were built with 100 trees. For SLMO, each of the four forests used 50 trees, while for LSO 50 trees were used for each row or column of the interaction matrix.

In the LT+TL AP evaluation setting, the pattern observed in **??** again emerges: the label averaging strategy employed by GMO performs considerably worse in comparison to forests that separately consider each known instance. In particular, the SLMO and SLSO adaptations yield the clear best BRF models in terms of LT+TL AP, and these adaptations interpret each instance as a separate output to be predicted. Among the local adaptations, SLMO significantly ouperforms SLSO. SLSO treats each training instance as a completely independent task, building a separate forests for each row and column of the interaction matrix. As such, the previous result shows that this complete independence is not desirable for the learning problems under study, and exploring label correlations between instances of the same domain, as performed by SLMO, is beneficial. This result could partially be a consequence of the very sparse nature of our problems: if the interaction information of each instance is limited, it becomes advantageus to aggregate information from other instances with correlated interactomes. We then speculate that the advantage of SLMO over SLSO could become less prominent once the number of known interactions per instance increases and more data are available for training.

Still considering LT+TL AP but focusing on BXT models instead, we notice that SLMO loses the advantage to the fully-grown bipartite trees GMO and GSO. A possible explanation comes from the fact that local approaches yield shallower trees, and much more randomized trees could be required to achieve comparable performance. In more detail, first notice that the individual performance of each tree in a BXT forest is lower than that of a tree in a BRF. Thus, a BXT ensemble requires a larger number of trees to reach satisfactory performance. Additionally, if the trees are shallow, they tend to be less representative of the training data (each tree node brings a little more information on the dataset). Even more trees then should be required to compensate the randomness of each individual. Finally, building a forest locally as in SLMO or SLSO results that each tree is trained on a much smaller number of samples in comparison to considering each dyad as a separate instance. Therefore, local approaches generate smaller and less representative trees. We then suggest that the advantage of SLMO could also manifest for BXT estimators if the number of trees in the ensemble were to be increased.

For both BXT and BRF in all the TT settings, the GMO model significantly outperforms the other estimators. GMO is also the best model in the LT+TL AUROC setting when comparing BXTs and the second best for BRFs. This reinforces the findings of **??** suggesting that larger leaves and label weighting seems to improve generalization to unseen instances. We also highlight SLMO as a prominent strategy for BRFs under TT AUROC and TT AP, being the second best model in both cases.

The SGSO strategy employing undersampling of negative annotations performed consistently worse than the others under the AP metric. Conversely, it was the second best BXT model in both LT+TL and TT AUROC settings, while surpassing BRF GMOSA also under both AUROC test sets. The explanation would be that the undersampling procedure naturally causes the model to equally prioritize the positive and negative labels, which matches the AUROC's behaviour of equally treating TPR and FPR. More specifically, these models tend to produce a larger number of false positives since they are unaware of the large label imabalance of our test sets. While this substantially affects AP, the AUROC metric is agnostic to the label imbalance (**??**), being much more forgiving with the number of false positives given the large total number of negative labels in the test sets.

We select LMO, GMO, GSO, and GMOSA to be further analysed in the next section.

2.5.5 Effect of interaction matrix reconstruction

It was previously suggested that employing logistic matrix factorization to create a denser representation of the interaction matrix and using this representation as the training data for a BXT forest could improve their performance on DTI datasets (**?**). To test this hypothesis, we compare the bipartite forests cross-validation scores with and without the interaction matrix reconstruction step. As done by (**?**), the reconstruction step was performed using neighborhood-regularized logistic matrix factorization (NRLMF)().

To select hyperparameters for the NRLMF algorithm, we performed a randomized search in which 100 different combinations of hyperparameters were evaluated in terms of their resulting mean squared error over a (nested) bipartite 5-fold diagnonal cross-validation. The best combination found in the inner CV loop was then used to reconstruct the interaction matrix of each outer CV fold, and the resulting matrices were used as the training data for the bipartite forests. Note that a single forest was built by outer CV fold, so that the NRLMF hyperparameter seach was performed independently to the downstream forest performance. The hyperparameters `lambda_rows`, `lambda_cols`, `alpha_cols`, `alpha_cols`, and `learning_-rate` were all independently sampled from a log-uniform distribution bounded by $\frac{1}{4}$ and 2. the number of latent vector components was set to be equal among both axes, and chosen between 50 and 100. The number of neighbors was randomly selected as 3, 5 or 10 in each iteration, and the maximum number of optimization steps was always set to 100.

The results for BRF and BXT are shown by figures **??** and **??**, respectively. The `nrlmf` suffix to a model indicates that it was constructed on the output of NRLMF for each fold.

The interaction matrix reconstruction step is shown to be especially beneficial in terms of ROC AUC scores, improving this score for almost all forest algorithms investigated. The only exceptions were `bxt_gmo` and `brf_gmo` in LT+TL ROC AUC, where the improvement is still observed but not statistically significant. As discussed in **??**, ROC AUC is characterized by leniency towards false positives, which explains the advantage posed by NRLMF in this setting: NRLMF explicitly defines a parameter to prioritize outputting positive interactions (**??**). In accordance with previous investigations (**?**, **?**, **?**, **?**), this parameter was set to 5, meaning that the loss function employed in the gradient descent procedure of NRLMF is weighted 5 times more for positive interactions than for negative interactions, as if 5 copies of each positive interaction were present in the training set.

Even so, the `bxt_gmo` and `brf_gmo` models employing our squared similarities output weighting (sections **??** and **??**) are placed second in their corresponding TT ROC AUC results, both significantly outperforming all NRLMF-combined models except for their own versions, `bxt_gmo__nrlmf` and `brf_gmo__nrlmf` (as aforementioned). `brf_gmo` was also the top performing BRF model in terms of TT average precision, significantly surpassing all but `brf_gmo__nrlmf` and `brf_lmo__nrlmf`. Similarly, `bxt_gmo` was the top performing BXT model under the same metric, significantly surpassing all estimator but `brf_-gmo__nrlmf`, `brf_lmo__nrlmf` and BXT GMOSA NRLMF.

Regarding TT AP, NRLMF still provides a significant improvement for all models except `brf_gmo`, `bxt_gmo` and `bxt_lmo`. While both BRF GMO and BXT GMO resulted in the highest average ranks for TT AP, the top position could not be statistically resolved between the BRFs using GMO, GMO NRLMF, and LMO NRLMF, nor between the BXTs using GMO, GMO NRLMF, LMO NRLMF, and GMOSA NRLMF.

In the LT+TL AP setting for BXT ensembles, NRLMF is shown to significantly degrade the performance of `bxt_gmo`. We attribute this effect to the aforementioned increase in false positives caused by NRLMF, especially since such degradation is not observed under LT+TL

AUROC. Notwithstanding, we note that this degradation of LT+TL AP is exclusive to the GMO adaptation, as all other BXT models are significantly improved by NRLMF in this setting. Moreover, we highlight the differences between the LT+TL AUROC and LT+TL AP results considering BXT GMO and BXT GMO NRLMF: while these models are the best ranked in the former setting, they are the clear worst in the latter. We thus argue that both the GMO strategy and the NRLMF reconstruction step, each in and of itself, have the effect of increasing the tendency towards false positives in the predictions. Their summed effects could thus impair predictive performance especially under LT+TL AP evaluation, whereas in other settings they could missing positives or AUROC evaluation in general, the combina...

For BRF models under LT+TL AP, the advantage of employing NRLMF is not clear. GSO was the only adaptation to significantly benefit from NRLMF in this setting, while GMO and LMO were significantly impaired by it. In fact, the BRF LMO model without label matrix reconstruction significantly outperformed all the other random forests for this evaluation setting.

Another hypothesis can be formulated based on the fact that the dense reconstructed matrix will cause trees to grow to a much deeper extent, as homogeneity of leaves is harder to obtain. As such, this could favor overfitting of BRF models to the output of NRLMF, carrying the generalization error of the NRLMF reconstruction to the results. At the same time, NRLMF being a better model than BRF on TT sets would generate, as observed, the inverse effect, with NRLMF improving performance in this case. Again, BXT's randomized split search procedure likely makes it more resilient to this influence, building uncorrelated individual estimators even if the trees are large and thus more generally benefitting from NRLMF.

In summary, interaction matrix reconstruction by matrix factorization seems to consistently improve BRF and BXT results in terms of ROC AUC and also the results of BXT regarding LT+TL average precision. On TT average precision, while less evidence is found, the results still seem to point in the same direction of a beneficial NRLMF transformation step. On the other hand, the comparisons of BRF under LT+TL average precision evaluation indicate the opposite conclusion, disfavoring usage of NRLMF especially for `brf_lmo` and `brf_gmo`. We thus discourage the application of NRLMF with BRF in scenarios where one of the entities of the interacting pair being predicted is always known to the model, such as in drug repositioning.

2.5.6 Comparison between semi-supervised forests

The comparison results are displayed by **??**.

Regarding LT+TL AUROC, the 0% ILR and 50% ILR do not result in statistically significant differences in performance. For ILR=70%, md random and md size significantly outperform the remaining models, while under IRL=90%, md random, md size, and md density are shown to surpass the others. These results indicate that the MD unsupervised impurity is the most suited to the LT+TL AUROC evaluation setting. Furthermore, we see that the random strategy to select the supervision amount is among the best in both ILR=70% and ILR=90%, suggesting that increasing diversity among the trees could be the main mechanism behind the improvement of the semi-supervised models, rather than necessarily guessing the best σ at each tree node. Since md size is also among the best, it is not clear if the advantage of md size over the other models is due to a better choice of σ or to a more stochastic nature of the σs it selects. As a future investigation, we suggest using completely random values for the unsupervised impurity, to assess the possibility of tree-diversity being the main factor behind the observed improvements.

As for TT AUROC, only ILR=90% yielded significant comparisons. In this case, the mse unsupervised impurity with size, random, and density σ selection were the best strategies, surpassing the remaining. MSE thus seems the better option in terms of AUROC for scenarios with very scarse information and completely unknown instances.

Under LT+TL AP, md fixed significantly outperformed the other models for ILR=50% and ILR=70%. For ILR=0%, md fixed is also the first place, but could not be statistically resolved from the second place ad size. AD size was also the second best model for ILR=0%, 50%, 70%, statistically outperforming all but md fixed in ILR=70%.

Finally, with respect to TT AP, the AD strategies prevail, being the best four models for ILR=50% and ILR=70%, and being among the five best models for ILR=0%. The presence of AD random among the best models again suggests that the main factor for performance improvement is tree-diversity.

2.5.7 Comparison between different forests

Based on the results of **??**, we select the following models for further comparison:

The comparisons are presented by **??**. The results reveal a clear superiority of forests employing the NRLMF as a label imputation strategy, in comparison to those using semi-supervised impurities. The four models employing NRLMF were the four highest ranked estimators in almost all evaluation settings, the two exceptions being LT+TL AP with ILR=0% and ILR=50%. In the first exception (LT+TL AP ILR=0%), BRF GMO NRLMF and BXT GMO NRLMF are the two worst-performing models, whereas BXT GMOSA NRLMF and BXT GSO NRLMF are the first and second best, respectively. In the second exception (LT+TL AP ILR=50%), the two best models are the same, and BRF GMO NRLMF is still one the worst performers. However, BXT GMO NRLMF jumps to the third best position.

Furthermore, under the other metrics (LT+TL AUROC, TT AUROC, and TT AP), BXT GMO NRLMF was the highest ranked estimator in almost all cases, the only exception being TT AP IRL=0%, where it was only behind BXT GMOSA NRLMF. Notwithstanding, the comparison between BXT GMO NRLMF and BXT GMOSA NRLMF was still not statistically significant in this setting. Similarly, no statistical difference is found between these models in the LT+TL AUROC 0% setting, where they also occupy the first positions. The leadership of

BXT GMO NRLMF is also not significant in comparison to BXT GSO NRLMF under TT AP 50% and TT AP 70%. For all remaining cases where BXT GMO NRLMF was the best model, it was statistically significantly better than all other estimators (TT AP 90, LT+TL AUROC 50, LT+TL AUROC 70, LT+TL AUROC 90, and all TT AUROC).

Another observation is that some semi-supervised impurities significantly improve predictive performance relative to the original BXT GSO, especially when more annotations are missing. This conclusion is based on the fact that, in all settings with ILR != 0 except LT+TL AP, the original BXT GSO is among the three worst performers. For all AUROC settings with ILR!=0, it was the lowest ranked model. On the other hand, md fixed is noted to significantly surpass BXT GSO in 10 of the 16 evaluation settings, the exceptions being LT+TL AUROC 0%, TT AUROC 0%, TT AUROC 90%, TT AP 0%, TT AP 70% and TT AP 90%. Therefore, although not as effective as the NRLMF reconstruction technique, using semi-supervised trees seems indeed beneficial when label information is scarce.

When comapring the BXT against the BRF models, we notice that BXT GMO NRLMF significantly outperformed BRF GMO NRLMF in all settings. The other random forest, BFR LMO, was also significantly outperformed by BXT GMO NRLMF in all cases but LT+TL AP 0%, where the opposite was observed. Similar results hold for the other BXT models as well: they significantly surpassed BRF LMO in the vast majority of test configurations, with the only exception being TT AUROC 0%, in which the comparison between BXT GMOSA NRLMF and BRF LMO was not significant. These results suggest that BXT models could offer significant advantages over BRF models in the context of DTI prediction. This conclusion is especially relevant given that the BXT training algorithm is considerably faster than the procedure for building BRFs, as discussed in **??**.

2.5.8 Comparison with previous works

The algorithms being considered in this section are listed below, and their scoring results are shown by **??**.

BXT GMO NRLMF was the best ranked model in 11 out of the 16 test settings analysed. The exceptions were the four LT+TL AP configurations and TT AP 0%. In TT AP 0%, BXT GMO NRLMF was only behind lmo rls, and among the three models that could not be statistically distinguished from lmo rls (the others being kron rls bxt and gmosa nrlmf). In LT+TL AUROC 0%, the comparison between BXT GMO NRLMF and the second best model, bxt gmosa nrlmf, was also not statistically significant. In the remaining 10 cases where BXT GMO NRLMF prevails, it was significantly better than all other learning algorithms analysed.

Regarding LT+TL AP, in ILR=50 and ILR=70 bxt gso nrlmf and bxt gmosa nrlmf significantly outperform the other models. In ILR=0, bxt gmosa nrlmf was the best model and bxt gso nrlmf was the second best, but neither could be statistically distinguished from the third best model, kron rls. In ILR=90, bxt gso nrlmf was the best model and bxt gmosa nrlmf was the second best, but neither could be statistically distinguished from the third best model, kron rls. Under ILR=90, bxt gso nrlmf and bxt gmo nrlmf are shown to significantly outperform the others.

We notice that the highest ranked bipartite forests in each evaluation setting are always able to significantly surpass NRLMF alone. This demonstrates that the forests are able to capture different patterns than the NRLMF model, even though they are trained on NRLMF's outputs. If this result were to not hold, one could argue that the forests might be merely approximating the predictive function learned by NRLMF, rather than expanding on the information extracted in the matrix factorization step.

We also note that the competitiveness of the kron rls model for TT AP 0% and LT+TL AP 0% is remarkable, since kron rls has notably low training times in comparison to the other models. However, the algorithm seems to not perform as well when a larger number of positive annotations is missing.

2.5.9 Drug-Target affinity prediction

In this section, we evaluate bipartite forests performance in a bipartite regression dataset, comparing them to state of the art deep neural networks.

- deep_dta_raw: Uses convolutional layers to encode raw amino acid sequences and SMILES strings of drug molecules. DeepDTA (**?**)
- transformer_raw: DNN employing transformer modules to embedd the raw amino acid sequence of target proteins and SMILES string of drug molecules. Parameters were based on MolTrans ()

The bxt_gmosa model (**?**) significantly outperforms both neural networks and brf_gso in all scenarios. bxt_gso also outperforms the neural networks and brf_gso in the TT setting, and score significantly higher than brf_gso and transformer_raw in the remaining configurations. Most impressively, the forest models take considerably less time to train than the neural networks given the experimental setup, as shown by Figure **??**, with the bxt_gso still displaying highly competitive performance despite providing sensible gains in time complexity in comparison to the GMO forests and a naive GSO implementation (see sections **??** and **??**).

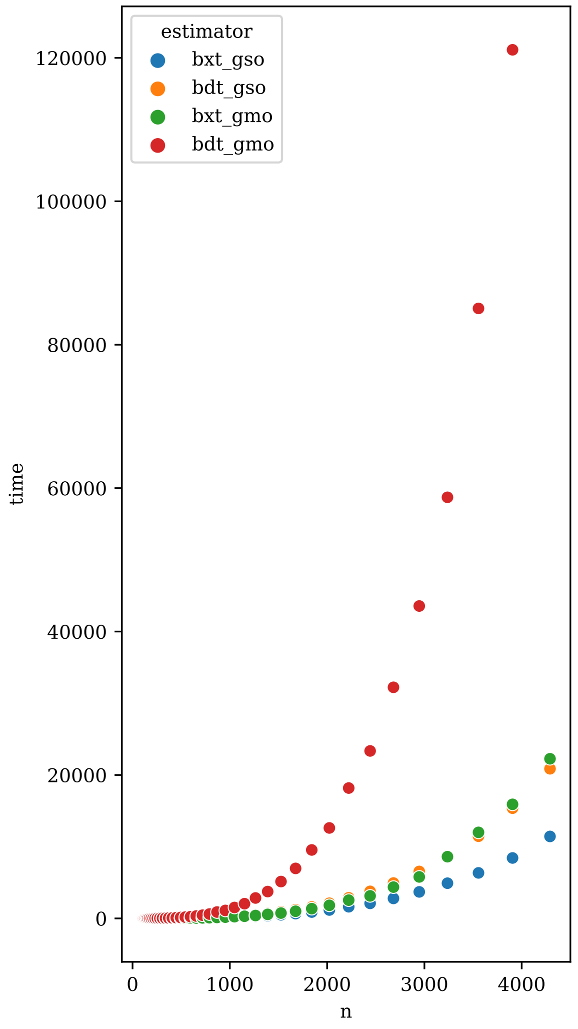
This result points bipartite ExtraTree ensembles as state-of-the-art regression models in drug-target affinity prediction tasks, with highly competitive improvements in training efficiency.

2.6 Final remarks

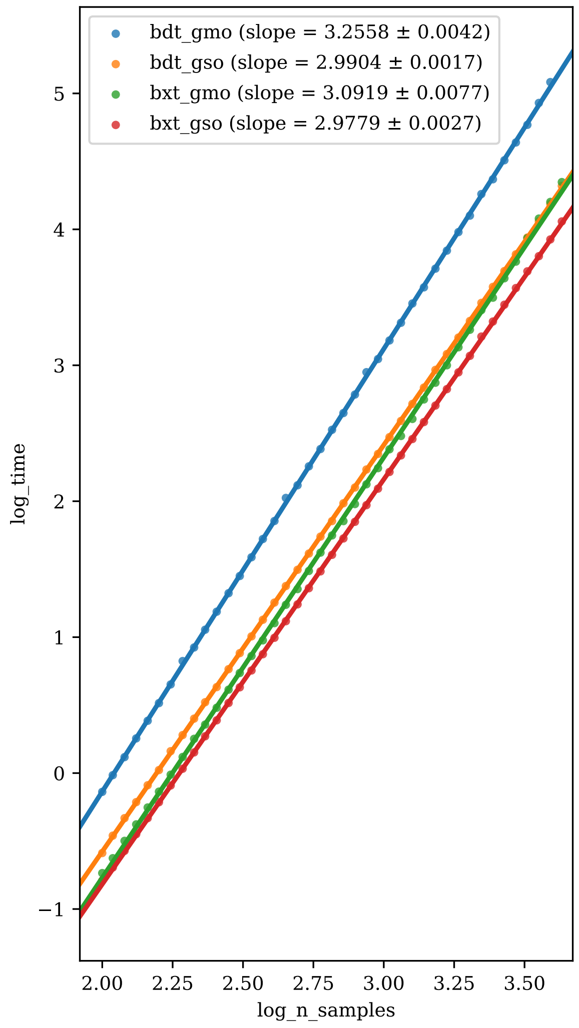
A new Biclustering Random Forest (BRF), and semi-supervised tree-ensembles models were proposed.

The BRF estimator obtained competitive scores against the original PBCT ensemble model eBICT, with nearly 0.1 higher AUROC median on completely new test sets, although not statistically significant.

Using only the splitting feature column to calculate impurity lead to poorer results and thus needs technique refinements for future analyses.



(a) Training durations in seconds versus the the number of samples and features.



(b) Logarithm of the training durations in seconds versus the logarithm of the number of samples and features.

Figure 1 – Empirical time complexity estimation of the proposed bipartite global single-output (BGSO) and the global multi-output (GMO) (**?**) algorithms. Bipartite versions of both extremely randomized trees (**?**) (BXT) and greedy decision trees (**?**) (BDT) were built under the GSO and GMO scheme and trained over artificial datasets of varying numbers of samples (as described in section **??**). Applying least-squares linear regression to the logarithm of the values in (a), as shown in (b), the empirical slopes and respective standard deviations are obtained. Independent two-sample t-tests comparing the slope estimates reveal that the time complexity of `bdt_gso` is highly significantly lower than `bdt_gmo` (p-value $< 10^{-64}$) and even `bxt_gmo` (p-value $< 10^{-20}$), and also that `bxt_gso` significantly exhibits lower complexity than `bxt_gmo` (p-value $< 10^{-22}$). Those values corroborate the theoretical estimates from section **??**.

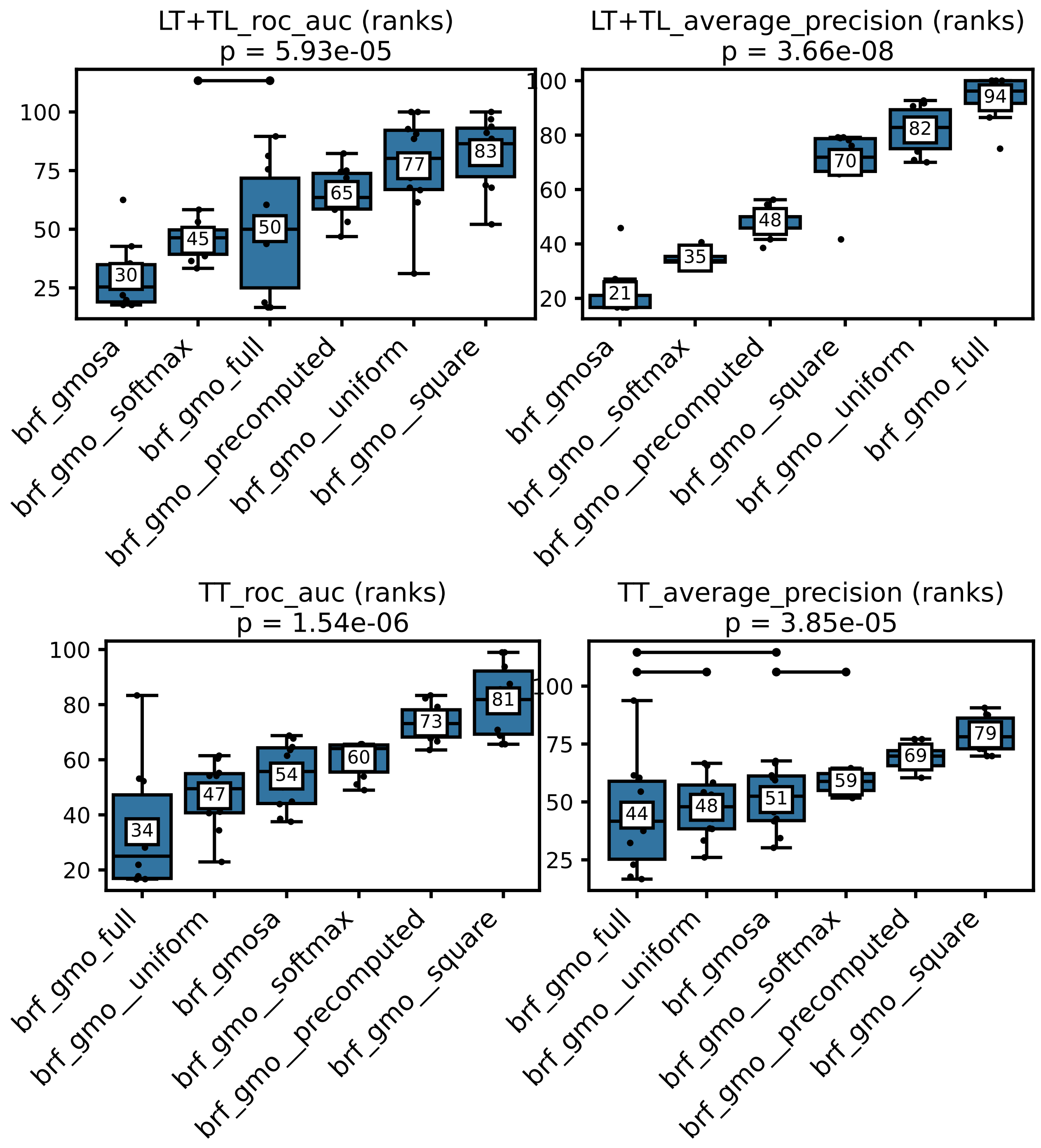


Figure 2 – Percentile rankings of prediction scores of bipartite random forests for different prediction weighting strategies, across all 10 datasets. The omnibus p-value obtained through a Friedman test is indicated below each plot's title. Significant results in each case lead us to perform pairwise Wilcoxon rank-sum tests, whose non-significant results are indicated by crossbars above each plot, that is, statistically undistinguished boxes are connected by the crossbars. The pairwise test results were corrected by the Benjamini-Hochberg procedure among each subfigure, considering that all possible estimator pairs were evaluated even if their comparison is not indicated in the plot. See **??** for further descriptions of each estimator.

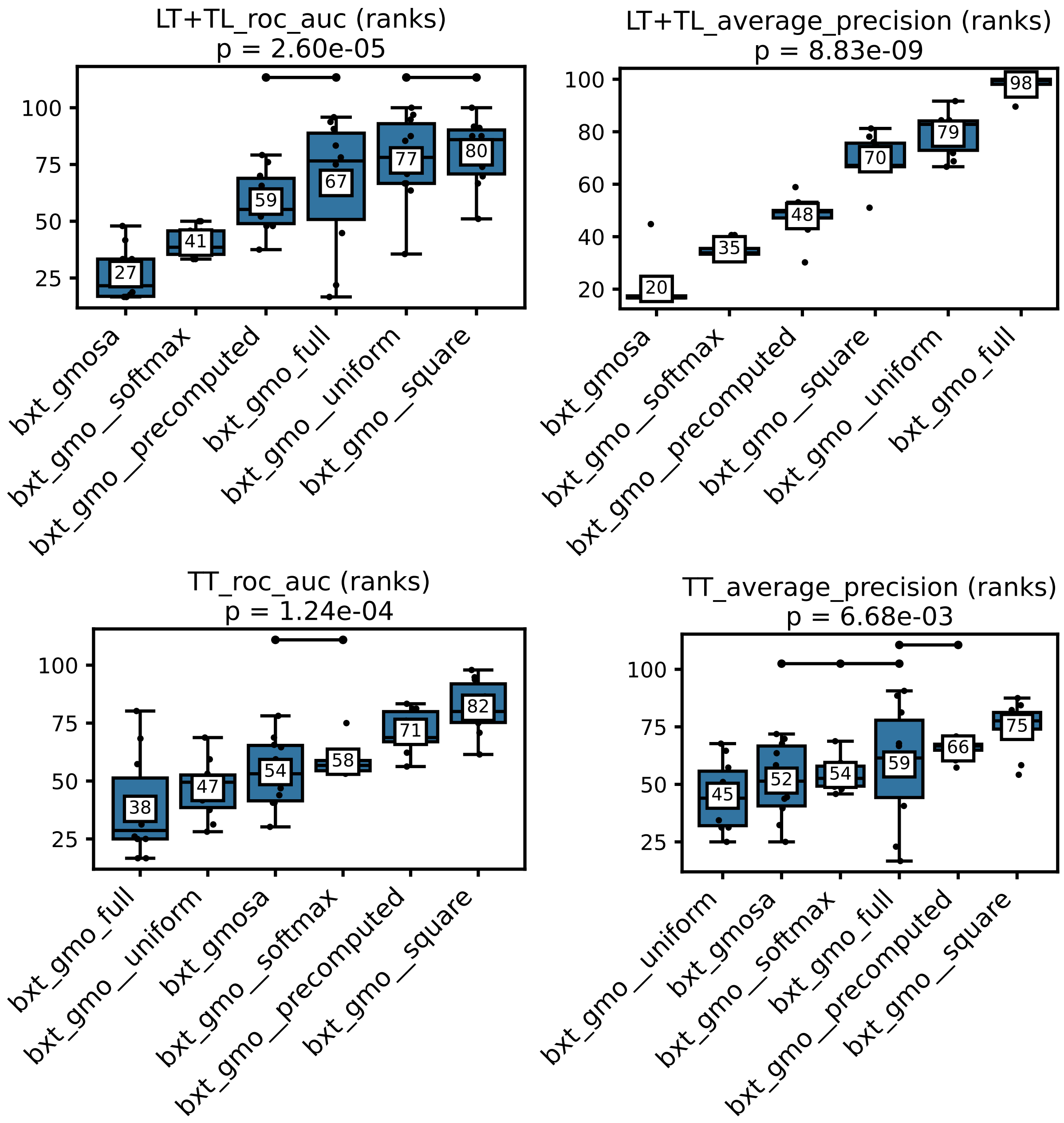


Figure 3 – Percentile rankings of prediction scores of bipartite extremely randomized trees for different prediction weighting strategies, across all 10 datasets. The omnibus p-value obtained through a Friedman test is indicated below each plot's title. Significant results in each case lead us to perform pairwise Wilcoxon rank-sum tests, whose non-significant results are indicated by crossbars above each plot, that is, statistically undistinguished boxes are connected by the crossbars. The pairwise test results were corrected by the Benjamini-Hochberg procedure among each subfigure, considering that all possible estimator pairs were evaluated even if their comparison is not indicated in the plot. See **??** for further descriptions of each estimator.

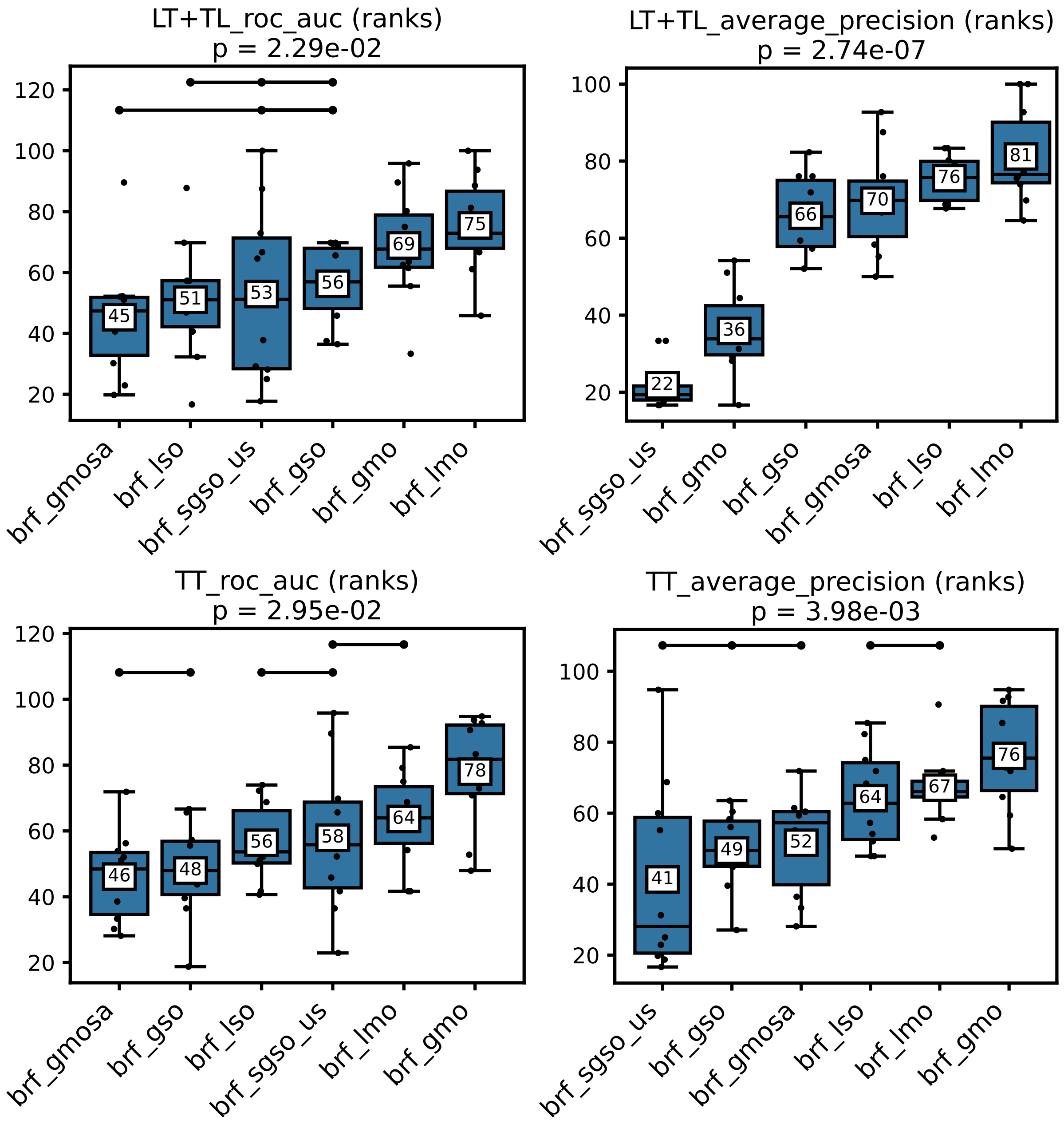


Figure 4 – Percentile rankings of prediction scores of random forest models under different adaptation strategies to bipartite interaction data, across all 10 datasets. The omnibus p-value obtained through a Friedman test is indicated below each plot's title. Significant results in each case lead us to perform pairwise Wilcoxon rank-sum tests, whose non-significant results are indicated by crossbars above each plot, that is, statistically undistinguished boxes are connected by the crossbars. The pairwise test results were corrected by the Benjamini-Hochberg procedure among each subfigure, considering that all possible estimator pairs were evaluated even if their comparison is not indicated in the plot. See **??** for further descriptions of each estimator.

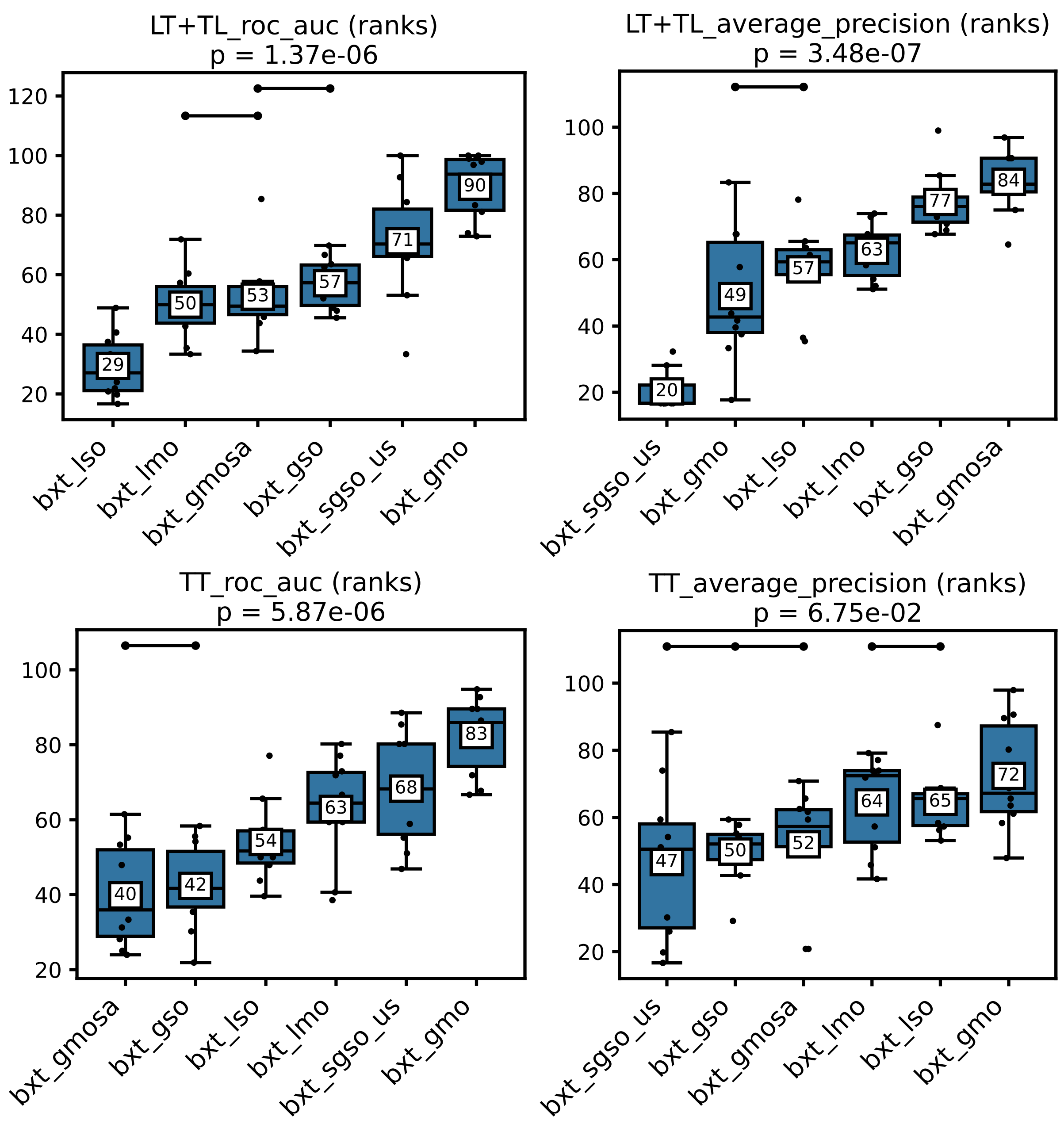


Figure 5 – Percentile rankings of prediction scores of extremely randomized tree ensembles under different adaptation strategies to bipartite interaction data, across all 10 datasets. The omnibus p-value obtained through a Friedman test is indicated below each plot's title. Significant results in each case lead us to perform pairwise Wilcoxon rank-sum tests, whose non-significant results are indicated by crossbars above each plot, that is, statistically undistinguished boxes are connected by the crossbars. The pairwise test results were corrected by the Benjamini-Hochberg procedure among each subfigure, considering that all possible estimator pairs were evaluated even if their comparison is not indicated in the plot. See **??** for further descriptions of each estimator.

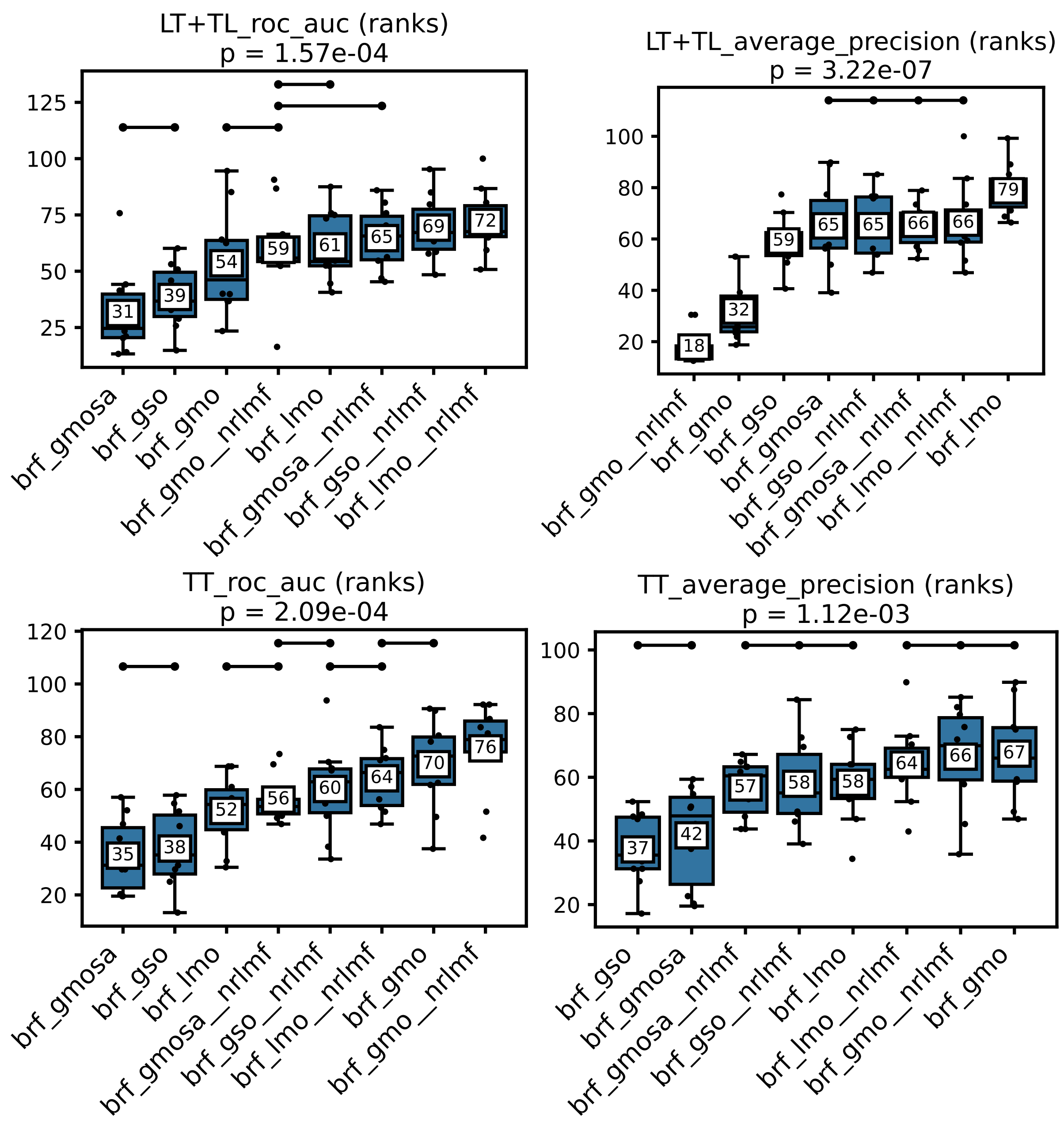


Figure 6 – Percentile rankings of prediction scores of extremely randomized tree ensembles under different adaptation strategies to bipartite interaction data, across all 10 datasets. The omnibus p-value obtained through a Friedman test is indicated below each plot's title. Significant results in each case lead us to perform pairwise Wilcoxon rank-sum tests, whose non-significant results are indicated by crossbars above each plot, that is, statistically undistinguished boxes are connected by the crossbars. The pairwise test results were corrected by the Benjamini-Hochberg procedure among each subfigure, considering that all possible estimator pairs were evaluated even if their comparison is not indicated in the plot. See **??** for further descriptions of each estimator.

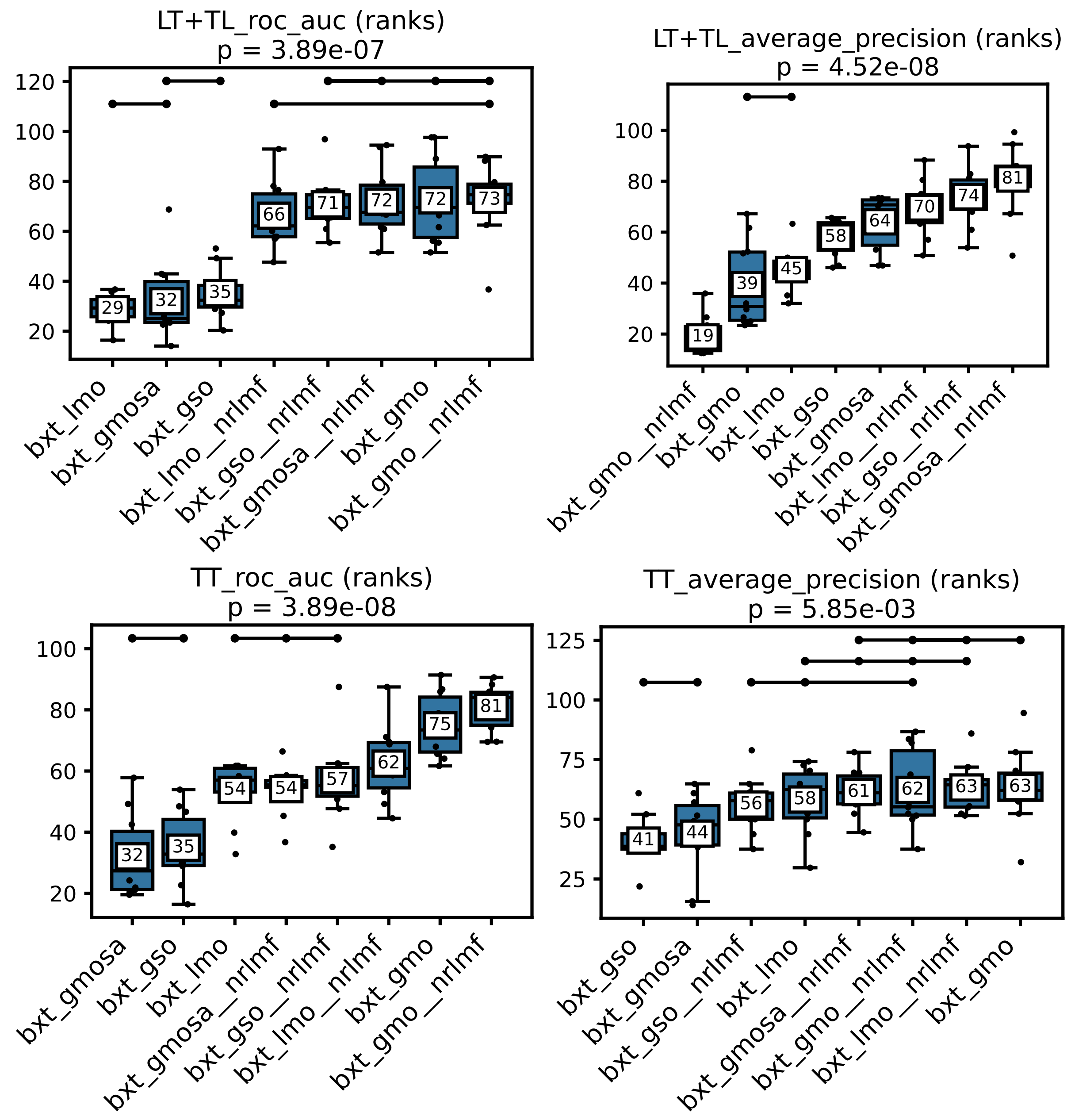


Figure 7 – Percentile rankings of prediction scores of extremely randomized tree ensembles under different adaptation strategies to bipartite interaction data, across all 10 datasets. The omnibus p-value obtained through a Friedman test is indicated below each plot's title. Significant results in each case lead us to perform pairwise Wilcoxon rank-sum tests, whose non-significant results are indicated by crossbars above each plot, that is, statistically undistinguished boxes are connected by the crossbars. The pairwise test results were corrected by the Benjamini-Hochberg procedure among each subfigure, considering that all possible estimator pairs were evaluated even if their comparison is not indicated in the plot. See **??** for further descriptions of each estimator.

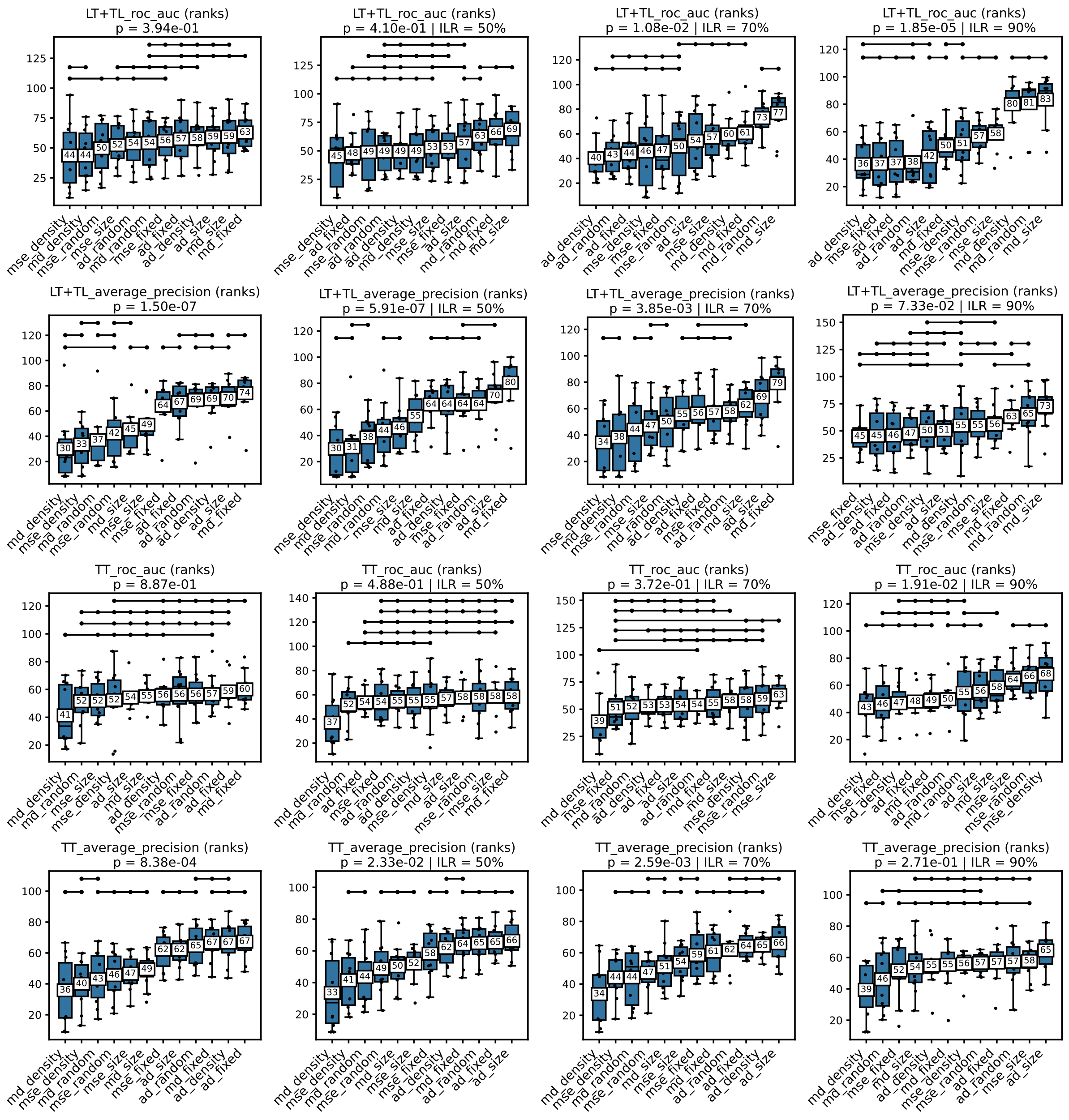


Figure 8 – Percentile rankings of prediction scores of extremely randomized tree ensembles under different adaptation strategies to bipartite interaction data, across all 10 datasets.

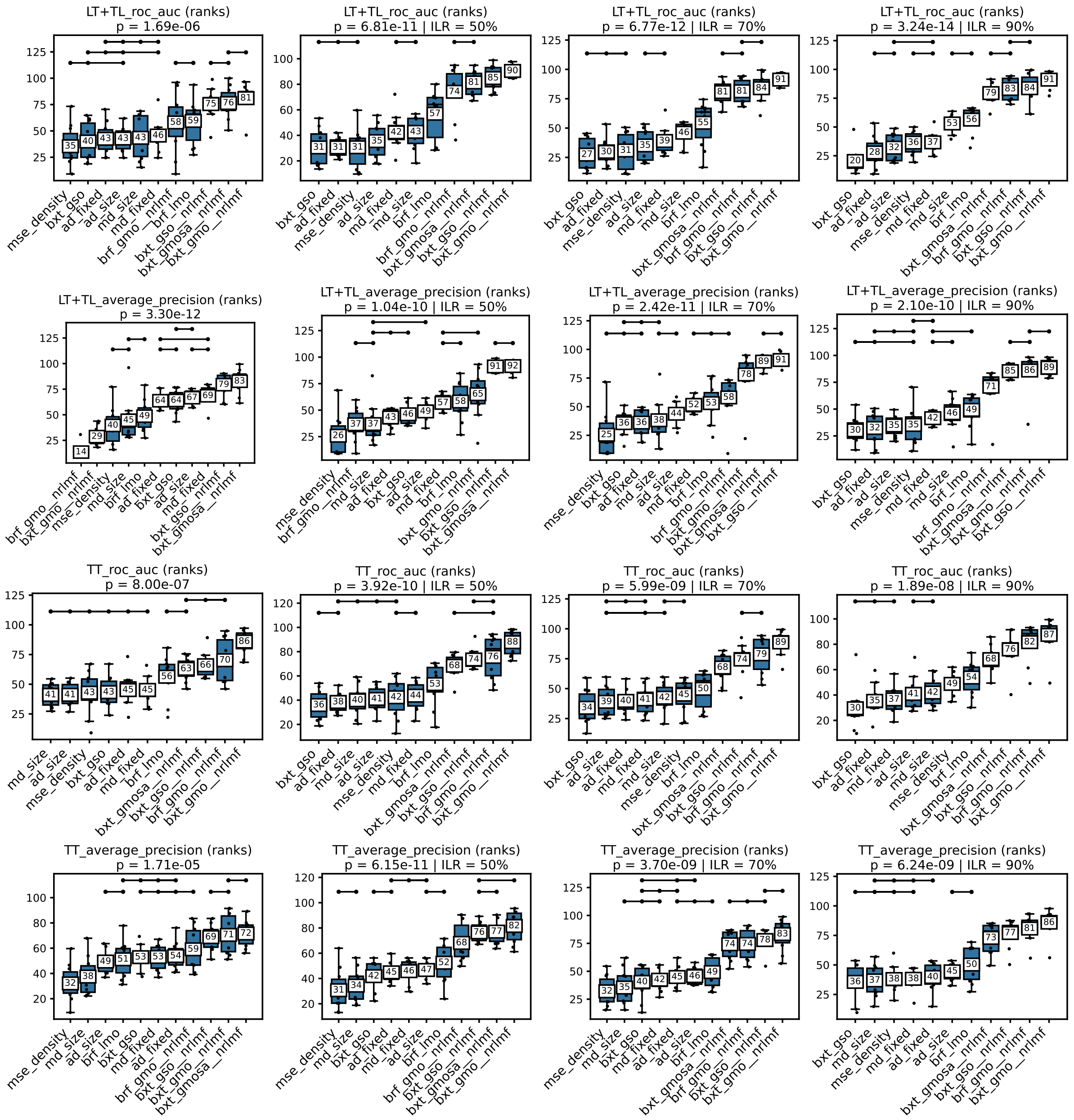


Figure 9 – Percentile rankings of prediction scores of extremely randomized tree ensembles under different adaptation strategies to bipartite interaction data, across all 10 datasets.

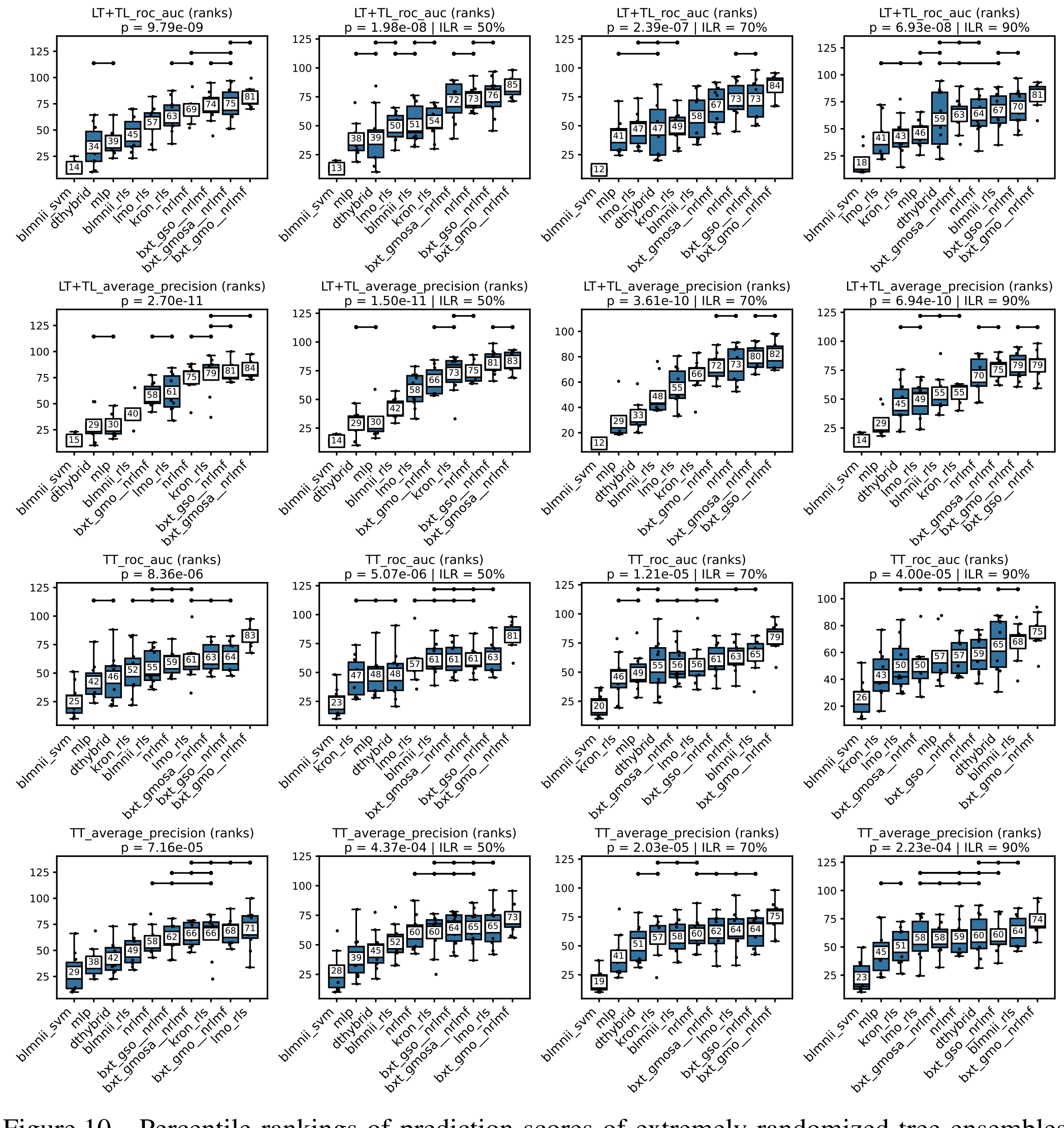


Figure 10 – Percentile rankings of prediction scores of extremely randomized tree ensembles under different adaptation strategies to bipartite interaction data, across all 10 datasets.

3 CONCLUSÃO

Apresentar as conclusões correspondentes aos objetivos ou hipóteses propostos para o desenvolvimento do trabalho, podendo incluir sugestões para novas pesquisas.

REFERENCES