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. 67p. 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. 67p. 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. 67p. 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.

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. 2.1.1).

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

When summing over all indices in a given dimension, we took the freedom of omitting the start and end positions (Eq. 2.1.1).

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

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 (Eq. 2.1).

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

The enclosing of indices within brackets also allows for the omission of parentheses when concomitantly using exponents, as exemplified by Eq. 2.2, 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 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.

2.2.1 DPI-E, DPI-G, DPI-I, DPI-N

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.

(1) further processed the dataset by removing all drugs and targets with less than 10 observations. In alignment with (**?**, 1), 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 (2). 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 (3). 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 (4).

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 (5, 6). 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 (4) 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.

2.3 Model evaluation protocol

2.3.1 Cross-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_g and Y_g, k-fold cross-validation consists in equally and randomly dividing both X_g and Y_g 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.

2.3.2 Prediction performance metrics

2.4 Applying monopartite estimators on bipartite data

As previously stated in subsection 2.1.2, 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 (7). 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.

2.4.1 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_g and a column-vector of labels we refer to as Y_g.

One way of doing so is by choosing indices as described by Eq. 2.3, where $\begin{bmatrix} x_1 & x_2 \end{bmatrix}$ denotes concatenation and all i_1 and i_2 combinations are explored (see **??**).

$$\begin{aligned} i_g &= (i_1 - 1)|X_2|_i + i_2 - 1 \\ X_g^{[i_g \cdot]} &= \begin{bmatrix} X_1^{[i_1 \cdot]} & X_2^{[i_2 \cdot]} \end{bmatrix} \\ Y_g^{[i_g 1]} &= Y^{[i_1 i_2]} \end{aligned} \tag{2.3}$$

Notice that, in order to consider the interactions of all possible pairs, X_g would have $|X_1|_i|X_2|_i$ rows and $|X_1|_j + |X_2|_j$ columns, with Y_g 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_g as defined by Equação 2.3 is highly redundant. As a

* 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.

result, naively dealing with such a large X_g 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 that 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 (**??**).

2.4.1.1 Previous work

2.4.2 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 LMO setting is as simple as training only the primary estimators, the primary rows estimator being trained on X_1 and Y, and the primary columns estimator on X_2 and $Y^\intercal$ (Function TrainLocalModel). 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 función PredictLocalModel and illustrated by **??**. The `combine` function used in line línea 15 of the función PredictLocalModel 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.

1 `train(`primaryRowsEstimator`,` X_1`,` Y`);`
2 `train(`primaryColumnsEstimator`,` X_2`,` $Y^\intercal$`);`
3 **return** primaryRowsEstimator, primaryColumnsEstimator

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 an 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 PredictLocalModel). 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.

Función PredictLocalModel(primary models, X_{new})

Input: The trained primary models and the unseen sample matrices X_{new} for both axes.
Output: Y_{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` $X_{2\text{new}}$

$Y_{\text{pred rows}} \leftarrow Y^{[\cdot j > |X_1|_i]}_{\text{pred rows}}$;

$Y_{\text{pred cols}} \leftarrow Y^{[\cdot j > |X_2|_i]}_{\text{pred cols}}$;

end

return `combine(`$Y_{pred\ rows}$, $Y^{\intercal}_{pred\ cols}$`)`

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 PredictLocalModel 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.

2.4.2.1 Previous work

2.5 Natively bipartite learning algorithms

2.5.1 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.

2.5.2 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 (8–10), 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 (8), 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.

2.5.3 Monopartite 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 networks inference, where insights into the underlying processes are greatly valued.

This final 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 section 2.1, 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 the last example, 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* (X_l, Y_l) and *right* (X_r, Y_r) partitions, as defined by Equação 2.4.

$$\begin{aligned} Y_l &= \{Y^{[k\cdot]} \; \forall \; k \mid X^{[kf]} \leq t\} \\ Y_r &= \{Y^{[k\cdot]} \; \forall \; k \mid X^{[kf]} > t\} \\ X_l &= \{X^{[kf]} \; \forall \; k \mid X^{[kf]} \leq t\} \\ X_r &= \{X^{[kf]} \; \forall \; k \mid X^{[kf]} > t\} \end{aligned} \tag{2.4}$$

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 pre-defined 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 Equação 2.4. 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 Equação 2.4 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 describe in subsection 2.5.4.

The exhaustive split search procedure is detailed by the Function FindSplitBest. The algorithm BuildTree then describes its use for growing a typical decision tree, while Predict explains how predictions are made given a model built by BuildTree and a new data instance.

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 13, is an interesting option when building tree ensembles (**??**), being the core idea behind the extremely randomized trees algorithm (11).

2.5.4 Split quality criteria and impurity metrics

The quality Q of a split is defined as the decrease of an impurity function $I(Y)$ calculated over the Y_l and Y_r partitions (Eq. 2.4) and averaged together, relative to the impurity of the Y_{node} parent's partition (Eq. 2.5). An additional $|Y_{\text{node}}|/|Y_{\text{root}}|$ factor is included, reducing the effect of nodes with smaller partitions which could introduce spurious variations of impurity. Notice that $|Y_{\text{node}}| = |Y_l| + |Y_r|$.

$$\begin{aligned} Q(Y, t, f) &= \frac{|Y_{\text{node}}|}{|Y_{\text{root}}|} \frac{I_{\text{node}}(Y) - I_{\text{post-split}}(Y)}{I_{\text{node}}(Y)} = \\ &= \frac{|Y_{\text{node}}|}{|Y_{\text{root}}|} \left[1 - \frac{|Y_l| I(Y_l) + |Y_r| I(Y_r)}{|Y_{\text{node}}| I(Y_{\text{node}})} \right] \end{aligned} \tag{2.5}$$

Several metrics can be chosen as the impurity function $I(\cdot)$. In this study we utilize the variance of each output column, averaged over all outputs (Eq. 2.6). Since the prediction values we return 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) of each output as if the node holding $Y_{\text{partition}}$ were a leaf.

$$I_{\text{MSE}}(Y) = (Y^{[ij]} - Y^{\langle i \rangle [j]})^{2\langle ij \rangle} = Y^{2\langle ij \rangle} - Y^{\langle i \rangle 2 \langle j \rangle} \tag{2.6}$$

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 Eq.

2.7 and culminates in the usual form of the average of Gini impurities across all outputs.

$$I_{\text{MSE}}(Y_{\text{bin}}) = Y_{\text{bin}}^{2\langle ij\rangle} - Y_{\text{bin}}^{\langle i\rangle 2\langle j\rangle} = (p^{[j]} - p^{[j]2})^{\langle j\rangle} \\ = [p^{[j]}(1-p^{[j]})]^{\langle j\rangle} = I_{\text{Gini}}(Y) \quad (2.7)$$

When the monopartite global adaptation Y_g is utilized (see Section 2.1), a global version of $I_{\text{MSE}}(\cdot)$ can be used to achieve the same result in the corresponding bipartite partition Y.

$$I_{\text{MSE}}(Y_g) = Y_g^{2\langle ij\rangle} - Y_g^{\langle i\rangle 2\langle j\rangle} = (Y_g^{\intercal})^{2\langle i\rangle} - (Y_g^{\intercal})^{\langle i\rangle 2} = \\ = Y^{2\langle ij\rangle} - Y^{\langle ij\rangle 2}$$

so that we can define I_{GMSE} as in Eq. 2.8.

$$I_{\text{GMSE}}(Y) \equiv I_{\text{MSE}}(Y_g) = Y^{2\langle ij\rangle} - Y^{\langle ij\rangle 2} \quad (2.8)$$

In such global scenario, the quality criteria can be rewritten as in Eq. 2.9.

$$Q_{\text{GMSE}}(Y, t, f) = \\ = \frac{|Y_{\text{node}}|}{|Y_{\text{root}}| I(Y_{\text{node}})} \left(\frac{|Y_l| Y_l^{\langle ij\rangle 2} + |Y_r| Y_r^{\langle ij\rangle 2}}{|Y_{\text{node}}|} - Y_{\text{node}}^{\langle ij\rangle 2} \right) \quad (2.9)$$

where we used that

$$1 - \frac{|Y_l| Y_l^{2\langle ij\rangle} + |Y_r| Y_r^{2\langle ij\rangle}}{|Y_{\text{node}}| I(Y_{\text{node}})} = 1 - \frac{\sum_i \sum_j Y_{\text{node}}^{\langle ij\rangle 2}}{|Y_{\text{node}}| I(Y_{\text{node}})} = \\ = 1 - \frac{Y_{\text{node}}^{2\langle ij\rangle}}{I(Y_{\text{node}})} = 1 - \frac{Y_{\text{node}}^{2\langle ij\rangle}}{Y_{\text{node}}^{2\langle ij\rangle} - Y_{\text{node}}^{\langle ij\rangle 2}} = \\ = \frac{-Y_{\text{node}}^{\langle ij\rangle 2}}{Y_{\text{node}}^{2\langle ij\rangle} - Y_{\text{node}}^{\langle ij\rangle 2}} = -\frac{Y_{\text{node}}^{\langle ij\rangle 2}}{I(Y_{\text{node}})}$$

Having only global averages in Eq. 2.8, i.e. always involving both i and j indices simultaneously, enables us to pre-compute averages of each row and column of Y_{node}, iterating over one-dimensional $\tilde{Y}_{\text{node}}$ proxies (Eq. 2.10) instead of the bi-dimensional matrix when searching for the best split. This property can be explored to build a more efficient training procedure for bipartite GSO decision trees in comparison to the naive approach (Section **??**), as discussed in the Section 2.5.5 and demonstrated in the asymptotic complexity analysis developed in Section 2.5.8.

$$\begin{aligned} \tilde{Y}_1^{[i]} &= Y^{[i]\langle j\rangle} \\ \tilde{Y}_2^{[j]} &= Y^{\langle i\rangle[j]} \end{aligned} \quad (2.10)$$

Dealing directly with bipartite data, another idea would be to take inspiration from the LMO strategy (see Section **??**) and define the quality of a node partition as $\frac{1}{2}[Q(Y_{\text{node}}, t, f) + Q(Y_{\text{node}}^{\intercal}, t, f)]$, the simple average between both directions. However, for a horizontal split, the impurity improvement on the columns axis is null for any impurity metric consisting of a simple average of impurities of each output, i.e. $I_{\text{total}}(Y) = (I(Y^{[ij]})^{[j]})^{\langle j\rangle}$, as briefly shown by Eq. 2.11 (which uses Eq. 2.5). As a consequence, the split search procedure with such impurities

on a bipartite dataset is essentially local, considering different outputs in only a single axis at a time.

$$
\begin{aligned}
I_{\text{total}}(Y^\intercal) = I(Y^{[ji]})^{\langle i \rangle} \implies \\
\implies |Y_l|_i I(Y_l^\intercal) + |Y_r|_i I(Y_r^\intercal) = \\
= \sum_i I(Y_l^{[ji]})^{[i]} + \sum_i I(Y_r^{[ji]})^{[i]} = \sum_i I(Y^{[ji]})^{[i]} \implies \\
\implies \frac{|Y_l|_i I(Y_l^\intercal) + |Y_r|_i I(Y_r^\intercal)}{|Y|_i} = I_{\text{total}}(Y) \\
\implies Q(Y^\intercal, t, f) = 0 \ \square
\end{aligned} \tag{2.11}
$$

This result is valid for the majority of multioutput decision tree implementations (), and leads us to simply define the LMO quality of a split on bipartite data as in Eq. 2.12, where f being a row feature means it represents a column of X_1 and, as such, a horizontal split. Otherwise, f is a column feature and designates a column of X_2, imposing a split in the vertical axis.

$$
Q_{\text{LMO}}(Y, t, f) = \begin{cases} Q(Y, t, f) & \text{if } f \text{ is a row feature} \\ Q(Y^\intercal, t, f) & \text{if } f \text{ is a column feature} \end{cases} \tag{2.12}
$$

2.5.5 Predictive Bi-Clustering Trees

(**?**) proposes an interesting method for building a decision tree directly on bipartite-formatted datasets. Given a general procedure `FindSplit` for finding a split threshold in a usual multioutput decision tree (see Section 2.5.3), 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 Eq. 2.12. Although each node locally performs the partitioning search on each axis, the overall tree is termed *Global MultiOutput* (GMO) by the original authors, and its training procedure corresponds to using `FindSplit` = FindBipartiteSplit in BuildTree.

2.5.6 Bipartite GSO trees

Although being developed specifically to interaction data, the PBCT algorithm shows no improvement in training time complexity relative to the naive GSO approach, as derived in the Section 2.5.8. However, in Eq. 2.9 we demonstrate how a single-output impurity metric can be used directly in the bipartite setup. Being single-output, no distinction is made among Y columns or rows, minimizing label deviance relative to the global average $Y^{\langle ij \rangle}$ instead of column averages $Y^{\langle i \rangle [j]}$. Considering such metric enables further optimization of the split searching procedure by employing single-column proxies of the interaction matrix (Eq. 2.10), as also described by the Function FindBipartiteSplit and justified in Section 2.5.8.

The bipartite decision trees grown with this procedure on X_1, X_2 and Y have the exact same structure as a usual monopartite decision tree trained on a dataset X_g and Y_g adapted with the monopartite global strategy (Section **??**). This property can be intuitively shown by noticing that X_1 and X_2 already contain all the information necessary to define a set of thresholds that yields all possible partitions, since each partition considers a single X column and building X_g does not removes or adds different elements to each of them (only repeats them). This result is a consequence of Eq. 2.3 and is formally stated by Eq. 2.13, in which we assume $f_2 = f_g - |X_1|_j$.

$$
\begin{aligned}
\{Y_g^{[k1]} \ \forall \ k \mid X_g^{[kf_g]} < t\} = \\
= \begin{cases} \{Y^{[i_1 i_2]} \ \forall \ i_1 \text{ and } i_2 \mid X_1^{[i_1 f_g]} < t\}, & \text{if } f \leq |X_1|_j \\ \{Y^{[i_1 i_2]} \ \forall \ i_1 \text{ and } i_2 \mid X_2^{[i_2 f_2]} < t\}, & \text{if } f > |X_1|_j \end{cases}
\end{aligned} \tag{2.13}
$$

2.5.7 Bipartite prediction strategies

With monopartite datasets, the `prototype` function (line 9 of Predict) most often simply returns the average label of the leaf's partition (Eq. 2.14).

$$\texttt{prototype}\,(Y_{leaf}) = Y_{\text{leaf}}^{\langle ij \rangle} \tag{2.14}$$

Nevertheless, some nuances may be appear when dealing with bipartite data, since there are cases in which one sample domain, row or column samples, is already known from the training set. As introduced by (**?**), 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 and x_2, we can set `prototype` as in Eq. 2.15.

$$\texttt{prototype}\,(Y_{leaf}) = \begin{cases} Y_{\text{leaf}}^{[k]\langle j \rangle} & \text{if } x_1 = X_{1\text{ leaf}}^{[k\cdot]} \\ Y_{\text{leaf}}^{\langle i \rangle[k]} & \text{if } x_2 = X_{2\text{ leaf}}^{[k\cdot]} \\ Y_{\text{leaf}}^{\langle ij \rangle} & \text{otherwise.} \end{cases} \tag{2.15}$$

However, specially when working with very unbalanced interaction matrices and sufficiently small leaf partitions, this approach may be more prone to random fluctuations, since the label averages in the prediction step are took over a much smaller sample size. We thus propose a softer approach: to weight the rows and columns of Y_{leaf} by similarity measures in the form $w_{s1}^{[i]} \equiv \text{similarity}(x_1^{[i]}, X_1^{[i\cdot]})$ between x and the training samples in the leaf node (Eq. 2.16).

$$\begin{aligned} \texttt{prototype}\,(Y_{leaf}) = \\ = \frac{\sum_{i \in \text{leaf}} w_{s1}^{[i]} Y_{\text{leaf}}^{[i]\langle j \rangle}}{2\sum_{i \in \text{leaf}} w_{s1}^{[i]}} + \frac{\sum_{j \in \text{leaf}} w_{s2}^{[j]} Y_{\text{leaf}}^{\langle i \rangle[j]}}{2\sum_{j \in \text{leaf}} w_{s2}^{[j]}} \end{aligned} \tag{2.16}$$

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. $w_s = e^{x^{[i]}}$

2.5.8 Asymptotic complexity analysis

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

$$\begin{aligned} O(FindSplitBest) = O(\tilde{n}_f S(|Y|_i) + \tilde{n}_f |Y|_i |Y|_j)) = \\ = O(\tilde{n}_f |Y|_i + \tilde{n}_f |Y|_i |Y|_j) = \\ = O(\tilde{n}_f |Y|_i |Y|_j) \end{aligned} \tag{2.17}$$

where $S(n)$ is the complexity of the chosen sorting algorithm when operating on n values. Since sorting of multiple subsets of the same X values will be performed, if X is real-valued, it is often effective to spend $O(|X|_j |X|_i \log |X|_i)$ time previously obtaining ranks for each of X's columns and for each axis, before generating X_g, so that the following partition sorting can be performed in linear time with integer-specific algorithms

such as Radix Sort (). That said, we thereafter consider $S(n) = O(n)$, and the pre-sorting term of the complexity will be disregarded in favor of the asymptotically dominant tree building complexity described ahead.

When applied on X_g and Y_g as in the naive GSO approach, we have Eq. 2.18.

$$O(FindSplitBest) = O(\tilde{n}_f|Y_g|_i|Y_g|_j) = O(\tilde{n}_f|Y_g|_i) = \\ = O(\tilde{n}_f|Y|_i|Y|_j) \quad (2.18)$$

For FindBipartiteSplit employing the GMO approach, the time complexity is given by Eq. 2.19, which renders it equivalent to the naive GSO strategy. $\tilde{n}_{fr}$ and $\tilde{n}_{fc}$ stand for the numbers of row and column features to be drawn, respectively.

$$O(FindBipartiteSplit_{\text{GMO}}) = \\ = O((\tilde{n}_{fr} + \tilde{n}_{fc})|Y|_i|Y|_j) \quad (2.19)$$

When considering the native GSO approach however, $\tilde{Y}_1$ and $\tilde{Y}_2$ column vectors are used instead of Y, effectively eliminating the `for` loops in lines 6 and 13 of FindSplitBest. 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 Eq. 2.20.

$$O(FindBipartiteSplit_{\text{GSO}}) = \\ = O(|Y|_i|Y|_j + \tilde{|}X_1|_j|Y|_i + \tilde{|}X_2|_j|Y|_j) \quad (2.20)$$

For the whole tree building process, considering $|Y|_i \propto |Y|_j := n_s$ and given any constant $k \in \mathbb{N}$ so that $O(\text{FindSplit}) = n_s^k$, we analyze the case of maximum relative tree width, where the number of samples in each node ($|Y| = n_s^2$) halves with each level added so that $T(n_s) = 2T(n_s/\sqrt{2}) + O(\text{FindSplit}(n_s))$, with $T(n_s)$ being the time took to build a node with n_s row instances and n_s column instances. Eq. 2.21 demonstrates the expected overall training complexity for the considered FindSplit functions (as given by the Master Theorem ()). Table 1 summarizes the last results for the different cases.

$$O(BuildTree) = O\left(\sum_{i=0}^{\log_2 n_s^2} 2^i \text{FindSplit}(n_s/\sqrt{2}^i)\right) = \\ = O\left(\sum_{i=0}^{\log_2 n_s^2} 2^i n_s^k/2^{ki/2}\right) = O\left(n_s^k \sum_{i=0}^{\log_2 n_s^2} 1/2^{(k/2-1)i}\right) \\ = \begin{cases} O(n_s^k) & \text{if } k > 2 \\ O(n_s^2 \log n_s) & \text{if } k = 2 \\ O(n_s^2) & \text{if } k < 2 \end{cases} \quad (2.21)$$

We can observe that, if the pre-sorting step is utilized, no asymptotic complexity improvement is expected for the GMO approach in comparison to the original naive GSO. Insofar, to the extent of our knowledge this essential step is not mentioned in previous works, so that we here reaffirm its importance and encourage future studies to take similar preprocessing procedures into more attentive consideration.

It is also seen that our proposed optimization for the GSO strategy reduces the tree building complexity by a factor of $\log n_s + \tilde{n}_f/(\tilde{n}_f \log n_s) = 1/\tilde{n}_f + 1/\log(n_s)$ in comparison to GMO, a major method improvement especially for datasets with high number of features. The use of similarity or kernel matrices, for instance, as the datasets explored in this work do, are greatly facilitated, and unseen scalability of decision tree-based learning methods is unlocked for interaction problems in general.

Regarding the performance of 13 as a substitute for FindSplitBest, although considerable amounts of operations are saved, the computational time needed by the algorithm is expected to grow with no different ratio, with no improvements in asymptotic complexity. This is due to the possibility of calculating Q iteratively as shown by FindSplitBest.

Strategy	Split search	Tree building	$\tilde{n}_f \propto n_s$
GMO	$O(\tilde{n}_f n_s^2)$	$O(\tilde{n}_f n_s^2 \log(n_s))$	$O(n_s^3 \log(n_s))$
GSO	$O(n_s^2 + \tilde{n}_f n_s)$	$O(n_s^2 \log(n_s) + \tilde{n}_f n_s^2)$	$O(n_s^3)$

Table 1 – Asymptotic time complexity comparison between the global multiple outputs and global single output approaches for Decision Tree building. n_s designates the number of samples in each axis, assumed to be similar between them. $\tilde{n}_f$ represents the number of features to be considered for split search in each node. The last column refers to the case where the number of features considered in each node is proportional to n_s the number of row or column samples in it. This scenario could arise, for instance, if one is dealing with pairwise features and would want to consider only intrapartition similarities.

2.5.9 Tree ensembles

A longstanding idea is that combining predictions of multiple machine learning models yields better results, much like averaging opinions of several people aids in decision taking (). In fact, it is well demonstrated that the generalization error for a group of weak learners asymptotically decreases with a higher number of individual estimators (). Due to their simplicity and transparency, decision trees are frequently chosen as individual learners to compose an ensemble of estimators.

Many strategies are possible to combine predictions of multiple models in a ensemble, one of the simplest of them being a majority voting system. With this approach, each individual estimator's prediction is considered a vote on the class to be outputted and the most voted class is regarded as the final prediction (). Nevertheless, several other methods have been explored, namely weighted voting () and (**?**). In regression tasks, the individual predictions could be simply combined by taking the average output value of them as the whole ensemble final guess ().

An important result by (**?**) was that the strength of the total ensemble model not only depends on the strength of the individual estimators but also on the correlation between them, so that reducing correlation between the individual components increases performance overall. Multiple ideas were then developed to generate a set of uncorrelated estimators to be further combined. () proposes to grow each tree on a bootstrap set of samples data, in which a predefined number of samples are drawn with replacement from the original dataset, so that they are equally distributed to the total samples. The number of samples drawn usually equals the total number of samples, making each bootstrap set also the same size as the original set. Since samples are chosen with replacement, each set lacks about a third of the original input data **??**, yielding thus distinct trees unaware of the whole dataset. This procedure is currently know as *bagging*, and was proposed by **??**.

Taking another step in reducing individual trees correlation, (**?**) proposed to, besides bootstrapping samples (X_a's rows) before creating a new tree, also to subsample $\tilde{\lfloor} X_a |_j$ features ($X_a j$'s columns) at each tree split, this time without replacement, defining one the most widely used machine learning algorithms today, the Random Forests (). The feature subsampling enables much faster training in comparison to Bagging or AdaBoost, their contemporaneous counterparts, but maintaining competitive prediction scores (**?**). Furthermore, their popularity might also stem from a low need for data preprocessing and hyperparameter tuning, making Random Forests easily configurable models especially suited for unstructured tabular data (). A typical choice is $\tilde{\lfloor} X_a |_j = \lceil \sqrt{\lfloor} X_a |_j \rceil$.

An even more aggressive randomization approach was proposed by (**?**), named *ExtraTrees*, from extremely randomized trees. In each node, instead of searching for the overall best split threshold t, ExtraTrees first draw a random t_f between the minimum and maximum values for each of $\tilde{n}_f$ randomly chosen feature columnns.

The best t_f and its corresponding feature, among the $\tilde{n}_f$ selected ones, is then returned. The higher randomization dispenses the use of sample bootstraping and turns the process of finding a split search a $O(\tilde{n}_f \tilde{n}_s)$ procedure, rather than Random Forests' .

In both Random Forests or ExtraTrees, the tree components are usually grown to their maximum size, without pruning or using early-stopping parameters.

Since PBCTs can be generated with the exact structure as common decision trees, the same ensemble techniques are possible for these models, with very small modifications regarding data sampling. (**?**) explores the use of ExtraTrees ensemble of PBCTs for drug-target interaction prediction, obtaining favorable results in comparison to other methods. However, superiority of Random Forests are often verified (), and no previous work was found to explore this algorithm.

We thus present an implementation for Random Forests of PBCTs, to which we suggest the name Biclustering Random Forests (BRF). Similarly, we thereafter call ExtraTrees of PBCT by Biclustering Extra Trees (BXT). The procedures to build Random Forests and BRFs are described by algorithms **??** and **??**, respectively.

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.

Q^*, f^*, $t^* \leftarrow$ `FindSplit` (X, Y, $\tilde{n}_f$);
`// Many stopping criteria are possible`
if `DecideToStop(`Q^*, f^*, t^*, X, Y`)` **then**
    return NodeObject {
        isLeaf ← True
        $X_{\text{leaf}} \leftarrow X$
        $Y_{\text{leaf}} \leftarrow Y$
    };
else
    Get X_l, Y_l, X_r, Y_r from f^* and t^* (Eq. 2.4);
    return NodeObject {
        isLeaf ← False
        childLeft ← `BuildTree(`X_l, Y_l`)`
        childRight ← `BuildTree(`X_r, Y_r`)`
        feature ← f^*
        threshold ← t^*
    };
end

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.

Node ← RootNode;
while *Node is not a leaf* **do**
    if $x^{[Node.feature]} > Node.threshold$ **then**
        Node ← Node.childRight
    else
        Node ← Node.childLeft
    end
end
return `prototype(`*Node.X, Node.Y*`)`

Função 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^*.

Initialize S_r and S_l as a $|Y|_j$-sized vectors;
Q^*, f^*, $t^* \leftarrow \mathbf{0}$;
Draw $\tilde{n}_f$ columns (features) of X without replacement;
foreach *feature index* f *of the* $\tilde{n}_f$ *drawn features* **do**
    $n_l \leftarrow 0$; // Holds $|Y_l|_i$
    $S_r \leftarrow \sum_i Y^{[i\hat{\jmath}]}$; // Holds $\sum_i Y_r^{[i\hat{\jmath}]}$
    $S_l \leftarrow \mathbf{0}$; // Holds $\sum_i Y_l^{[i\hat{\jmath}]}$
    Get the permutation P that sorts $X^{[\cdot f]}$;
    Apply P to Y's and $X^{[\cdot f]}$'s rows:
        Y_P, $X_P \leftarrow P(Y)$, $P(X^{[\cdot f]})$;
    foreach *row index* $\hat{\imath}$ *of* Y_P **do**
        $n_l \leftarrow n_l + 1$;
        foreach *column index* $\hat{\jmath}$ *of* Y_P **do**
            $S_r^{[\hat{\jmath}]} \leftarrow S_r^{[\hat{\jmath}]} - Y_P^{[\hat{\imath}\hat{\jmath}]}$;
            $S_l^{[\hat{\jmath}]} \leftarrow S_l^{[\hat{\jmath}]} + Y_P^{[\hat{\imath}\hat{\jmath}]}$;
        end
        Use S_l, S_r and n_l to calculate Q (Eq. 2.9). Notice that other node-specific constants might be needed;
        if $Q > Q^*$ **then**
            $Q^* \leftarrow Q$;
            $f^* \leftarrow f$;
            $t^* \leftarrow \frac{1}{2}(X_P^{[\hat{\imath}]} + X_P^{[\hat{\imath}+1]})$;
        end
    end
end
return Q^*, f^*, t^*;

2.5.10 Implementation details

2.5.11 Dataset

In this work, we use data from Drug-Target interactions in the experimental validation of the proposed model. DTI is an area of the literature that has been the focus of several recent advances and consists of methods for predicting interactions between drugs and targets (Proteins, Diseases, Ligands). This area has already been investigated in several applications present in the literature, such as (**?**, **?**). In this context, the dataset used is the Drug-Protein Interaction Networks, defined in (**?**). This dataset consists of four bipartite interaction networks between proteins and drugs: Ion channels (DPI-I), Nuclear receptors (DPI-N), protein-coupled receptors G (DPI-G), and Enzymes (DPI-E). Both networks of interactions form datasets, and the interaction prediction in these contexts and applying them to the real world can bring innovations and discoveries.

Función FindSplit$_{\text{random}}(X, Y)$

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^*.

$Q^*, f^*, t^* \leftarrow \mathbf{0}$;
Draw $\tilde{n}_f$ columns (features) of X without replacement;
foreach *feature index f of the $\tilde{n}_f$ drawn features* **do**
  Find $\texttt{min}(X^{[\cdot f]})$ and $\texttt{max}(X^{[\cdot f]})$;
  Draw a random threshold value $t \in \mathbb{R}$ so that $\texttt{min}(X^{[\cdot f]}) < t < \texttt{max}(X^{[\cdot f]})$;
  Calculate Q for the drawn t (Eq. 2.9) ; // $O(|Y|)$
  if $Q > Q^*$ **then**
    $Q^* \leftarrow Q$;
    $f^* \leftarrow f$;
    $t^* \leftarrow \frac{1}{2}(X_P^{[i]} + X_P^{[i+1]})$;
  end
end
return Q^*, f^*, t^* $1_{\texttt{FindSplit}_{\texttt{random}}(I)}$ $\texttt{FindSplit}_{\texttt{random}}$

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^*.

if adapterStrategy *is GSO* **then**
  // Build Y proxies $\tilde{Y}_1$ and $\tilde{Y}_2$ (Eq. 2.10)
  $\tilde{Y}_1 \leftarrow Y^{[\cdot]\langle j \rangle}$;
  $\tilde{Y}_2 \leftarrow Y^{\langle i \rangle[\cdot]}$;
else
  // Using GMO strategy, no proxies are used
  $\tilde{Y}_1 \leftarrow Y$;
  $\tilde{Y}_2 \leftarrow Y^{\intercal}$;
end
// Generate a split in each axis. Get each split's position, feature and quality score
$Q_r^*, f_r^*, t_r^* \leftarrow \texttt{FindSplit}(X_1, \tilde{Y}_1)$;
$Q_c^*, f_c^*, t_c^* \leftarrow \texttt{FindSplit}(X_2, \tilde{Y}_2)$;
if $Q_r^* > Q_c^*$ **then**
  return Q_r^*, f_r^*, t_r^*
else
  // f_c^* value lets clear its X_2 ownership
  return Q_c^*, f_c^*, t_c^*
end

2.5.12 Code and data availability

2.6 Results and Discussion

2.6.1 General experimental settings

2.6.2 Empirical time complexity analysis

To empirically access 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 1. 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 2.5.8, 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 their GMO counterparts (see the caption for Figure 1).

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, which can be done much faster than the search for the best split employed by the greedy decision trees, eventhough 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 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.

2.6.3 Comparison between GSO models

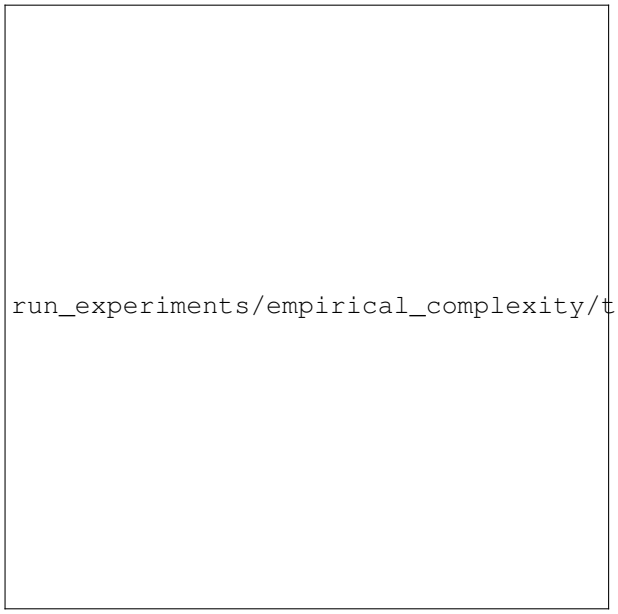
To access the impact of global single-output optimizations in bipartite decision tree growing, we compare three slightly different training methods for BXT and BRF models.

- **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 3), eventhough 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,

(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 tree algorithms. bxt stands for bipartite ExtraTree, bdt for a bipartite decision tree using the greedy split search procedure. Independent two sample t-tests comparing the slope estimates in (b) reveal that the time complexity of bdt_gso is highly significantly lower than bdt_gmo (p-value $< 10^{-37}$) and also that bxt_gso significantly exibits lower complexity than bxt_gmo (p-value $< 10^{-19}$). Those values corroborate the theoretical estimates from Section 2.5.8.

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,

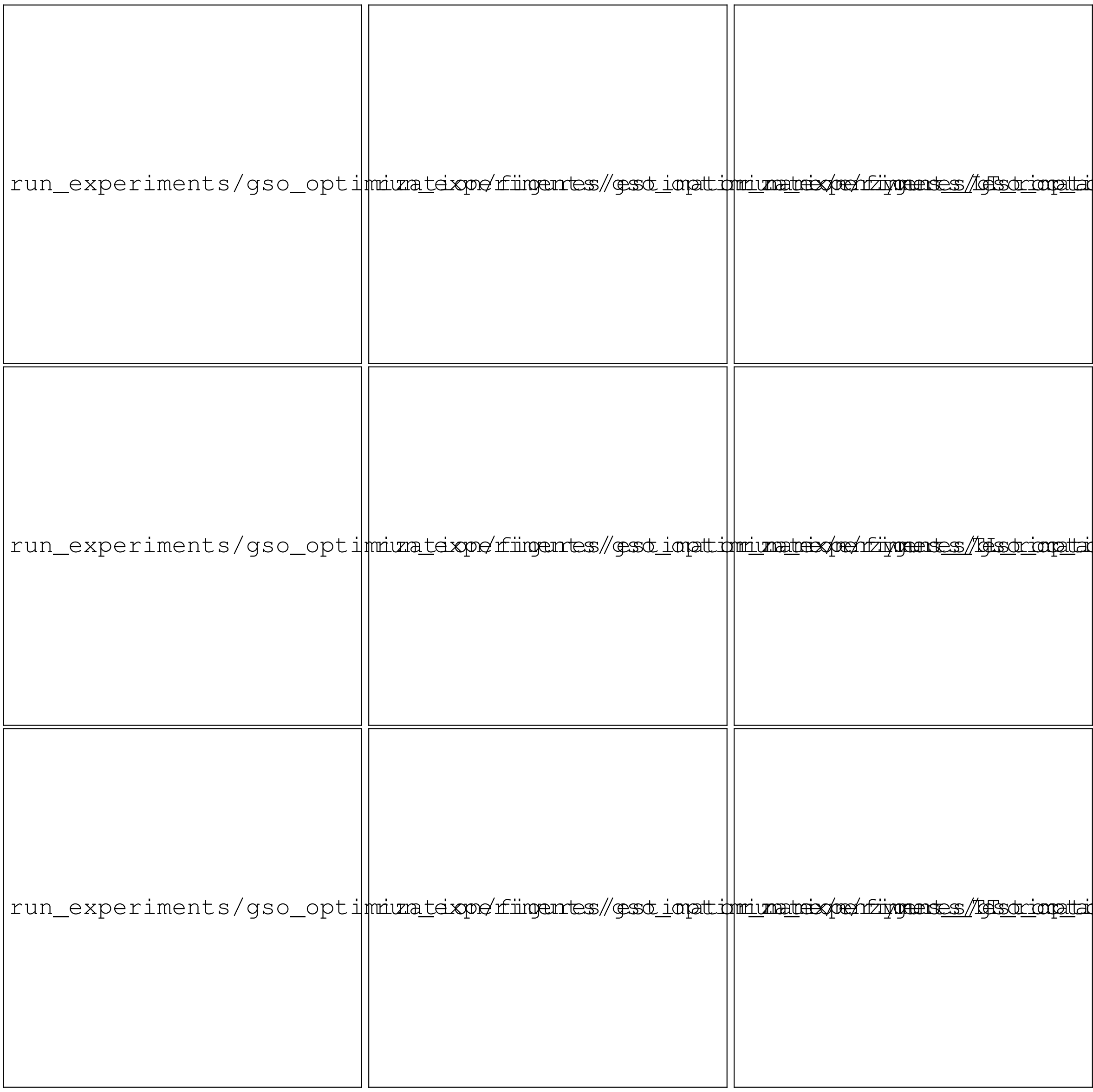


Figure 2 – Comparison between GSO models on the Enzymes dataset.

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.

2.6.4 Comparison between GMO prediction weights

In order to compare the different prediction weighting strategies (see Section **??**), a BXT and a BRF model were built for every option. The minimum rows per leaf and minimum columns per leaf were both set to 5, ensuring a minimum number of co-leaf samples of each sample domain to be taken into consideration in a

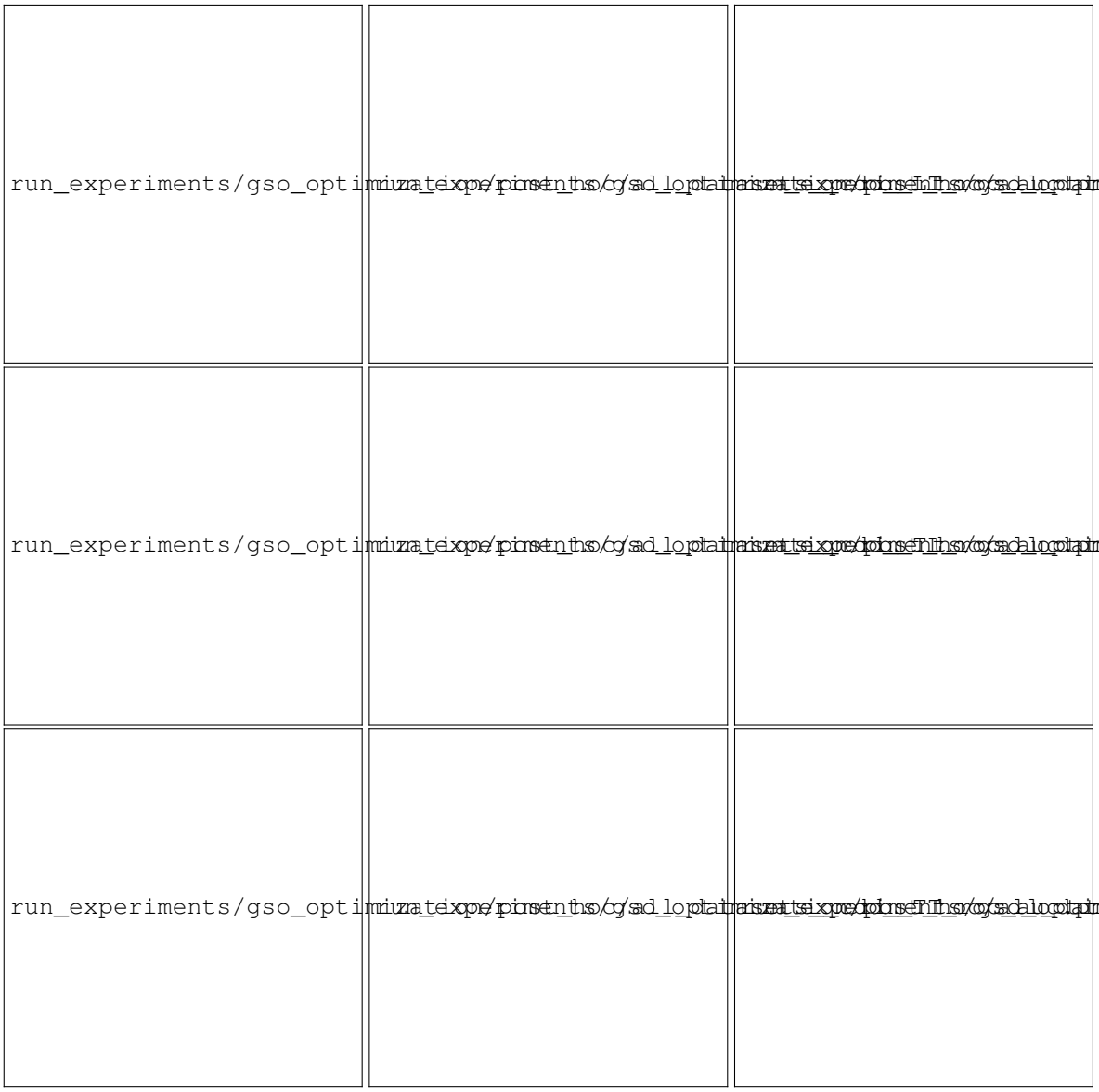


Figure 3 – Percentile score rankings for each global single output strategy.

weighted-neighbors fashion during evaluation (Section **??**).

As shown by Figure **??** and Table **??**, BXT models show an overall superior performance in comparison to BRF models, with each BXT model scoring significantly higher than its BRF counterpart with the same prediction weights. Furthermore, the weighted GMO predictions seem to prevail relative to the leaf-wise prototype GMOSA (Section **??**). Specifically, bxt_square significantly outperforms all other bipartite forests except for bxt_precomputed, both in terms of AUROC and average precision (AUPR) in the TT sets (Figure **??**).

Hence, the bxt_square model was selected for the downstream analyses, keeping 5 by 5 as the minimum leaf partition size. While GMOSA will also be further investigated, the leaf size constraint will be dropped, generating fully grown trees for this strategy.

2.6.5 Comparison between adaptation strategies

Still restricted to forest estimators, we compare each of the described approaches for adapting models to bipartite data (Section **??**). To avoid differences in random sampling when using the naive GSO adapter versus the natively bipartite GSO tree, no bootstraping was applied to any forest, providing all trees with the whole training

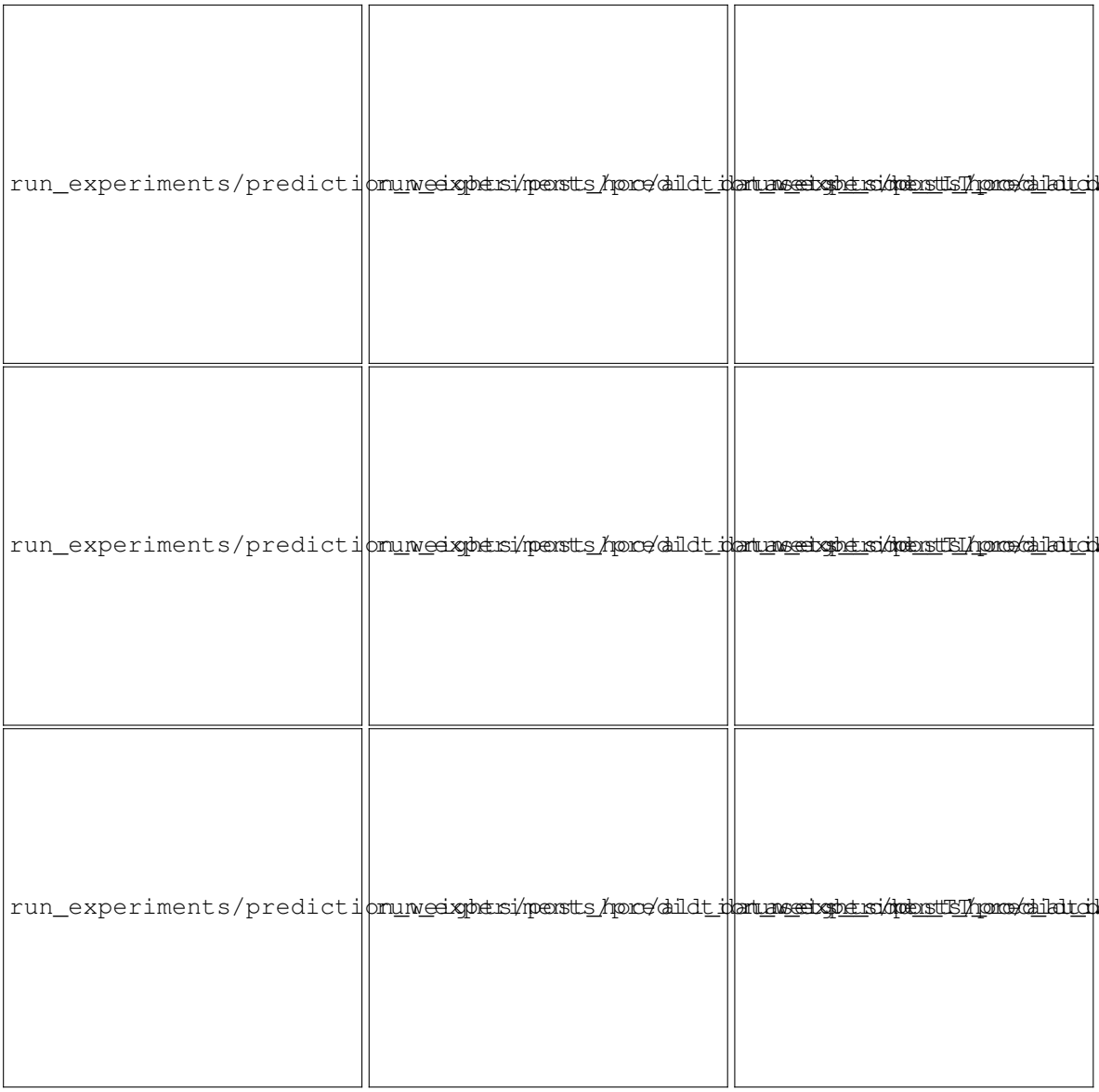


Figure 4 – Percentile score rankings for each prediction weighting strategy.

samples space. To still ensure randomization in random forest estimators, the maximum features parameter was set to 0.5, meaning that each tree in a random forest was trained on a random subset of half the features from each sample domain. Due to implementation details, this means the naive GSO forests will sample features slightly differently: they will pick half the features from the whole feature space combined, while the natively bipartite GSO forests will ensure half the features from each sample domain is selected. This is not expected to have a significant impact on the results, given that the total number of features is especially high in the present scenario, where similarity matrices are being employed.

All forests were composed of 100 tree estimators and were fully grown, with the exception of the GMO models, whose leaf sizes were limited to a minimum of 5 by 5 samples (at least 5 samples from each domain) in order to take advantage of neighborhood weighting (which was set to the squared similarities, see Section **??**).

In all of the evaluated scenarios, a BXT model was ranked the best. In both TT-MCC and TL-AP, the BRF models and bxt_gmo were significantly surpassed by the remaining BXT forests. In TL-MCC, bxt_lso significantly outperformed all the other models. Given that this test-set provides the greatest intersection with the training set, due to the overall higher number of column samples in the datasets we used, we suggest that the LSO model

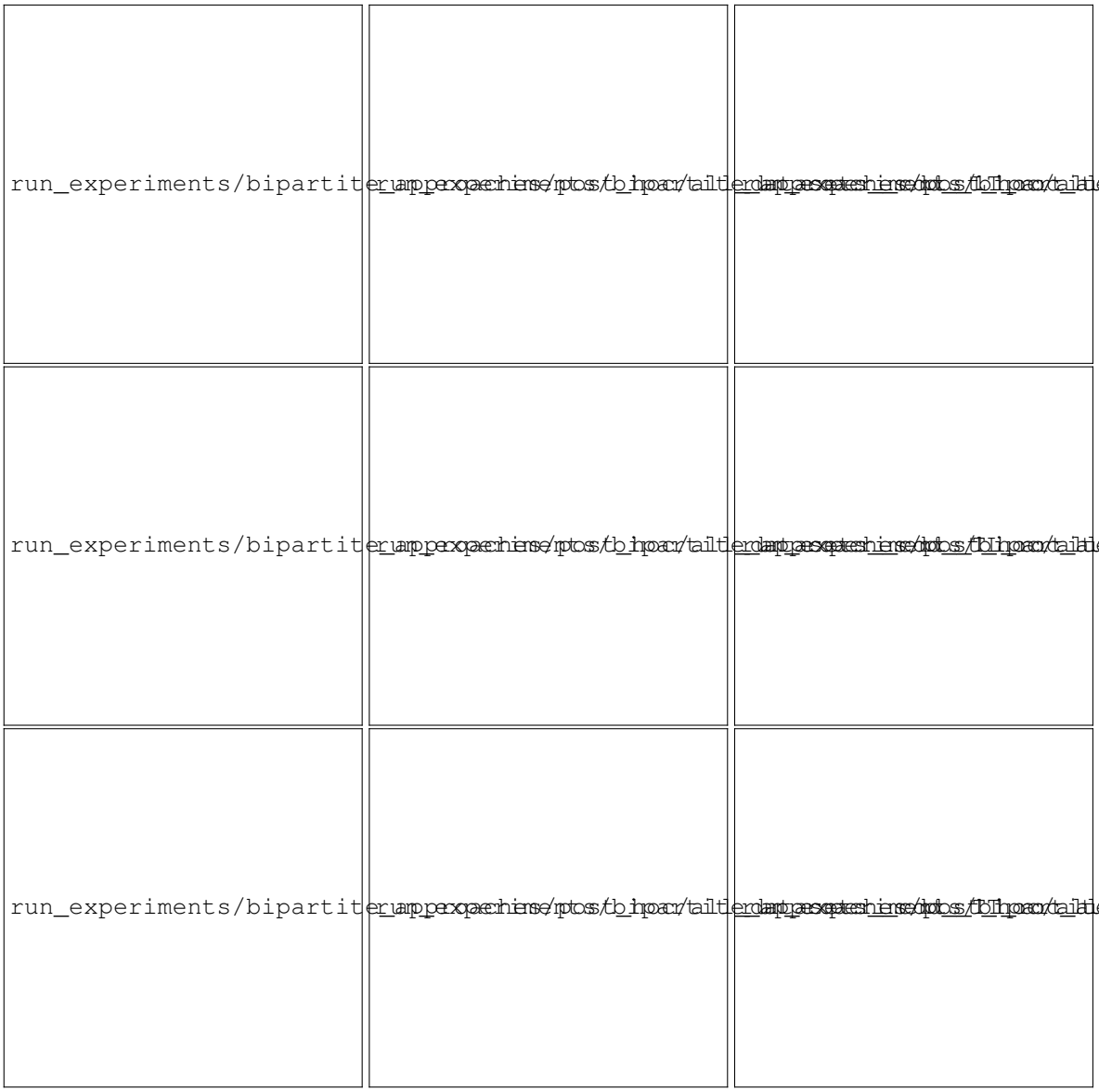


Figure 5 – Percentile score rankings for each bipartite adaptation approach.

could be better at taking advantage of already seen information from a sample domain, since a forest is grown separately for each row and column. From another perspective, this effect could be regarded as a form of overfitting. This hypothesis is further supported by the fact that both GMO models, which were the only forests subject to a minimum leaf size constraint and thus less prone, in theory, to overfitting, were the most frequently outperformed models in the TL-MCC and TL-AP.

Contrastingly, both GMO models were consistently the best performing models in all AUROC settings, which is commonly assumed to be a less indicative metric in highly unbalanced classification contexts such as the present one (). This could be explained by a bias towards the majority class, possibly caused by the averaging of the larger leaves employed in this methods.

2.6.6 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)(). The results are shown in Figure **??**.

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 were set to be equal among noth 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 are shown in Figure **??**.

In accordance to the previous findings of (**?**), the models with output space reconstruction either show significantly higher AUROC and AUPR or generated inconclusive results regarding those metrics when compared to the models without the reconstruction by NRLMF.

Surprisingly, however, the MCC results tend to favor the opposite conclusion, with the models without reconstruction showing higher MCC scores in most cases.

A first explanation could be that the NRLMF hyperparameter search was not exhaustive enough, and that a better combination of hyperparameters could have been found. However, we later show in section **??** that the NRLMF model alone displays competitive performance, disfavoring such hypothesis of underfitting.

We also notice that (**?**) performs bipartite cross-validation in unusual fashion, by replacing with zeros the positive labels of pairs assigned to the test set but still using them to train the model. Albeit test labels are masked, each model thus still receives all available samples during training, and we hypothesize that unsupervised information from the test set could possibly still be exploited during training. For instance, estimating the sample density of the feature space could provide an importance score, a weight factor, for each sample, in order to favor correct predictions of denser clusters and undermine outliers. Wether and how this or similar mechanisms are explored by the NRLMF algorithm is out of the scope of this work, but we discourage the use of such CV strategy and point it as a possible reason for the higher NRLMF scores observed by previous authors (**?**).

2.6.7 Comparison with previous works

In this section we employ several algorithms previously described in the literature, all of which were reimplemented and had their performance measured on the four DTI prediction datasets according to this study's evaluation framework (Section **??**).

The algorithms being considered in this section are listed below, and their scoring results are shown in Figure 7 and Table **??**. To further explore the potential of bipartite forests, all forests evaluated in this section were composed of 1000 trees, as opposed to the 100 trees used in the previous experiments.

- NRLMF (**?**)
- BLM-NII

Regarding the AUROC metric, bxt_gmosa_nrlmf, bxt_gso_nrlmf and nrlmf are consistently the three highest ranked models in all three testing settings, with both bxt_gmosa_nrlmf and bxt_gso_nrlmf significantly out-

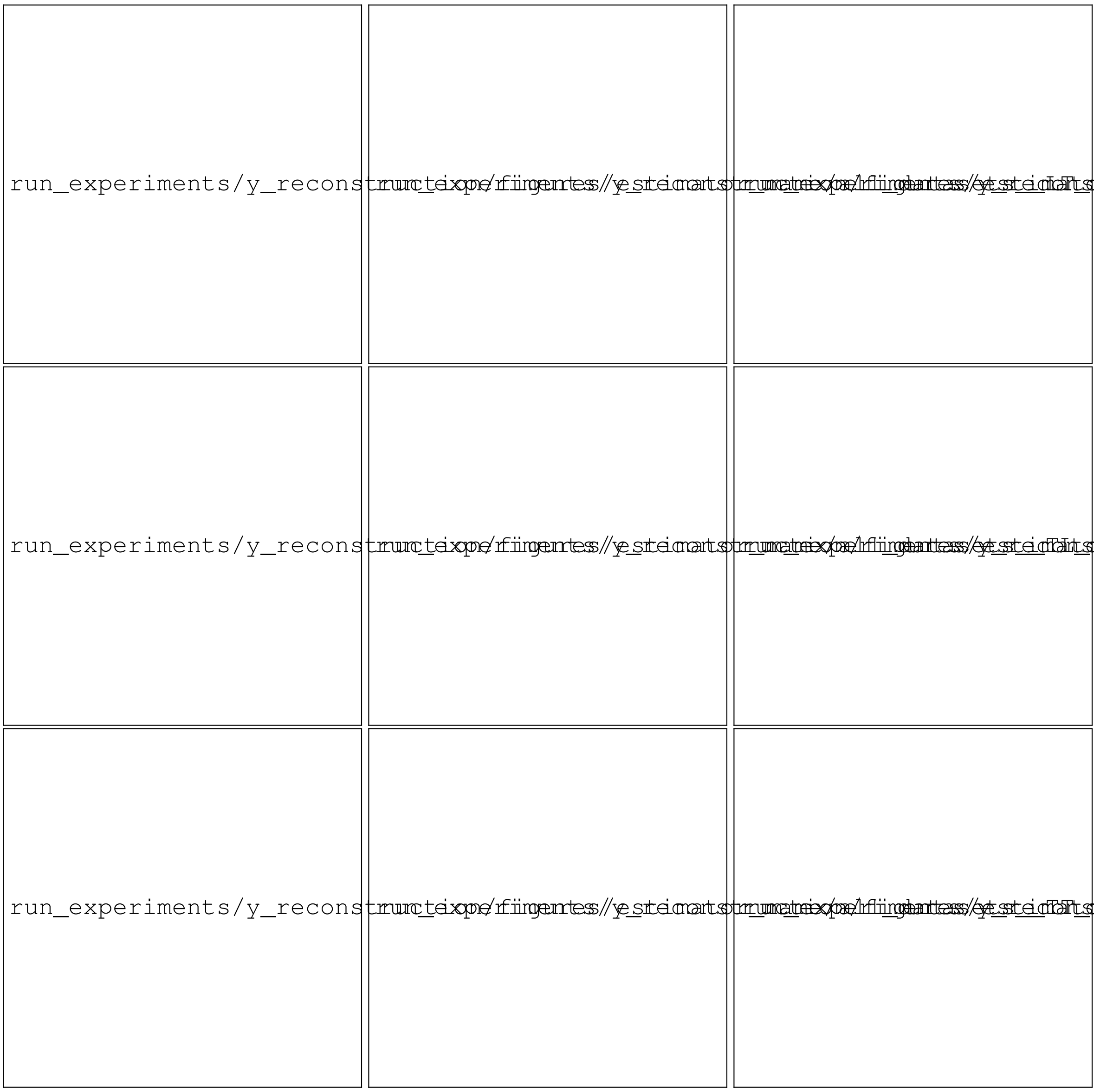


Figure 6 – Comparison of scores for the bipartite forests with and without output space reconstruction on the enzymes dataset.

performing BLM-NII-SVM, BLM-NII-RLS, and DTHybrid. Additionally, DNILMF was outperformed by bxt_-gmosa_nrlmf and bxt_gso_nrlmf in the TL and LT settings, and by bxt_gmosa_nrlmf in the TT setting.

Both in the LT and TT settings, the bipartite forests without matrix reconstruction (bxt_gmosa and bxt_-gso) showed no significant difference in performance when compared to their counterparts employing interaction matrix reconstruction by NRLMF (bxt_gmosa_nrlmf and bxt_gso_nrlmf, respectively), contrary to the results on the TL testing configuration, where matrix reconstruction was shown to be significantly beneficial. A possible explanation relies on the fact that the TL test set has the higher intersection with the training data in our specific scenario, since in the four datasets considered the drug molecules outnumber the protein targets by considerable margins (see Table **??**). That said, NRLMF fundamentally relies on the label information of a small neighborhood of each sample to infer interactions and, as such, this result may suggest that the benefit of the matrix factorization step is elevated in cases where the test set is similar to the training set, and could be specifically useful in drug repurposing scenarios rather than in drug discovery. On the other hand, overfitting may arise as a potential concern when new drugs and new targets are of the main interest.

Additionally, one must recall we are using larger forests in comparison to the previous section, which

may render the matrix factorization step less relevant. This is further supported by the fact that the NRLMF model itself could not significantly outperform bxt_gso in the LT_roc_auc setting, bxt_gmosa in the TL_roc_auc setting and also could not outperform either forest in the TT_roc_auc setting (Figure 7), not ruling out the explanation that the tree ensemble itself is sufficient to encompass neighborhood information.

When the average precision metric is considered, the reconstruction step is again regarded unbeneficial, with no significant difference in employing it versus using the bipartite forest alone. Interestingly, all BXT forests perform significantly better than all other methods in the TL_average_precision setting, even than the matrix factorization techniques NRLMF and DNILMF (Figure 7). NRLMF, DNILMF and lmorls were the only methods not proven significantly inferior to the BXT in the remaining average precision settings.

The superiority of DNILMF with respect to NRLMF as claimed by its authors () was not observed in our experiments, with NRLMF even proven significantly superior to DNILMF in the TL_roc_auc and LT_roc_auc settings (Figure 7).

With respect to the MCC metric, the bxt_gmosa and bxt_gso stand out as the best performing models, significantly outperforming all the other models in TL_mcc, all but bxt_gmosa_nrlmf in LT_mcc, and all but bxt_gmosa_nrlmf and bxt_gso_nrlmf in the TT setting.

However, this result is likely affected by the classification threshold choice, since this is the only metric we used that is threshold-dependent. Since we chose this threshold as the probability of interaction in each training set (the average of all binary values of y), the MCC metric will favor well calibrated models, i.e. whose predicted probabilities are close to the measured propabilities. As such, this result mainly indicates better calibration of the BXT models out of the box. Conversely, the other models may benefit from calibration techniques such as Platt scaling () or isotonic regression (), with BLM models being especially underperformant regarding MCC.

Overall, bxt_gso and bxt_gmosa are the most consistently higher ranked models among the algorithms we tested. Remarkably, we demonstrate in sections 2.5.8 and 2.6.2 that our optimized GSO forests are considerably faster to train than the GMO forests proposed by (**?**), even if no statistically significant decrease in performance is observed. This time complexity superiority enables forest estimators to tackle much larger bipartite datasets than was possible with the current bipartite trees, and points native GSO bipartite forests as a strong candidate to be further studied across similar learning scenarios in the future.

2.6.8 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 8, 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 2.5.8 and 2.6.2).

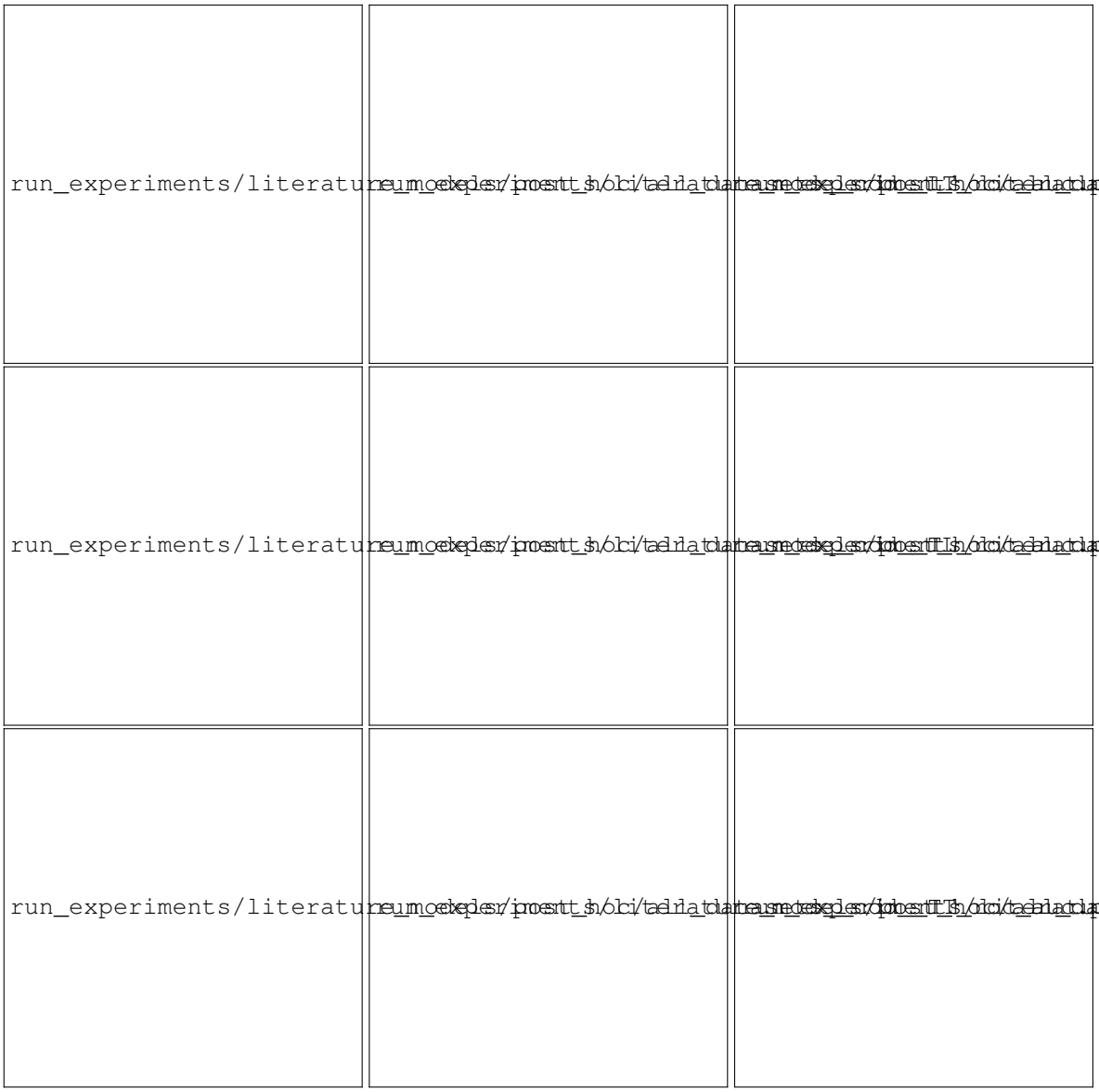


Figure 7 – Percentile score rankings for several literature models.

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.

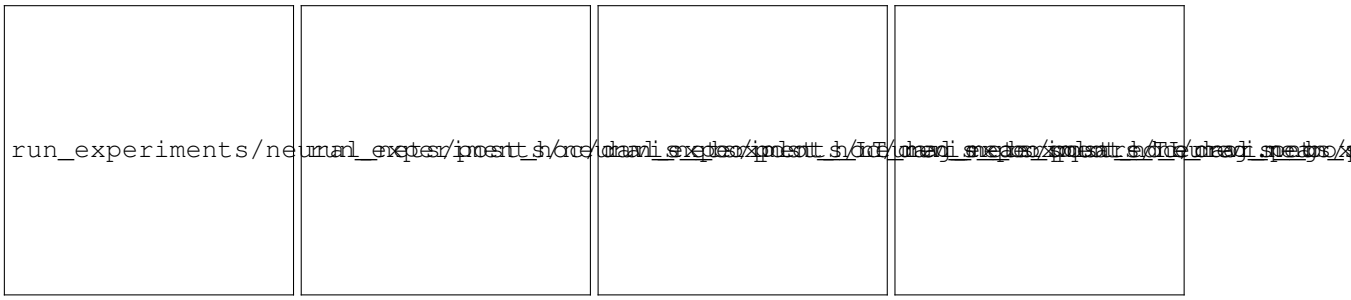
2.6.9 Estimated impact of missing labels

2.7 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.

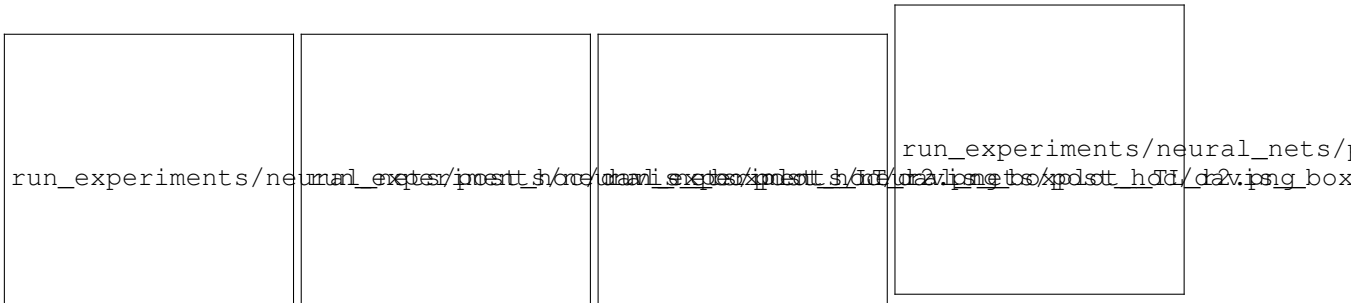


(a) Negative mean squared error on the LT test set.

(b) Negative mean squared error on the TL test set.

(c) Negative mean squared error on the TT test set.

(d) Training duration in seconds.



(e) R^2 score on the LT test set.

(f) R^2 score on the TL test set.

(g) R^2 score on the TT test set.

(h) Prediction duration in seconds.

Figure 8 – Model performance on the DAVIS dataset.

3 CONCLUSÃO

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

REFERENCES

1 HE, T. *et al.* SimBoost: A read-across approach for predicting drug–target binding affinities using gradient boosting machines. BioMed Central, v. 9, n. 1, p. 1–14.

2 HUANG, K. *et al.* DeepPurpose: A deep learning library for drug–target interaction prediction. Oxford University Press, v. 36, n. 22-23, p. 5545–5547.

3 FRANKISH, A. *et al.* GENCODE 2021. Oxford University Press, v. 49, n. D1, p. D916–D923.

4 COCK, P. J. *et al.* Biopython: Freely available Python tools for computational molecular biology and bioinformatics. Oxford University Press, v. 25, n. 11, p. 1422.

5 WU, T. *et al.* NPInter: The noncoding RNAs and protein related biomacromolecules interaction database. Oxford University Press, v. 34, p. D150–D152.

6 TENG, X. *et al.* NPInter v4. 0: An integrated database of ncRNA interactions. Oxford University Press, v. 48, n. D1, p. D160–D165.

7 VERT, J.-P. **Reconstruction of Biological Networks by Supervised Machine Learning Approaches**. Available at: http://arxiv.org/abs/0806.0215.

8 PLIAKOS, K.; GEURTS, P.; VENS, C. Global multi-output decision trees for interaction prediction. Springer, v. 107, p. 1257–1281.

9 PLIAKOS, K.; VENS, C. Network inference with ensembles of bi-clustering trees. Springer, v. 20, p. 1–12.

10 PLIAKOS, K.; VENS, C. Drug-target interaction prediction with tree-ensemble learning and output space reconstruction. Springer, v. 21, p. 1–11.

11 GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. Springer, v. 63, p. 3–42.

APPENDIX

APPENDIX A – APÊNDICE(S)

Elemento opcional, que consiste em texto ou documento elaborado pelo autor, a fim de complementar sua argumentação, conforme a ABNT NBR 14724 (**?**).

Os apêndices devem ser identificados por letras maiúsculas consecutivas, seguidas de hífen e pelos respectivos títulos. Excepcionalmente, utilizam-se letras maiúsculas dobradas na identificação dos apêndices, quando esgotadas as 26 letras do alfabeto. A paginação deve ser contínua, dando seguimento ao texto principal. (**?**)

APPENDIX B – EXEMPLO DE TABELA CENTRALIZADA VERTICALMENTE E HORIZONTALMENTE

A Tabela 2 exemplifica como proceder para obter uma tabela centralizada verticalmente e horizontalmente.

Table 2 – Exemplo de tabela centralizada verticalmente e horizontalmente

Coluna A	**Coluna B**
Coluna A, Linha 1	Este é um texto bem maior para exemplificar como é centralizado verticalmente e horizontalmente na tabela. Segundo parágrafo para verificar como fica na tabela
Quando o texto da coluna A, linha 2 é bem maior do que o das demais colunas	Coluna B, linha 2

Fonte: Elaborada pelos autores.

APPENDIX C – EXEMPLO DE TABELA COM GRADE

A Tabela 3 exemplifica a inclusão de traços estruturadores de conteúdo para melhor compreensão do conteúdo da tabela, em conformidade com as normas de apresentação tabular do IBGE.

Table 3 – Exemplo de tabelas com grade

Coluna A	**Coluna B**
A1	B1
A2	B2
A3	B3
A4	B4

Fonte: Elaborada pelos autores.

ANNEX

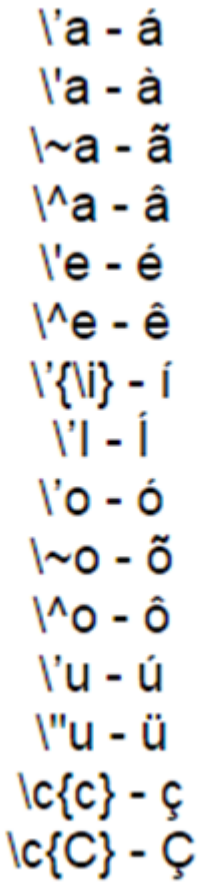
ANNEX A – EXEMPLO DE ANEXO

Elemento opcional, que consiste em um texto ou documento não elaborado pelo autor, que serve de fundamentação, comprovação e ilustração, conforme a ABNT NBR 14724. (**?**).

O **ANEXO B** exemplifica como incluir um anexo em pdf.

ANNEX B – ACENTUAÇÃO (MODO TEXTO - LaTeX)

Figure 9 – Acentuação (modo texto - LaTeX)



Fonte: **?**

INDEX