Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Input-Output Kernel Regression applied to protein-protein interaction network inference

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Maycas Nadal

Tutores: Jana Kludas y Óscar Pastor López

2013/2014

Resumen

El estudio de las redes de interacción de proteínas ha recibido una gran atención por parte de la comunidad investigadora en los últimos años. Sin embargo, los estudios experimentales para la reconstrucción de este tipo de estructures son caros. Consecuentemente, varios métodos de aprendizaje automático para inferir redes de interacción de proteínas han sido desarrollados. En este trabajo presento la implementación y el análisis del Input-Output Kernel Regression (IOKR) desarrollado por [9, 10] para llevar a cabo la inferencia utilizando varios conjuntos de datos experimentales. IORK está basado en el aprendizaje de un kernel de salida que nos permita aplicar modelos de regresión en un espacio de características donde podemos calcular la similitud de pares de proteínas para inferir la existencia de interacción. Además, esta aproximación extiende el Kernel Ridge Regression a una aproximación semi-supervisada donde la inferencia se convierte en completar una red. La técnica de aprendizaje de múltiples kernels es aplicada en los datos de entrada para tratar las diferentes fuentes de datos. Finalmente, comparo el rendimiento de la implementación con otras aproximaciones supervisadas para la inferencia de redes de interacción de proteínas.

Palabras clave: proteína, inferencia de redes, interacción, output kernel, regresión, aprendizaje de kernel, inferencia, Saccharomyces cerevisiae

Abstract

The study of protein-protein interaction networks has received a lot of attention by the research community lately. However, the experimental studies to reconstruct this kind of structures are expensive. Consequently, several machine learning approaches have been developed that automatically infer PPI networks. In this work I present the implementation and analysis of the Input-Output Kernel Regression (IOKR) developed by [9, 10] to compute the inference using various experimental data sets. IOKR is based on the learning of an output kernel that let us apply regression models on a feature space where we can compute the similarity of pairs of proteins to infer the existence of interactions. Furthermore, this approach extends the Kernel Ridge Regression to a semi-supervised approach where the inference turns into a matrix completion. The Multiple Kernel Learning is applied on the input side to deal with the different data sources. Finally, I compare the performance of the implementation with other supervised approaches for the inference of PPI networks.

Keywords: protein, network inference, interaction, output kernel, regression, kernel learning, Saccharomyces cerevisiae

Table of contents

Abbreviations and Acronyms

PPI	Protein-protein interaction
IOKR	Input-Output Kernel Regression
OK3	Output Kernel Tree
MKL	Multiple Kernel Learning
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve

1. Introduction

1.1. Motivation

Nowadays, the understanding of biological networks is one of the major challenges on the study of the systems biology. These structures comprise among others protein-protein networks, metabolic pathways and gene regulatory networks.

The knowledge extracted from this kind of biological structures has many applications. Drug production can be improved with a better level of knowledge of the protein interactions of a living cell, leading to produce better drugs. Furthermore, the interaction between proteins can be used to annotate proteins based on the properties of their neighboring proteins in the network. Also, the understanding of metabolic pathways helps to understand how biological processes are performed in an organism, for instance degradation or synthesis.

With the appearance of new high-throughput technologies for analysis of biological material, such as next generation sequencing techniques, the amount of experimental data have highly increased. This fact demands new approaches to analyze huge amounts of data with reasonable and feasible computational time and space.

1.2. Problem statement

Currently, the amount of experimental data of biological networks is still not enough to reconstruct most of these structures. Moreover, the extraction of this data in wet experiments in a laboratory is a difficult task that implies high costs.

Because of that, several machine learning approaches have been developed to infer the structure of biological networks. The inference of this kind of networks can be seen as a classification problem, so machine learning techniques can be used in order to solve it [5].

The aim of the machine learning models is the classification of each link of the network. In the case of PPI networks, it is a binary classification task because we look for the existence or absence of an interaction between two proteins.

The inference of a biological structure, in this case a PPI network, consists of training a model using some kind of input data in order to be able to predict the labels of the links of the network. As said above, the amount of well-known protein-protein interactions is low, therefore, this setup can help to correct existing data and find new interactions.

Several machine learning models can be used for this purpose. I have selected the Input-Output Kernel Regression because it has shown in previous applications a better performance and it requires less computational time and space [9, 10].

1.3. Objectives

The goal of this project is the implementation and testing of the machine learning approach based on Kernel Ridge Regression named Input-Output Kernel Regression (IOKR) [9, 10] for the inference of protein-protein interaction networks. Multiple Kernel Learning is applied to combine the different input data sources to get a better performance.

The implementation has been done in MATLAB. Experiments have been carried out in order to tune the parameters of the model and the validation with curated data of the PPI interaction network of the yeast *Saccharomyces cerevisiae*'s protein secretory machinery [1]. Moreover, I compare the performance of our model with other implementations on the inference of PPI networks.

Therefore, specific tasks were defined in order to clarify the different stages of the project:

1. Review the state-of-the-art methods for the inference of PPI networks using machine learning.
2. Implementation of the Input-Output Kernel Regression (IOKR) as given in [9, 10] from a given MATLAB code developed in the work [9].
3. Testing the model with data from the yeast *Saccharomyces cerevisiae*. Different experiments have been carried out: parameter tuning, supervised vs. semi-supervised performance and Multiple Kernel Learning (MKL) vs. no MKL performance.
4. Comparing the results of the IOKR with other state-of-the-art methods.

1.4. Structure of the project

This work is structured as follows. First, in Section 2, I am going to present the background of the studied problem, which is the inference of protein-protein interaction networks. Moreover, I describe other state-of-the-art methods that have been used to solve the problem and the regression methods, which are the basis of the model I have used.

In Section 3, once I have presented the background of our project I describe the methods that I have used. The section is divided in two main parts. First, I describe the Input-Output Kernel Regression which is used for inferring the interaction labels. Then, I introduce the Multiple Kernel Learning for improving the fusion of the data sources on the input side.

Section 4 covers the experimental setup which includes the parameter tuning. I present the experiments that are used to analyze the performance of the classifier and the comparison between the different settings of the model. Furthermore, I compare the performance of IOKR with other state-of-the-art methods.

In Section 5, I discuss the experimental results from the previous section. Finally, I summarize the work of the project in the conclusions in Section 6.

2. Background

In the following sections, I describe the necessary background to understand the implemented model. First, I explain the concept of PPI networks. Later, I give a brief description of the kernel trick used in high dimensional spaces. Then, I briefly describe previous approaches for the inference of graphs. Finally, I introduce the Kernel Ridge Regression, which is the basis of the IOKR. These concepts will help the reader to understand the Input-Output Kernel Regression.

2.1. Protein-protein interaction networks

Proteins are large biological molecules made of a chain of amino acids held together by peptide bonds. They are the most abundant biological material in a cell, almost 50% of it. These molecules are the product of transcription and translation of DNA (Figure 1). The DNA sequence determines the amino acids that are produced when it is read. Proteins are versatile molecules, meaning that the same protein can show different amino acid chains [2].

This kind of molecule is involved in the cell functionality, performing different biological functions depending on the protein type. For example, enzymes are in charge of catalyzing chemical reactions in the cells, for instance accelerating or delaying them. Furthermore, there are antibodies which are the defenses of the body against foreign invasions, as viruses or bacteria, and transport proteins in charge of moving molecules around the organism [2].

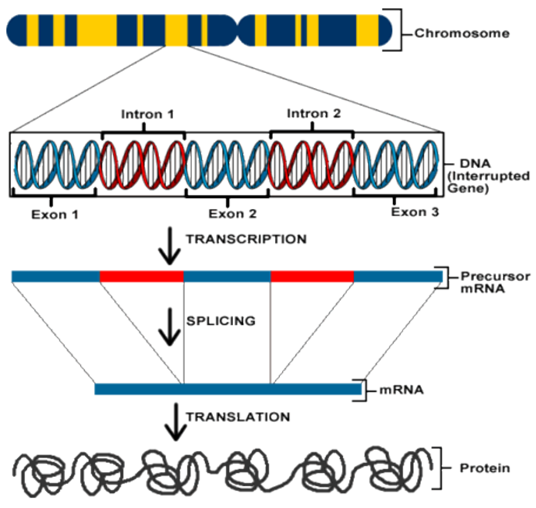

Figure 1: Process for generating the proteins [17]

Changes in the protein production or protein properties can have external effects. The lack or the excess of a certain protein can lead to a disease in the organism [2]. For instance, the Proteinuria, which is the excess of the serum protein, can cause kidney diseases. On the other hand, some changes in a protein can lead to health benefits, in such cases the mutation is known as evolution.

A protein-protein interaction (PPI) network defines the physical and functional contacts between a set of proteins in a cell or a living organism. These interactions are not with other molecules such as DNA, RNA or ligands. The interactions needed for basic functionality of the proteins, such as production or degradation, are not included in this kind of structure [2]. Figure 2 shows an example of a PPI network where the protein TMEM8A is involved.

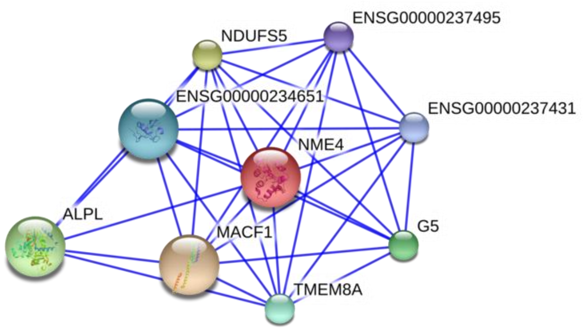

Figure 2: Protein Interaction Network for TMEM8A in humans (2013) [18]

The analysis of PPI networks is part of the field of study called interactomics [3]. Interactomics is the study of the interactions among proteins and between proteins and other molecules. The mapping of all the interactions of a living being is called interactome.

As described in [3], several experimental techniques have been developed to measure biological interactions in the laboratory. For example, the yeast two-hybrid system allows the identification of physical interactions between proteins under in vivo conditions using a bay-prey system. There exist other experimental methods such as the Affinity purification-mass spectrometry (APMS).

2.2. The Kernel Trick

Basic Machine Learning methods model the input output relations linearly. However, real problems tend to be more complex and require high dimensional representation of the data. As described in [4], we can use kernels to avoid working in such high dimensional spaces.

In inference tasks we have the domain $\mathcal{X}$ which is represented by a nonempty set of inputs x_i (predictor variables) and the domain $\mathcal{Y}$ that represents the targets (response variables). In Machine Learning the aim is to predict the target $y \in \mathcal{Y}$ of an unseen input instance $x \in \mathcal{X}$.

The y is selected by choosing a pair (x, y) similar to the training instances [4]. Consequently, we need to measure the similarity between instances in the domains X and Y. A kernel $k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ can be used as the similarity measure. For all $x, x' \in \mathcal{X}$, a kernel satisfies:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle$$

Where $\phi: \mathcal{X} \rightarrow \mathcal{H}$ is a mapping to a dot product space $\mathcal{H}$. This space $\mathcal{H}$ is an infinite dimensional product space, usually high dimensional, sometimes called feature space. This property defined for the input domain can be transferred to the output domain $\mathcal{Y}$.

The equation above is known as the Kernel Trick. Consequently, we can compute the similarity of two instances by the evaluation of a kernel instead of computing it as a dot product in a high dimensional feature space [4].

2.3. Graph inference

A protein-protein interaction network can be seen as an undirected graph, where each vertex represents a protein of the network [5, 6, 16]. In this graph, there is an edge between two vertices if the proteins interact.

Let $G = (V, E)$ be an undirected graph that represents a PPI network. The finite set of vertices $V = (v_1, v_2, \ldots, v_n)$ is the set of proteins of an organism. The set of edges $E \subset V \times V$ defines how the vertices of the graph are connected, which means how the proteins of the network interact with each other [5]. A feature vector $x(v_i)$ is provided for each protein v_i of the network.

The graph inference can be considered as a pattern recognition problem, due to the fact that we can assign a label value y_i to an edge that defines whether two vertices are connected [4]. An edge will be labeled with 1 if its vertices represent two proteins that interact with each other in the PPI network, otherwise, the edge will be marked with 0.

As a result, I am interested in learning a model that can predict if two proteins interact. As I have already mentioned, many Machine Learning approaches have been developed to solve this problem. We can divide them in two groups, unsupervised and supervised inference models. Unsupervised methods consist of inferring the labels of the edges directly from data of the proteins without using the data from the labeled edges. For this purpose several techniques have been used like probabilistic methods such as Bayesian networks or dynamical system equations [5].

As I explained in the previous section, various experimental techniques have provided well-known interactions and non-interactions that can be used for performing supervised machine learning. Supervised approaches aim for training a binary classifier using the given labeled edges as training set to infer unknown edges in a PPI network. Let S be the training set for a supervised model, then $S = ((e_1, y_1), (e_2, y_2), \ldots, (e_p, y_p))$ where $e_i \in E$ is an edge of the graph, y_i is the label of the edge and $p < |E|$.

In this work I am focusing on supervised methods, where the Input-Output Kernel Regression is included, because it has been shown that these methods outperform the unsupervised models [16]. In the following sections I review two general supervised models for the graph inference based on local models and global models as described in [5] and the Output Kernel Tree [6].

2.3.1. Local Models

This approach uses each vertex of the graph as seed and infers the label of the edges between this vertex and the other vertices of the graph. For each seed vertex, we solve a local pattern recognition task for the subgraph around the seed.

First, we select a vertex from V as seed vertex v_{seed}. Then, we extract a subset from S that includes the v_{seed} and the other vertexes of the graph connected with v_{seed}. This model labels the vertices instead of the edges, so the resulting training set is $S_{seed} = \left((v'_1, y'_1), \ldots, (v'_n, y'_n)\right)$ where $v'_i \in V$ is a vertex connected with v_{seed} and y'_i is the label of edge between v'_i and v_{seed} [5].

Then, we use the set S_{seed} to train a machine learning algorithm, for example Support Vector Machine, in order to infer the labels of every vertex v' that are not in S_{seed}. The label of v' is assigned to the edge between v' and v_{seed}. Each of the previous steps is repeated for each vertex of the set V, choosing it as v_{seed}. Finally, we combine the predictions of the iterations over an edge's label to obtain the final label of the edge.

The pattern recognition algorithm used in this approach exploits the idea that if a vertex v', which is known to be connected with the v_{seed}, is similar to a vertex v'', then, it is likely that v'' is also connected to v_{seed} [5].

2.3.2. Global Models

The previously introduced local models do not take advantage of the whole training set to infer new edges due to the fact that in each iteration only the labeled edges around the seed vertex are used. Consequently, global models have been developed in order to train a classifier using the whole training set.

These models are based on the idea of inferring unknown edges between two vertices using data of similar pairs of vertices with known edges. Then, we try to find two vertices v'' and v''', where v'' is similar to v and v''' is similar to v'. Consequently, it is reasonable to think that the label of the edge between v'' and v''' will be the same of the edge (v, v') [5]. This inference cannot be done on local models.

Vert describes in [5] the use of the Kernel Trick to compute the similarity of pairs of vertices. First, we use the direct product $\psi_{\otimes}(u, v)$ to represent the pair of vertices v and v' in a feature space where a binary classification of the pairs can be done.

$$\psi_{\otimes}(v, v') = \phi(v) \otimes \phi(v')$$

Where $\phi(v)$ and $\phi(v')$ are the feature vectors of the vertices v and v'. This representation allows for applying the kernel trick to compute the similarity between two pairs of vertices. Let $K_{\otimes}$ be the kernel between two pairs of vertices (v, v') and (v'', v''') [5].

$$\begin{aligned} K_{\otimes}\big((v, v'), (v'', v''')\big) &= \psi_{\otimes}(v, v')^T \psi_{\otimes}(v'', v''') = \big(\phi(v) \otimes \phi(v')\big)^T \big(\phi(v'') \otimes \phi(v''')\big) \\ &= \phi(v)^T \phi(v'') \times \phi(v')^T \phi(v''') = K_V(v, v'') \times K_V(v', v''') \end{aligned}$$

Where $K_V: V \times V \to \mathbb{R}$ is a kernel that computes the similarity between two vertices. Basically for the similarity between (v, v') and (v'', v''') we compare v to v'' and v' to v'''. The measure of similarity of the kernel $K_{\otimes}$ can be used in a machine learning method such as Support Vector Machines to do the classification of new edges.

2.3.3. Output Kernel Trees

As mentioned, several machine learning approaches have been developed for the inference of biological networks. The Output Kernel Trees (OK3) [5] is one of the supervised models for this task. This approach proposes the kernelization of regression trees to learn a kernel that allows making predictions about the edge between two vertices in a graph. Moreover, this model uses the original input space that gives more interpretability contrary to other methods with black-box models [6].

Based on the formulation of a graph that I have introduced at the beginning of this section, this method defines a positive definite symmetric kernel $k: V \times V \to \mathbb{R}$ that encodes the proximity of two vertices in the graph. The kernel k gives a higher value to pairs of connected vertices. As explained in Section 2.2 this kernel induces a feature map ϕ into a Hilbert Space $\mathcal{H}$.

$$k(v, v') = \langle \phi(v), \phi(v') \rangle$$

The aim is to find an approximation of k denoted as $\hat{k}$ described by their input features [5]. The OK3 method tries to find an approximation $\hat{\phi}(v)$ of the output feature vector $\phi(v)$ by growing a binary classification tree on the input vectors of the training set [5].

The construction of the tree using binary tests over the input features of the vertices is based on the minimization of the square distance in $\mathcal{H}$ between the training samples in the different nodes and leafs. Each leaf L of the tree is labeled with the average of the output feature vectors $\hat{\phi}_L$ of the different learning samples of the leaf. An approximation $\hat{\phi}_L(v)$ of the output feature vector of a new vertex v is given by searching in the tree the proper leaf.

Given two vertices v and v', we have found that they lie on the leafs L_1 and L_2 respectively. Then, we can approximate its kernel value $\hat{k}(v, v')$ averaging the sum of the kernel values between the learning samples of L_1 and L_2 given by the kernel k. OK3 predicts the binary label of the edge between a pair of vertices (v, v') thresholding $\hat{k}(v, v')$ [6]. If we obtain a value over the threshold, we will predict that there exists interaction between the proteins represented by the vertices v and v'.

2.4. Regression methods

The Input-Output Kernel Regression is based on the application and extension of Kernel Ridge Regression. Regression models are widely used in machine learning. Before describing the IOKR model, a brief introduction to regression methods is given.

2.4.1. Linear Regression models

The linear regression consists of finding a function $E(Y|X)$ that shows how the features of the input data (X) condition the output (Y). Let X be the vector that represents the features of an input object $X = (X_1, X_2, \ldots, X_n)$, where X_i represents the ith-feature. Moreover, let w be a vector of regression parameters. The linear function between the input and output is:

$$f(X) = w_0 + \sum_{i=1}^{l} w_i \, x_i$$

In order to learn the values of w I have a set of training data $S = ((x_1, y_1), (x_2, y_2), \dots, (x_l, y_l))$ where x_i is the feature vector of the ith instance and y_i is the target or output value. We use the least square method to choose the w that minimizes the residual sum of squares (RSS).

$$RSS(w) = \sum_{i=1}^{l} (y_i - f(x_i))^2 = \sum_{i=1}^{l} \left(y_i - \left(w_0 + \sum_{i=1}^{l} w_i \, x_i \right) \right)^2$$

As described in [6] we can minimize the previous cost function by taking derivatives. First, we rewrite the residual sum-of-squares as follows.

$$RSS(w) = (y - Xw)^T (y - Xw)$$

Where X is a matrix with size $l \times (p + 1, w)$ with each row is an input vector from the training set with an additional first column with 1, l is the number of samples in the training set and p is size of the input vectors. Then, we derive by w obtaining the unique solution of the minimization problem [6].

$$\hat{w} = (X^T X)^{-1} X y$$

The unique solution is an approximation of the vector of regression parameters. This approximation is used in the initial function f to predict the output of a new input vector.

2.4.2. Ridge Regression

The Ridge Regression, also known as Tikhonov regularization, is a type of linear regression model where a regularization constant is introduced to achieve "weight decay" [7]. The purpose of the regularization term is to penalize the norm of the parameter vector w to avoid overfitting. Consequently, the cost function can be written as follows.

$$C(w) = \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \, \|w\|^2 = \sum_{i=1}^{n} \left(y_i - \left(w_0 + \sum_{i=1}^{l} w_i \, x_i \right) \right)^2 + \lambda \, \|w\|^2$$

Where w is the vector of regression parameters, x_i is an input vector of the training set, y_i is the output of such input vector and λ is the regularization term. As described in [7] the optimum vector w can be computed by taking the derivatives from the previous cost function.

$$w = \left(\lambda I + \sum_i x_i x_i^T\right)^{-1} \left(\sum_i y_i x_i\right)$$

Where I is the identity matrix. The value of the regularization parameter λ has to be determined experimentally. For this purpose, we can use for example cross validation methods.

2.4.3. Kernel Ridge Regression

The Ridge Regression algorithm can be combined with kernels to carry out the task of learning a non-linear function between input and output. The input feature vectors are not anymore defined by a value x_i, instead a transformed feature vector $\phi(x_i)$ is used. As a consequence, we can rewrite the derivation of the parameter vector w as follows [7].

$$w = \phi(\phi^T\phi + \lambda I_l)^{-1}y$$

Where ϕ is a matrix where each row i contains the feature vector of the instance x_i, I_l is the identity matrix of size l and y is a vector with the output value of each instance. In order to predict the value of a new instance x I project its feature vector onto the hyperplane defined by w. The linear regression model that retrieves the prediction $\hat{y}$ of a new data-case can be defined as follows.

$$\hat{y} = f(x) = w^T\phi(x)$$

This formulation let us introduce a kernel κ and its Gram matrix K to encode the similarity between the different instances. Then, $\hat{y}$ can be written as follows.

$$\hat{y} = y(\phi^T\phi + \lambda I_l)^{-1}\phi^T\phi(x) = y\,(K + \lambda I_l)^{-1}\kappa(x)$$

Where the values of the Gram matrix are defined as $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$, where x_i and x_j are two different instances. This Gram matrix defines the kernel $\kappa(x) = K(x_i, x)$.

3. Methods

The approach used for the network inference in this work is based on the framework called Input-Output Kernel Regression introduced in [9, 10]. This method extends the regression models explained in Section 2.4. Moreover, the work in [9, 10] extends the IOKR to a semi-supervised setting. Both settings have been reviewed and implemented to be analyzed in this work.

IOKR contrary to OK3, described in the Section 2.3.3, uses the kernelized input space to learn an output kernel. Using this output kernel we can encode the proximity of the proteins to each other in the PPI network. As in OK3 does, the proximity value of a pair of proteins is thresholded to infer whether an interaction exists or not. Although I focus in this work on the inference of PPI networks, IOKR can be applied on the link prediction of other graph-based structures such as social networks [9].

The data for PPI network inference usually comes from different sources. Consequently, I need to implement methods to combine the input data sources in order to learn the input kernel. The selected methods are a simple average sum and a Multiple Kernel Learning using the Kernel Centered Alignment to compute a weighted combination of the data sources.

First of all, the following information is available for the network inference:

- A set of proteins $\mathcal{O}$, where a protein $o \in \mathcal{O}$ is represented by a feature vector $x(o)$ that describes different properties of the protein, where $n = dim(\mathcal{O})$. This set defines the full graph G.

- We define a subset $\mathcal{O}_l \subset \mathcal{O}$ that represents the proteins of the training set, where $l < n, l = \dim(\mathcal{O}_l)$. This set contains a total of l proteins from a random split of $\mathcal{O}$ and defines a subgraph G_l.

- We are given an adjacency matrix W of size $n \times n$. Let o_i and o_j be two proteins of the training set, $W(o_i, o_j)$ is the label of the edge between the vertices that represent o_i and o_j on the graph G.

Once I have defined the available data, I review the applied methods based on the Input-Output Kernel Regression and the Centered Alignment used for Multiple Kernel Learning of the input kernel.

3.1. Input-Output Kernel Regression

Previously, supervised approaches revised in the Section 2.3 are based on the classification of the edges of the graph using a binary pairwise classifier that has two vertices as an input. The Input-Output Kernel Regression transforms this classification problem into the learning of an approximation of the output kernel.

The IOKR is based on the kernelization of the output side defining an output feature space F_y. In this output feature space we can encode the proximity of the vertices in the graph using an output kernel κ_y in order to predict the label of the edges thresholding this proximity value. The method consists of the use of the Kernel Trick in the output feature space, similarly to OK3, to learn the output kernel to encode the proximity of the vertices in such feature space.

Let $\kappa_y: \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}$ be a PDS kernel that gives the proximity of two nodes in a PPI network. There exists a Hilbert Space $\mathcal{F}_y$ which corresponds to the output feature space. The proteins of the network are mapped in $\mathcal{F}_y$ using the function $: \mathcal{O} \rightarrow \mathcal{F}_y$. The proximity of two proteins of the PPI network encoded by the output kernel can be defined as the dot product of their images in the output feature space.

$$\forall\, o, o' \in \mathcal{O}, \kappa_y(o, o') = \langle y(o), y(o') \rangle_{\mathcal{F}_y}$$

The output kernel κ_y is unknown, thus, I need to learn an approximation $\hat{\kappa}_y$ based on the input data.

$$\forall\, o, o' \in \mathcal{O}, \hat{\kappa}_y(o, o') = \langle h(o), h(o') \rangle_{\mathcal{F}_y}$$

In that way, the aim is learning a mapping function $h: \mathcal{O} \rightarrow F_y$ which predicts the output feature vector $y(o)$ of a protein o in the feature space F_y (Figure 3) where we can measure its proximity. This is similar to the OK3 where the prediction is given by the label of a leaf of the tree. Then, the IOKR proposes a classifier function $f_\theta: \mathcal{O} \times \mathcal{O} \rightarrow \{0,1\}$ that thresholds the proximity value given by the output kernel $\hat{\kappa}_y$ to infer whether two proteins o and o' of the PPI network interact [9].

$$f_\theta(o, o') = sgn(\hat{\kappa}_y(o, o') - \theta) = sgn(\langle h(o), h(o') \rangle_{\mathcal{F}_y} - \theta)$$

As I have mentioned, we do not know the output kernel κ_y, however, we know the output Gram matrix K_{Y_l} that gives information about the proximity of two proteins of the training set S in $\mathcal{F}_y$. Let K_{Y_l} be a positive semi-definite matrix of dimension $l \times l$ where $\forall\, i, j \leq l, o_i, o_j \in \mathcal{O}, K_{Y_l}(i,j) = \kappa_y(o_i, o_j)$.

We need to compute a kernel that encodes the proximity between the vertices in the graph. The diffusion kernel is suitable to encode the proximity [6, 9, 10]. Then, the Gram matrix is defined as $K_{Y_l} = \exp(-\beta_1 L)$, where $L = D - W_l$ is the Laplacian matrix, D is the degree matrix for the vertices of the training set and W_l is the adjacency matrix of the graph G_l. The parameter β_1 controls the diffusion over the graph and its value will be set by cross validation.

IOKR kernelizes the input space X to encode the similarity between proteins of a PPI network. Consequently, the input data is defined by a Gram matrix K_X that encodes the similarity of each possible pair of proteins of $\mathcal{O}$. K_X is defined by a PDS kernel $\kappa_x: \mathcal{O} \times \mathcal{O} \to \mathbb{R}$, so each component of the matrix is given by $\forall\, i, j \leq n, K_X(i,j) = \kappa_x(o_i, o_j)$. Contrary to the output kernel, the input kernel κ_x is known. The computation of the matrix K_X is discussed in the Section 3.2.

As I described above, this model extends the Kernel Ridge Regression. In this case I am looking for the function h that computes an approximation of the output feature vector of a protein in $\mathcal{F}_y$.To develop the IOKR, we assume that there exists a general matrix A with dimension that projects the feature vector $x(o)$ of a protein o into the feature space $\mathcal{F}_y$ as the parameter vector w does in Kernel Ridge Regression (KRR) described in Section 2.4.3.

$$o \in \mathcal{O}, h(o) = h_A(o) = A\, x(o)$$

The computation of the matrix A, as the computation of the vector w in Ridge Regression, corresponds to the solution of a minimization problem.

The IOKR implemented in this work and described in [9, 10] extends the Input-Output Kernel Regression to a transductive setting, where I attempt to complete an existing network using the data of the nodes of the whole network. This setting is referred as a semi-supervised approach for network inference and is detailed in the Section 3.1.2.

The next sections describe the two settings of the model: supervised and semi-supervised. The main differences between both are the input side and the cost function used to learn the function h.

3.1.1. Supervised setting

In the supervised approach of the IOKR we use the input data of the proteins of the training set O_l to infer the labels of the rest of the edges of the network. Therefore, as input kernel matrix, I use a submatrix K_{X_l} of the Gram matrix K_X with only those rows and columns that correspond to proteins of the training set.

As in Ridge Regression, for the weight vector w, an optimization problem has to be resolved to learn the matrix A. I need a function h_A that minimizes the distance between the output feature vector $y(o_i)$ of a protein o_i and the prediction of its feature vector $h_A(o_i)$ in the output feature space $\mathcal{F}_y$. Therefore, the optimization problem consists of the minimization of a square loss function with a regularization parameter [9].

$$\min \sum_{i=1}^{l} \|h_A(o_i) - y(o_i)\|_{\mathcal{F}_y}^2 + \lambda_1 \|A\|_{\mathcal{H}}^2$$

This cost function can be seen as an extension of the Ridge Regression, where λ_1 is the regularization term to avoid the overfitting of the model defined by the function h_A. The minimization of the previous cost functions leads to a closed form solution for computing the model parameters [9].

$$\hat{A} = Y_l(K_{X_l} + \lambda_1 I_l)^{-1} X_l^T$$

Where Y_l is a matrix of dimension $dim(\mathcal{F}_y) \times l$ whose ith column corresponds to the output feature vector $y(o_i)$ of the protein o_i of the training set, I_l is the identity matrix of dimension $l \times l$. Moreover, X_l is the matrix of dimension $dim(\mathcal{F}_x) \times l$ where the ith column corresponds to the input feature vector $x(o_i)$ of the protein o_i in the feature space $\mathcal{F}_x$. The value of the regularization term λ_1 will be set testing the performance of a range of values and selecting the best.

3.1.2. Semi-supervised setting

The semi-supervised model consists of using additionally the input information of the proteins of the test set to train the classifier. The task of inference of the PPI network using the semi-supervised approach can be seen as the completion of the missing values of the matrix K_{Y_l}[9].

The work [10] describes how the cost function of the supervised is extended to this new model introducing the unlabeled data. A smoothness constraint λ_2 is introduced on the regression model. This constraint penalizes protein pairs (o_i, o_j) with a high similarity in the input features and a high distance between them in the output feature space $\mathcal{F}_y$. Consequently, we can define the optimization problem to learn the matrix A as follows.

$$\min \sum_{i=1}^{l} \|h_A(o_i) - y(o_i)\|_{\mathcal{F}_y}^2 + \lambda_1 \|A\|_{\mathcal{H}}^2 + \lambda_2 \sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij} \left\|h_A(o_i) - h_A(o_j)\right\|_{\mathcal{F}_y}^2$$

Where M is a matrix that encodes the similarity of the proteins in the input space [10]. As in the supervised setting, the minimization of the previous cost function leads to a closed form solution for computing the model parameters [9].

$$\hat{A} = Y_l U (K_{X_n} U^T U + \lambda_1 I_n + 2\lambda_2 K_{X_n} L_{X_n})^{-1} X_n^T$$

Where $L_{X_n} = \exp(-\beta_2(D - W))$ is a diffusion kernel matrix of the whole graph, where D is the degree matrix of the vertices, W is the adjacency matrix of the graph and I_n is the identity matrix of dimension $n \times n$. Moreover, the matrix U is a matrix of dimension $l \times n$, where the left side is the identity matrix of size $l \times l$ and the right side is a zero matrix of size $l \times t$, where $t = n - l$ is the size of the test set. X_l is defined as a matrix of dimension $dim(\mathcal{F}_x) \times l$ where each column corresponds to the projection of the feature vector of the protein o_i in the feature space $\mathcal{F}_x$.

The values of the regularization term λ_1 and the smoothness constraint λ_2 will be set by cross validation. The same process will be done for selecting the value of the parameters β_1 and β_2 of the diffusion kernels.

3.2. Multiple Kernel Learning

Studies of PPI networks usually involve several data sources of a protein in order to infer its interaction. Different data sources are used because they should contain complementary information about PPIs that can be helpful to improve the performance of the PPI network inference.

The use of different data sources requires the use of methods to fuse them to compute the input Gram matrix K_X. I have implemented two methods. The first one is the average sum of the different data sources. The second one is a Multiple Kernel Learning that uses the correlation between the data sources and the output kernel matrix given by the kernel centered alignment to weight the data sources.

3.2.1. Average sum

This solution can be seen as a naive approach of combining kernels, by computing a uniform combination. Basically, I sum the different Gram matrices of the data sources and normalize the sum. The combined input kernel K_X is defined as follows.

$$K_X = \frac{1}{d}\sum_{k=1}^{d} K_k$$

Where K_k is the Gram matrix that results from the application of some kernel on the k-th data set and d is the number of data sets.

Due to its simplicity this algorithm takes into account each data source equally. However, previous study [16] has shown that if we analyze the power of prediction of each data source individually, we can find differences in their performance. Because of this, several approaches have been developed to implement a weighted combination of kernels.

3.2.2. Kernel Centered alignment

The goal of the following method is to compute weights for the kernels of different data sources during combination or Multiple Kernel Learning. This method, described and tested in [11], shows better performance than uniform combination.

First of all, let us introduce the notion of centered kernel matrices. Let K be a kernel matrix defined by a PSD kernel function $\kappa\colon \mathcal{O} \times \mathcal{O} \to \mathbb{R}$. Centering a kernel matrix consists of centering the feature map $y\colon \mathcal{O} \to \mathcal{F}$ associated with κ removing its expectation. Consequently, each component of the centered matrix K_c can be computed from K as follows.

$$\forall\, i,j \leq n, [K_c]_{ij} = K_{ij} - \frac{1}{n}\sum_{i=1}^{n} K_{ij} - \frac{1}{n}\sum_{j=1}^{n} K_{ij} + \frac{1}{n^2}\sum_{i=1}^{n} K_{ij}$$

The alignment is computed between one data source and the target kernel, this correlation value is used as a weight during combination of the data sources. The centered alignment provides us correlation measure between two kernels. As described in [11] we can obtain the correlation between two kernel matrices K and K' as follows.

$$\hat{\rho}(K, K') = \frac{\langle K_c, K_c' \rangle_F}{\|K_c\|_F \|K_c'\|_F}$$

Where $\langle K_c, K'_c \rangle_F$ and $\|K.\|_F$ denote the Frobenius product and the Frobenius norm respectively [11].

$$\langle K_c, K'_c \rangle_F = Tr[K_c^T K'_c] = \sum_{i,j} K_{c_{ij}} K'_{c_{ij}}$$

$$\|K_c\|_F = \sqrt{\langle K_c, K_c \rangle_F}$$

The method consists of computing the centered alignment between a base kernel matrix and the target kernel matrix individually [11]. The computed correlation is used as a weight for the base kernel K_k. Then, the input Gram matrix K_X can be computed as a weighted sum of the base kernels.

$$K_X = \sum_{k=1}^{d} \hat{\rho}(K_k, K_Y) K_k$$

Where K_Y is the target kernel matrix, a Laplacian matrix $K_Y = D - W_l$, where D is the degree matrix for the vertices of the training set and W_l is the adjacency matrix of the graph G_l. K_k is the Gram matrix that results from the application of some kernel on the k-th data source and d is the number of data sources. I assume that a data source whose matrix kernel is more correlated to K_Y will perform the PPI network inference better, then, a higher weight is given to this data source.

4. Experimental setup and results

The programming language selected for this work is *MATLAB*. This was based on the fact that the model has a great mathematical complexity and this language offers several advantages. It provides a huge range of already implemented mathematical functions and an automatic parallelization of operations, for example loops. The code of the implementation of the IOKR can be consulted in the Section Appendix I of this work.

4.1. Performance analysis methods

In this work I use two methods to analyze the results of the experiments: the Receiver Operating Characteristic (ROC) and the Area Under the Curve (AUC). The first one gives visualization of the classifier's performance and the second one is a performance measure [12].

The "raw data" from the execution of a binary classifier are the counts of how many instances of the problem have been classified correctly and wrongly. In the design of the binary classifier for the inference of PPI network I try to find a classifier that increases the number of existing interactions classified as existing interactions {1} (True positives (TP)) and decreases the number of non-interactions classified as existing interaction {1} (False positives (FP)).

The Receiver Operating Characteristic (ROC) is a way of visualizing a classifier's performance represented as a curve in a two-dimensional graph [12]. It consists of plotting the True positive rate against the False positive rate varying the decision threshold of the classifier [12]. A classifier with a ROC curve closer to the upper-left corner is better.

This curve is also used to select the optimal threshold of the classifier. The decision threshold or operating point of the classifier will be the proximity value of the instance represented by the closest point of the curve to the upper-left corner [12].

The Area Under the Curve is a performance measure that consists of calculate the area under the ROC curve. A higher AUC value indicates a better performance of the classifier.

4.2. Experimental setup

In order to get truthful measures of the performance of the different methods and setups we run each experiment ten times. Each time, I randomly sub-sample a training set of proteins that represents a specific percentage of the total amount of proteins of the PPI network and consider the other proteins as testing set. After the runs I average the different ROC and AUC results over the different runs. This repeated evaluation of performance on random subsets is called cross validation and guarantees unbiased performance measurements.

Secondly, as [6] does, I consider two sets of interactions in the inference of a PPI network:

- The interactions between proteins of the training set and proteins of the testing set (TR/TS). This means that one of the interaction partners has been seen during training.
- The interactions between proteins of the testing set (TS/TS). This means that none of the interaction partners has been seen during training, thus the inference of this group is more difficult than the previous.

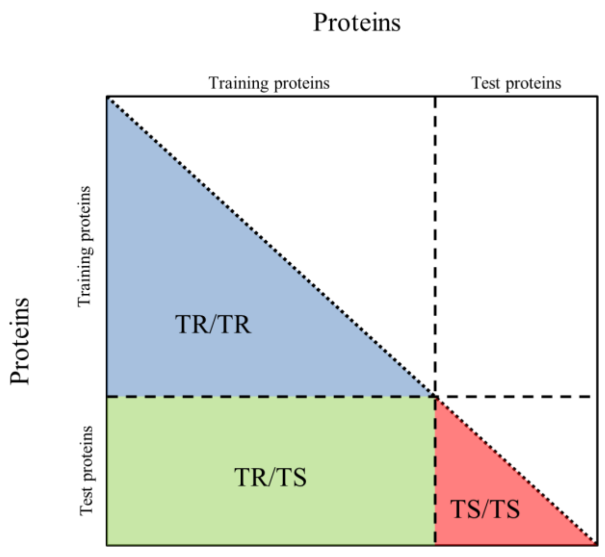

Figure 3: Representation of the interactions between proteins in a symmetric binary matrix. The interactions are split in three sets, which are represented by different colors.

Figure 3 shows how the interactions are represented in a symmetric binary matrix of size $n \times n$, where the columns and rows are the proteins of the PPI network and the cells are the label of the edge between two vertices of the network. It includes the interactions between proteins of the training set (TR/TR), this means that both of the interaction partners have been seen during training. This set is not considered in the performance analysis because it is expected that a classifier will obtain an AUC close to 1. Therefore, its analysis was used only during the development to detect errors in the implementation.

I analyze the performance of the classifier on each set separately. The first set of interactions usually gets a better performance due to the classifier has been trained using the input and output data of one of the interaction partners versus none for the second case.

4.3. Experimental data

The protein-protein interaction network considered in this work for analyzing the performance of the implementation of the IOKR is the PPI network of the Protein Secretory Machinery of the yeast *Saccharomyces cerevisiae* [1]. This PPI network is formed by 161 proteins directly involved in several functions of the Secretory Machinery.

A total of 14 data sources have been used to represent the features of the proteins of the network:

- Microarray expression data contains scores that represents the level of co-expression of proteins obtained by microarray experiments.
- Cell localization data. Each protein has a binary vector where 13 cell localizations are considered, for instance the cytoplasm. The ith value of the vector is set to 1 if the protein has been found in the ith localization.
- BLAST sequence alignment score of the protein sequence with sequences of the UniProt database. BLAST is a sequence similarity search program that provides statistical information about an alignment [13].
- Global Trace Graph (GTG) is an improved sequence alignment score of the protein sequence with genetic sequences of interest. GTG is a cluster algorithm to perform sequence alignments [14].
- InterProScan is a tool that unifies several protein signature databases and provides functional analysis of a given protein sequence [15]. The available data comes from the following protein signatures databases: FingerPRINTScans, CATH-Gene3D, HAMAP, PANTHER, patternscans, Pfam, PIRSF, ProDom, pfscan, SMART, SUPERFAMILY and TIGRFAMs.

The work [16] found by testing different kernels that Microarray expression data achieves the best performance using the RBF kernel. On the other hand, the other data sources have obtained the best performance with linear kernel. I have used this kernel selection in the following experiments.

4.4. Parameter tuning

In this section I show the results of the experiments for setting the values of the different parameters of the model. First, I start with the parameters β_1 and β_2 of the diffusion kernels. Then, I present the result of the experiments to set the values of the regularization term λ_1 and the smoothness constraint λ_2.

I have tested different values of β_1 for the supervised setting and different pairs of (β_1, β_2) for the semi supervised setting. The range of values for both parameters was 1, 2 and 3. In both settings I used 80% of the proteins as training set.

After analyzing the AUC scores of the different experiments I did not find noticeable performance differences in any of the settings between the different values of β_1 and β_2. Consequently, I set the value of β_1 and β_2 to 1 in both settings.

Supervised setting

In the case of the supervised setting the regularization term λ_1 in the cost function has to be set to a value that gives the best performance. I have tested different values and selected the one which gives the best AUC.

Table 1 shows the AUC scores obtained testing a range of values for λ_1 from 0.1 to 1. I have analyzed the performance on the two sets of interactions as explained above. We can see that the performance of the classifier is increased when we choose a higher value. Nevertheless, when we choose a value higher than 0.6 the penalization applied on the cost function produces a worse performance.

	λ_1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
AUC	tr/ts	0.856	0.865	0.87	0.872	0.873	**0.874**	0.873	0.872	0.871	0.87
	ts/ts	0.78	0.786	0.789	0.79	0.79	**0.791**	0.79	0.789	0.788	0.787

Table 1: AUC of the Supervised setting varying the regularization parameter λ_1, using 80% of the data for training and $\beta_1 = 1.0$. *tr/ts* is the inference of interactions between proteins of the training set and proteins of the test set, and *ts/ts* is the inference of interactions between proteins of the test set.

This situation can be seen easily in the Figure 4, where a greater range of values of λ_1 is shown. We can see that there is an increment of the AUC in both sets of interactions from 0.05 to 0.5. Then, if we increase the value there is a stabilization of the AUC from 0.5 to 0.6, followed by a decrement from 0.65 to 1. Consequently, in the following experiments with the supervised setting I will set the value of λ_1 to 0.6.

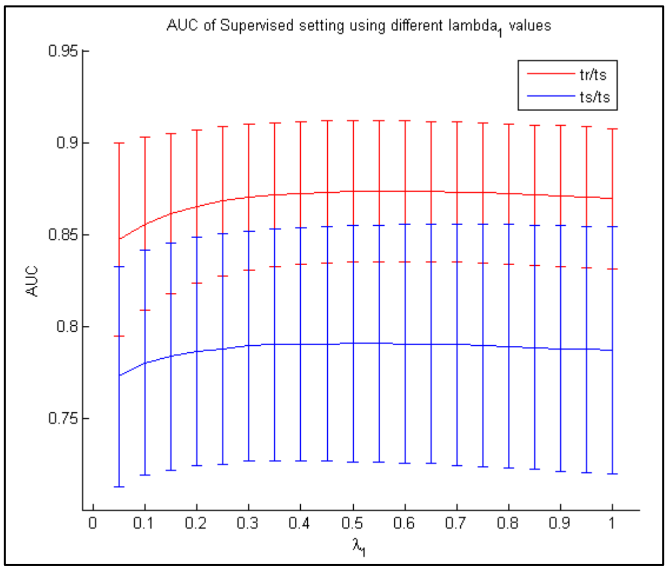

Figure 4: AUC error bars of the supervised setting using different values of λ_1 for the interactions between the training set and testing set and between the testing set. 80% of the proteins are used as training set.

Semi-supervised setting

In the case of the semi-supervised setting the regularization term λ_1 and the smoothness constraint λ_2 of the cost function have to be set to a value that gives the best performance. I have tested different pairs of values and selected the one that gives the best AUC.

First, I have run the semi-supervised setting varying the $\lambda_1$and λ_2 from 0.1 to 1.0. Table 2 shows a subset of the experiments that are representative to show the behavior of the AUC when we vary $\lambda_1$and λ_2.

It can be seen in Table 2 that there is an increment of the AUC with λ_1 from 0.2 to 0.8. Then, the performance of the classifier decreases when we use a value of $\lambda_1$higher than 0.8. On the other hand, we can see that there is a slight difference on the performance when we vary the value λ_2. However, we can appreciate a slight improvement when we use low λ_2 values.

	λ_2				
λ_1	0.2	0.4	0.6	0.8	1.0
0.2	0.841	0.8408	0.8407	0.8407	0.8406
0.4	0.865	0.865	0. 865	0.8649	0. 8649
0.6	0.8716	0.8716	0.8714	0.8714	0.8713
0.8	**0.8731**	**0.873**	**0.8729**	**0. 8729**	**0.8728**
1.0	0.8727	0.8726	0.8725	0.8725	0.8725

Table 2: AUC of the Supervised setting for the different value pairs of λ_1 and λ_2. The interactions are between proteins of the training set and proteins of the test set. 80% of the proteins are for training, $\beta_1 = 1.0$ and $\beta_2 = 1.0$.

The results of Table 2 lead me to carry out another experiment using a fixed value of $\lambda_1$and varying λ_2 using a bigger range. I select the λ_1 value with the best AUC, in this case is 0.8. Given the behavior of the AUC when we decrease the λ_2, I try lower values of λ_2 in order to see if there is an improvement of the performance. The Table 3 shows the results of the experiment.

	$\boldsymbol{\lambda_2}$	0.0001	0.001	0.01	0.05	0.15	0.25	0.5	0.75	1.0
AUC	tr/ts	0.879	**0.882**	0.881	0.875	0.874	0.873	0.873	0.873	0.873
	ts/ts	0.7427	**0.76**	0.7481	0.7301	0.7263	0.7255	0.7249	0.725	0.725

Table 3: AUC of the Supervised setting for the different values of λ_2 with a fixed $\lambda_1 = 0.8$. The interactions are between proteins of the training set and proteins of the test set. 80% of the data is for training, $\beta_1 = 1.0$ and $\beta_2 = 1.0$.

In the Table 3 we can see that decreasing the value of λ_2 improves the performance of the classifier. Figure 5 shows clearly the mentioned improvement for both sets of interactions.

The value 0.001 for λ_2 achieves the best performance in the classification of the two types of interactions sets. A slight decrement on the value of λ_2 would harm the performance of the classifier in both set of interactions.

Given the results of this last experiment I can state that the best performance is achieved when we set the values of λ_1 and λ_2 with 0.8 and 0.001 respectively. Therefore, I will set the values of λ_1 and λ_2 to 0.8 and 0.001 respectively for the following experiments.

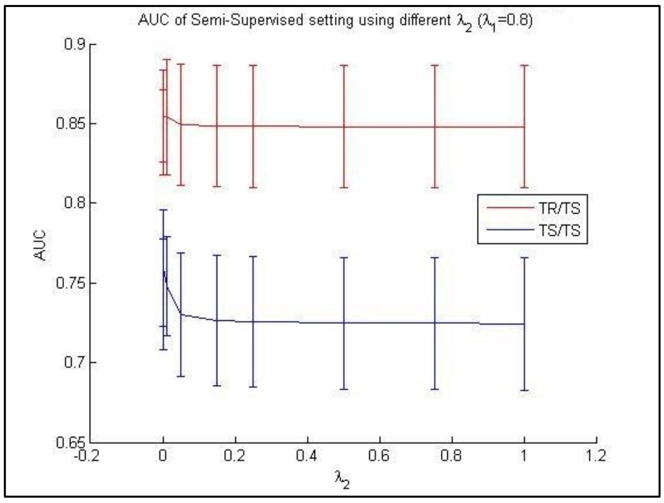

Figure 5: AUC of the Supervised setting for the different values of λ_2 with a fixed $\lambda_1 = 0.8$. 80% of the proteins are used as training set.

4.5. Performance analysis

After I have analyzed the values of the parameters of the model that achieve the best performance, I describe the results of the different experiments that have been carried out to analyze the performance of the IOKR using different settings.

4.5.1. Supervised vs. Semi-Supervised

In this section I present the results of several experiments to compare the two settings of the Input-Output Kernel Regression. Both settings use the average sum to combine the data sources to generate the input kernel. I have run the IOKR for the different settings using different percentages of data in the training phase.

First, I will show the results of the inference of TR/TS interactions. In Figure 7 we can see that both settings improve their performance when we increase the size of the training set. This behavior can be considered as normal because the classifier build during the training phase has more information about the PPI network, and therefore its prediction should be more precise.

A relatively large improvement of the performance is appreciated in both settings when we increase the training data percentage from 10% to 50%. However, the performance improvements flatten down when more than 50% of the data is used in the training phase. Moreover, we can see that in both methods the use of more than 80% of the proteins as training set has no significant effects on the performance.

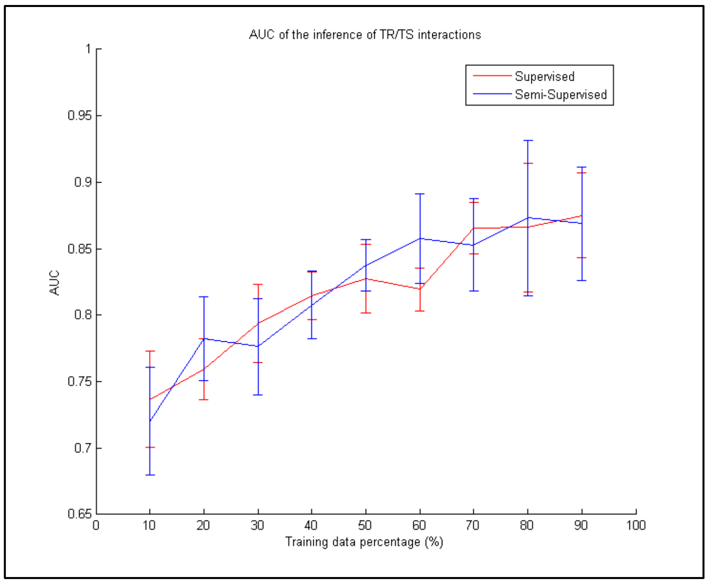

Figure 6: AUC error bars of the inference of TR/TS interactions using the supervised and semi-supervised setting and varying the percentage of proteins used in the training phase.

There is no a dominant setting in this case. The supervised setting seems to have a better performance when we use less data in the training phase, from 10% to 40%. On the other hand, the semi-supervised setting achieves a better performance in higher percentage, especially from 50% to 80%, obtaining the highest performance with 80% of data as training set.

Figure 7 represents the comparison of the AUC scores of both settings on the inference of TS/TS interactions of the PPI network. The plot represents the variation of the performance when the amount of proteins used in the training phase is increased.

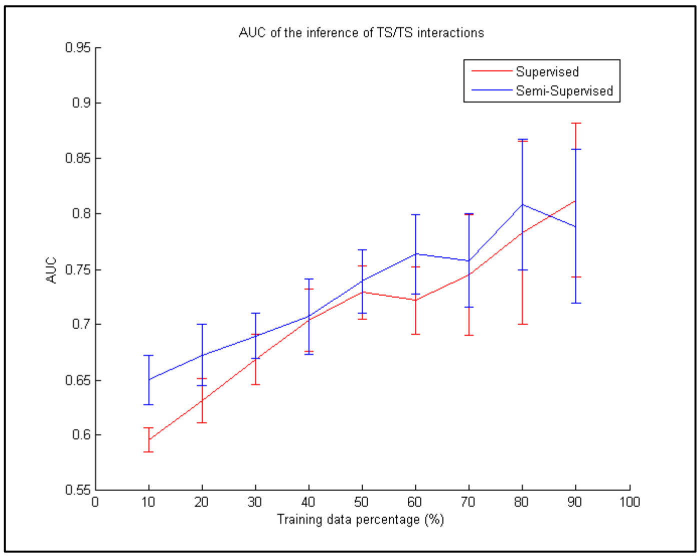

Figure 7: AUC error bars of the inference of TS/TS interactions using the supervised and semi-supervised setting and varying the percentage of proteins used in the training phase.

As noticed for the TR/TS interactions the performance of the classifier increases when the size of the training set is bigger. However, we can see that the semi-supervised setting experiments a decrement of the performance when 90% of the proteins are used as training set. However, this anomalous behavior can be caused by the high variation of the AUC of this last percentage.

In this case the supervised setting outperforms the semi-supervised setting using from 10% to 80% of the data in the training phase. Using 90% of the data we can see that the average AUC is higher for the supervised setting. Nevertheless, there is a high overlapping of the error bars of both settings, which indicates a high variation of the AUC for this percentage.

4.5.2. Individual data sources

As I explained in Section 3.2.2 the Multiple Kernel Learning gives a weight to each data source in order to build the input kernel using a weighted sum of the data sources. Consequently, I want to analyze if the given weight of the data source is correlated with the performance of a classifier trained only with this data source.

First, I show the weights of the different sources given by the MKL. Later, I will show the correlation between the weights of the data sources and the performance (AUC) of the classifier trained with individual data sources.

Table 4a and Table 4b show the weights of the data sources computed using the Kernel Centered alignment. The data sources BLAST and GTG, which are sequence alignment scores, show the highest weights. One could think that these two data sources will achieve the best classification results when used individually to train a classifier. In the next paragraphs I analyze whether this idea is correct.

ID	1	2	3	4	5	6	7
Data source	Expression	Localization	BLAST	FPrintScan	Gene3D	GTG	HMMPanther
Weight	0.3638	0.3528	**0.7191**	0.2975	0.5440	**0.7079**	0.5632

Table 4a: Average of the weight of each data source given by the MKL for the weighted sum of the data sources. The first row corresponds to the numeric identifier of the data source. Using 80% of data in the training phase.

ID	8	9	10	11	12	13	14
Data source	PatternScan	HMMPfam	HMMPIR	ProfileScan	Protein clusters	HMMSmart	Superfamily
Weight	0.3699	0.5837	0.2746	0.3814	0.4069	0.419	0.5319

Table 4b: Average of the weight of each data source given by the MKL for the weighted sum of the data sources. The first row corresponds to the numeric identifier of the data source. Using 80% of data in the training phase.

I have run an experiment where I trained a classifier using each data source isolated. Both settings of the IOKR were taken into account in this experiment. Then, I performed the classification of the TR/TS interactions and TS/TS interactions using these classifiers. Figure 8 and Figure 9 show the AUC vs. weight points of the different data sources and the regression line of such points of the classifiers trained with single data source using the supervised and semi-supervised setting.

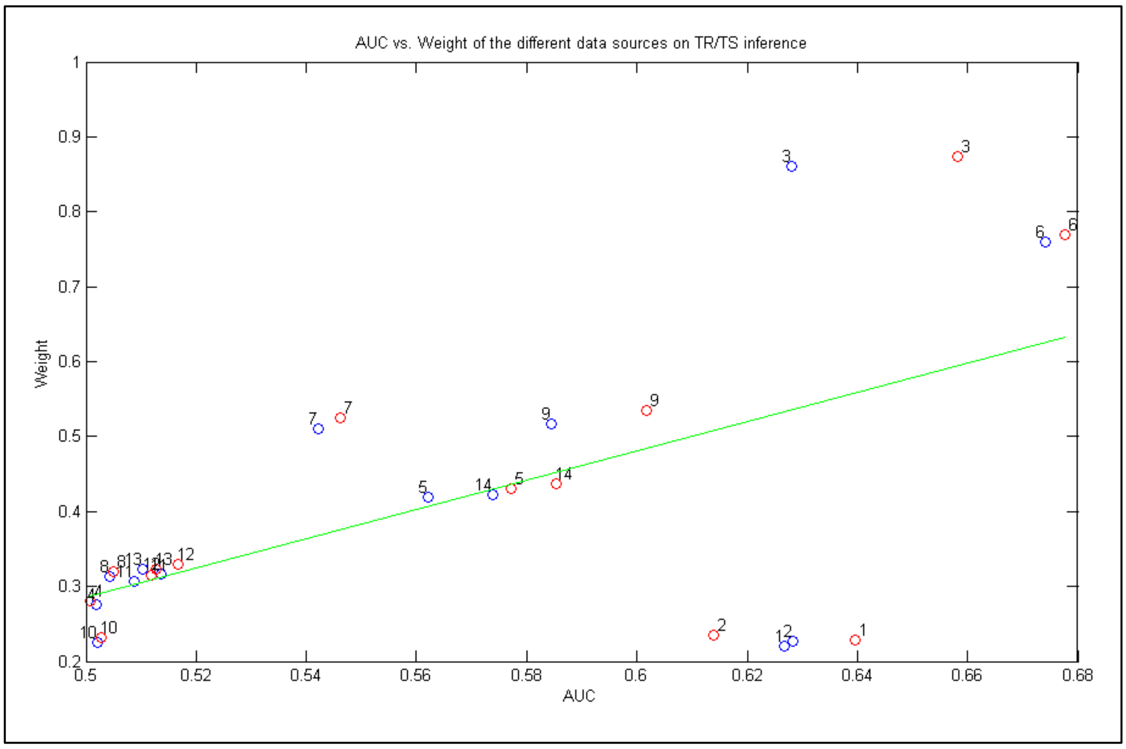

Figure 8: AUC vs. Weight linear regression of inference of TR/TS interactions using individual data sources on the supervised and semi-supervised settings. 80% of the proteins are used in the training phase. Red points correspond to the AUC of the supervised setting and blue points correspond to the AUC of the semi-supervised method. The labels of the points correspond to the data source identifiers of tables 4a and 4b.

First, I will analyze the performance of the data sources on the set of TR/TS interactions. In Figure 8 we can see that the BLAST and GTG, which obtained the highest weights, have the best performances. GTG outperforms all the other data sources on both settings while BLAST is in the same range of performance on the semi-supervised setting.

If we analyze the performance of the data sources with the lowest values we can see for example that FPrintScan with a weight of 0.2975 has an AUC around 0.5, which means a random performance. On the other hand, the HMMPanther with a high weight shows small AUCs, but they are better than the data sources with lower weights, which show AUCs around 0.5.

The expression data and localization data, at the bottom on the left of the figure, achieve a classifier with a great performance, reaching the BLAST data source in the supervised setting. However, the Multiple Kernel Learning gives them small weight to generate the combined kernel.

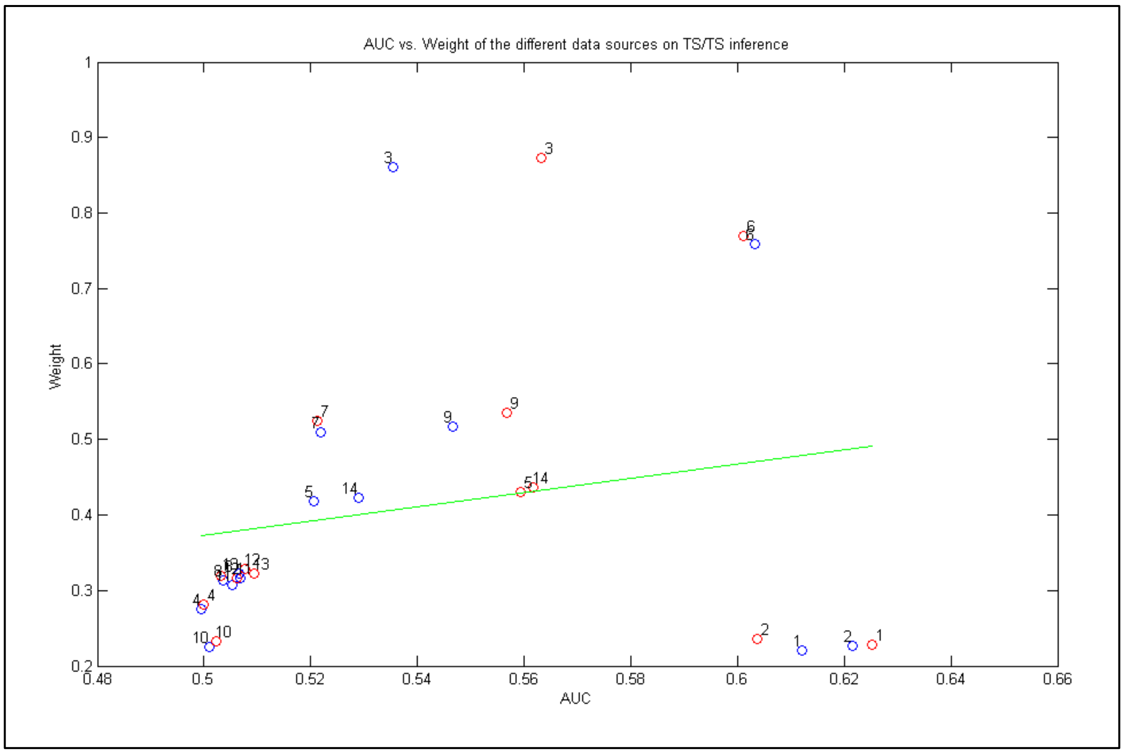

Figure 9: AUC vs. Weight linear regression of inference of TS/TS interactions using individual data sources on the supervised and semi-supervised settings. 80% of the proteins are used in the training phase. Red points correspond to the AUC of the supervised setting and blue points correspond to the AUC of the semi-supervised method. The labels of the points correspond to the data source identifiers of tables 4a and 4b.

Analyzing the linear regression in Figure 8 we can see that in the inference of TR/TS interaction the weight of the data sources is somewhat related with its performance as individual data source for training a classifier. The performance of a classifier in the inference of TR/TS interactions tends to be better when it its trained with a data source with higher weight.

If we analyze the performance on the set of TS/TS interactions shown in Figure 9 we find a contradiction with the hypothesis mentioned at the beginning of this section. BLAST shows a poor performance in both settings. However, GTG maintains some of the highest performance in the supervised and the semi-supervised setting. On the other hand, the expression data and localization data have the highest AUC in both settings. However, both data sources have received a low weight.

Analyzing the linear regression in Figure 9 we can see that in the inference of TS/TS interaction the weight of the data sources are not very related with its performance as individual data source for training a classifier.

4.5.3. Multiple Kernel Learning

In this section I describe the results of the experiments to analyze the performance of the implementation of the Multiple Kernel Learning to build the input kernel. I compare the performance of the MKL on the supervised and semi-supervised settings with the performance of the average sum of kernels, both described in Section 3.2.

The following figures show the evolution of the AUC scores and their variation for both settings when we vary the training data percentage using the MKL and the average on the input side. First, I will analyze the performance of the settings in the inference of interactions of the TR/TS set.

In Figure 10 we can see that for the supervised setting the use of the average outperforms the MKL. Only when we use 60% of the data in the training phase both settings achieve the same performance.

The increment of performance is similar in both cases, specially from 10% to 70%. Using more than 80% of the data as training set has slight improvements in the performance.

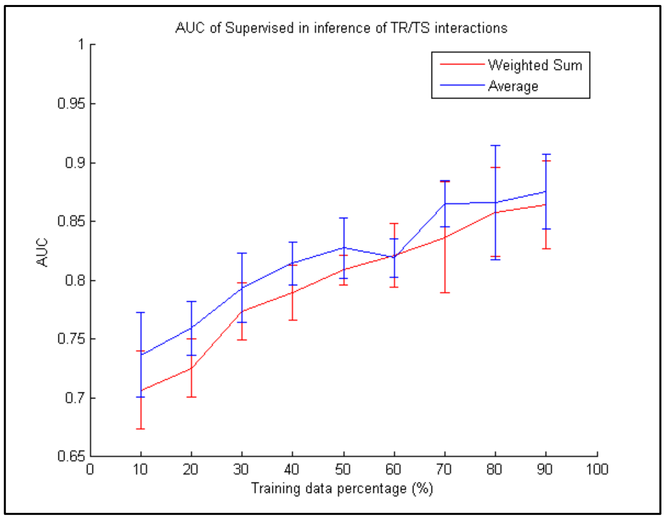

Figure 10: AUC of the inference of TR/TS interactions using the supervised with various setups. The setups consist of using the Weighted Sum to build the input kernel or the Average Sum of the data sources. 80% of the proteins are used as training set.

If we analyze the Figure 11 we can see that in general the semi-supervised setting works better when we use the MKL in the input side. The difference is slight when we use small percentages of data in the training phase. However, the differences are bigger when we use from 40% to 80% of the data as training set.

The case of 90% is confusing because of both the error bars of both setups are overlapped. So, the selection of the better setup in this case is a bit difficult. We should take into account the whole graph.

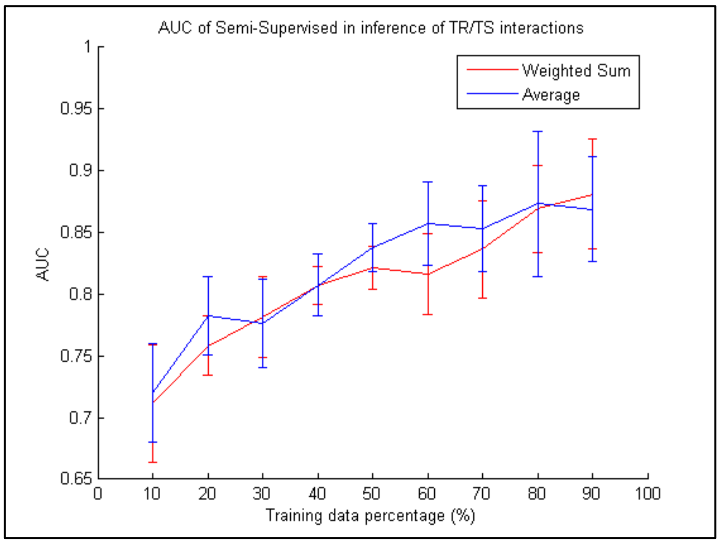

Figure 11: AUC of the inference of TS/TS interactions using the supervised with various setups. The setups consist of using the Weighted Sum to build the input kernel or the Average Sum of the data sources. 80% of the proteins are used as training set.

In the next paragraphs I show the results of the different setups for the inference of TS/TS interactions. First, in Figure 12 we can see clearly that the average combination outperforms the MKL when we use the supervised setting.

The differences in the performance are bigger when we use larger percentages of data in the training phase, from 60% to 90%. Using smaller training percentages, from 10% to 30%, the differences are slight.

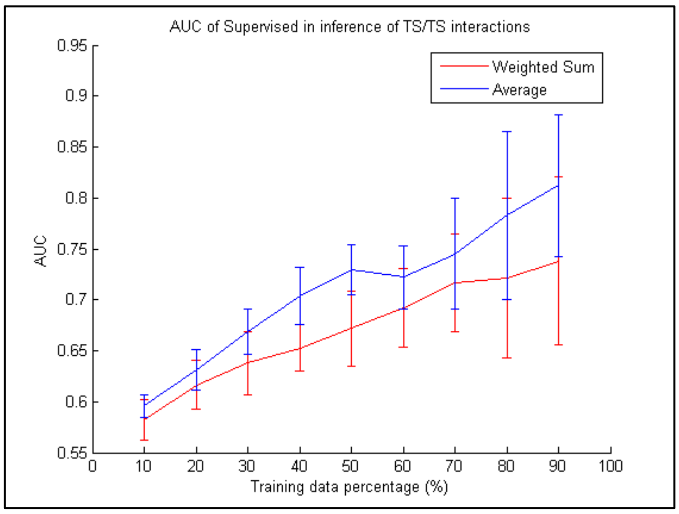

Figure 12: AUC of the inference of TR/TS interactions using the semi-supervised settings with various setups. The setups consist of using the Weighted Sum to build the input kernel or the Average Sum of the data sources. 80% of the proteins are used as training set.

Finally, in Figure 13 we can see the performance of the semi-supervised setting using the different setups for the inference of TS/TS interactions.

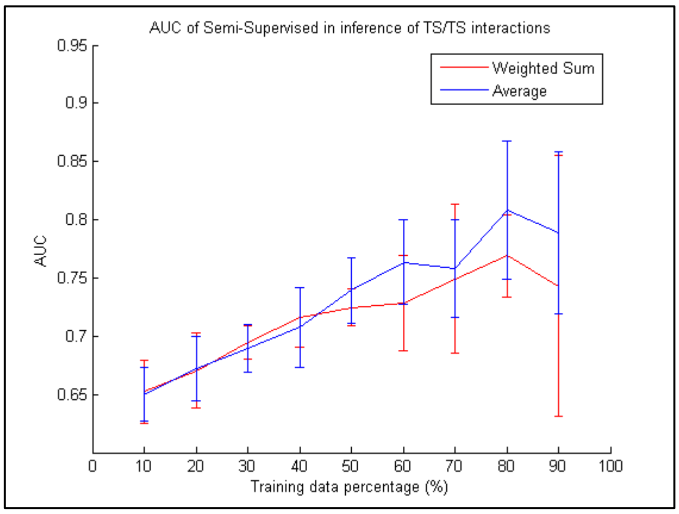

Figure 13: AUC of the inference of TS/TS interactions using the semi-supervised settings with various setups. The setups consist of using the Weighted Sum to build the input kernel or the Average Sum of the data sources. 80% of the proteins are used as training set.

Both setups show similar performance when we use small training percentages, from 10% to 40%. However, the MKL outperforms the average combination when larger percentages are used. The MKL achieves the highest AUC when we use 80%.

When we increase the data until we use 90% of the data in the training phase, almost the whole set of proteins, we can see that both setup experiment a performance decrement.

4.6. Comparison with other inference methods

It is interesting to compare the performance of the implementation of the IOKR done in this work with other state-of-the-art machine learning methods for the inference of PPI networks. I have chosen the support vector machine classifier with kernels on pairs of proteins developed in [16] and described in Section 2.3.2, and the OK3 developed in [6] and described in Section 2.3.3.

The three methods have been tested using the data from the Secretory Machinery of the yeast *Saccharomyces cerevisiae*. In every method, 80% of the proteins of the PPI network have been used in the training phase. As I have done in the previous experiments, I will analyze separately the performance of the methods over the two types of interactions.

I have chosen the best setup for each setting of the IOKR to be compared with the other methods. In the case of the supervised setting I have selected the Multiple Kernel Learning to build the input kernel and the parameters selected on the Section 5.4. On the other hand, for the semi-supervised setting the Average sum performs better, thus, this setup is used in the comparison.

Figure 11 shows the ROC curves of the different methods. Analyzing the ROC curves I can state that there is a slight difference between the support vector machine classifier with kernels on pairs of proteins and the semi-supervised setting of the IOKR. These two methods show better ROC curves than the others.

If we focus our attention on the AUC scores given in the legend of Figure 11 we can confirm the slight differences of performance of the methods. Moreover, I would point out the improvement of performance of the support vector machine classifier with kernels on pairs of proteins over the OK3, with the lowest AUC. I can state that the support vector machine classifier with kernels on pairs of proteins is the best to infer the TR/TS interactions of the PPI network of the methods considered.

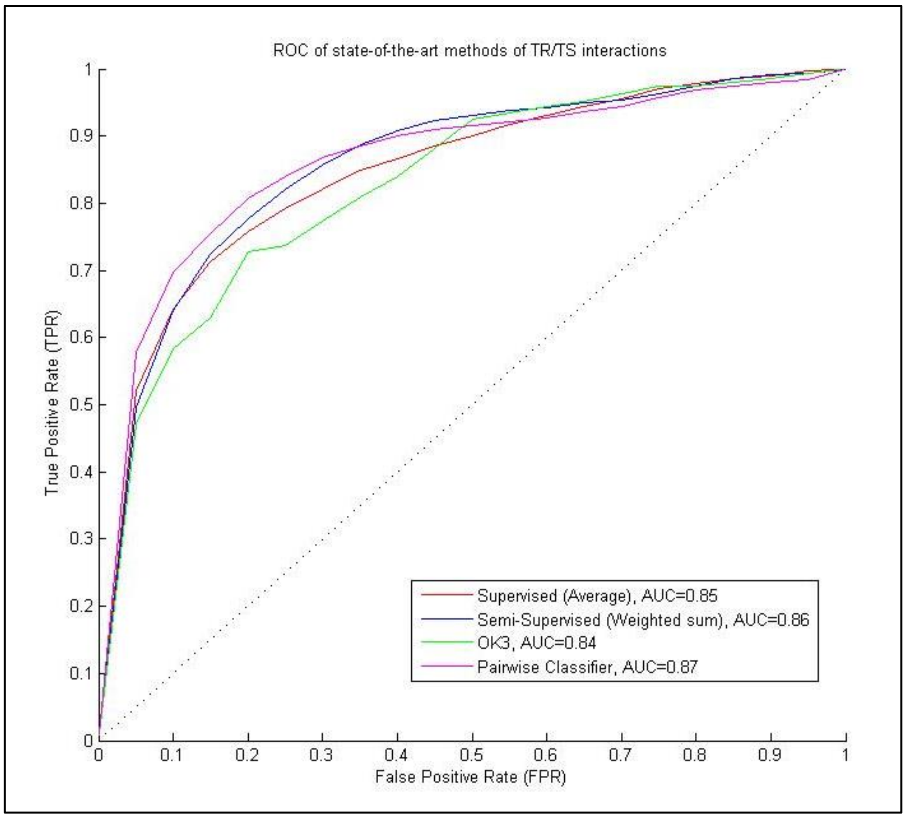

Figure 14: ROC curves of state-of-the-art-methods compared to the implementation of the IOKR on the inference of the TR/TS interactions. The IOKR is represented by the supervised and semi-supervised setting. I choose the best setup of each setting for the comparison, which is the Average Sum on the supervised setting and the Weighted Sum on the semi-supervised. All methods use 80% of the proteins as training set.

In Figure 12 we can see the visualization of the ROC curves of the methods for the inference of the TS/TS interactions. Contrary to the previous figure, we observe huge differences on the ROC curves. In this case the OK3, with the lowest performance on the TR/TS interactions, has the best ROC curve. On the other hand, the support vector machine classifier with kernels on pairs of proteins has the worst performance. I would point out that the differences between the settings of the IOKR are greater in this experiment.

Analyzing the AUC scores, we can find that the difference of performance between the OK3 and the support vector machine classifier with kernels on pairs of proteins is really significant. I can state that the OK3 is the best for the inference of TS/TS interactions in the PPI network considered.

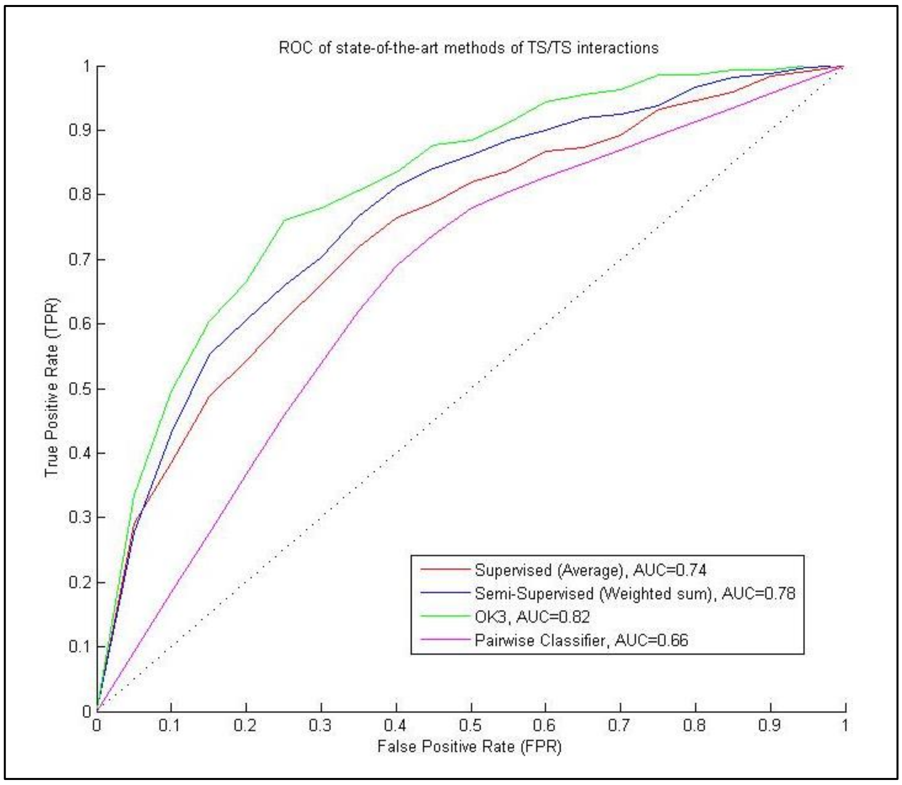

Figure 15: ROC curves of state-of-the-art-methods compared to the implementation of the IOKR on the inference of the TS/TS interactions. The IOKR is represented by the supervised and semi-supervised setting. I choose the best setup of each setting for the comparison, which is the Average Sum on the supervised setting and the Weighted Sum on the semi-supervised. All methods use 80% of the proteins as training set.

5. Discussion

After I have presented the experiments to analyze the performance of the implementation of the IOKR, I discuss in this section the obtained results. I go over the subsections of the previous sections to explain the findings.

Parameter tuning

I have found that in the supervised setting the value of the parameter β_1, which controls the grade of diffusion of the diffusion kernel, does not affect the performance of the classifier significantly. In the case of the semi-supervised setting, after testing different pairs of values of β_1 and β_2 I could state that using different values of this parameters does not change the performance of the classifier significantly, meaning that diffusing more into the graph [6] does not affect the performance of the classifier.

In the parameter tuning of the semi-supervised setting, testing a range of values for λ_1 from 0.1 to 1.0, I have found that the best performance is achieved when I use a value around 0.6. This means that the complexity of the model still being not too complex.

In the parameter tuning of the semi-supervised setting I have found that it works better with a high value of λ_1, around 0.8. The range of values of λ_2 tested initially did not show significant differences to select one of them as the value with the best performance. However, I noticed a slight decrement when the value of λ_2 was decreased.

A new experiment with a larger range of λ_2 values showed that lower values of λ_2, around 0.001, produce a classifier with better performance. A small smoothness constraint is sufficient effective for improving the performance of the semi-supervised approach over the supervised. On the other hand, the high value of λ_1 indicate that the method requires the use of a complex model to predict the label of the interactions.

Supervised vs. semi-supervised

In the experiments I could compare the performance of the settings of the IOKR. I have not found significant differences in the performance of the settings for the inference of TR/TS interactions. A setting outperforms the other in some data percentages.

On the other hand, in the inference of TS/TS interactions there exists a dominant, which is the semi-supervised setting. This setting has outperformed the supervised setting in each of the percentages tested.

Consequently, although the computation of the semi-supervised setting has a higher cost due to the introduction of information about the proteins of the testing set, this approach is worthy for the inference of the PPI network. This is more remarkable when we try to infer TS/TS interactions which are more difficult due to the lack of information about the instances of the testing set.

Individual Data Sources

The experiments of the individual data sources have provided interesting findings. First, I have found that the MKL gives the highest weights to BLAST and GTG data source.

Analyzing the performance of the individual data sources we can see that in the inference of TR/TS interactions the GTG and BLAST outperform most of the other data sources. Moreover, in this case the linear regression shows that a data source with a greater weight usually has a better performance.

On the other hand, I have found differences in the results of the inference of TS/TS interactions. In this case the BLAST data source obtains a performance of medium quality. However, the GTG data source still outperforms most of the other data sources. Analyzing the linear regression, we can see that a greater weight is no a sign of better performance.

In both sets of interactions I have found that the expression and localization data sources have some of the best performance, although the MKL gives them small weights. The great performance of this data sources is not strange, it is the cause that these kinds of data has been used in most of the PPI inference tasks [16].

Multiple Kernel Learning

After the experiments done using the Multiple Kernel Learning on the two settings I have found interesting results. First, I can state that the semi-supervised setting improves its performance using the weighted sum to compute the input kernel. However, the supervised setting obtains worse results, achieving better performance using the uniform combination of the data sources.

The semi supervised setting works better with training percentages from 50% to 80%, experimenting performance decrements when we use 90% of the data in the training phase. However, the supervised setting has experimented greater increments of performance using training percentages from 60% to 90%.

I have observed that in both settings the differences between the setups are slighter using small training percentages, up to 40%. Moreover, supervised and semi-supervised setting shows greater differences in the performance in the inference of TS/TS interactions. This is because this set is harder to infer, then, improvements in the methods are more visible in this set of interactions.

Comparison with other models

The results presented in the Section 4.7 has given a general picture of the position of the implementation of the IOKR of this work respect to some state-of-the-art machine learning methods for the inference of PPI networks.

The model has not achieved the best performance in any of the two cases considered: the inference of TR/TS interactions and the inference of TS/TS interactions. The support vector machine classifier with kernels on pairs of proteins and the OK3 have shown the best performance. However, the semi-supervised setting has shown a good performance on the TR/TS interactions, very close to the support vector machine classifier with kernels on pairs of proteins, with an AUC of 0.87.

On the other hand, even though the support vector machine classifier with kernels on pairs of proteins and OK3 have shown high performance on the inference of one kind of protein interactions they have failed on the other type. Nevertheless, the settings of the IOKR have shown acceptable performances in both types of interactions.

6. Conclusion

The implementation of the IOKR has not shown the best results on the inference of PPI. However, the obtained results for this implementation and the OK3 support the idea of the Kernelization of Regression models to build classifiers.

After analyzing the results of the Multiple Kernel Learning, I can state that in task inferences more data is not a synonym of a better performance. However, the importance is how the different features are combined to extract rich information from the different data sources.

The tested protein-protein prediction network has a small size. This causes those experiments with high percentage of training data gives a huge variation. In future experiments I would like to test the implementation with bigger PPI networks.

After finishing this project, I have been able to notice the difficulty of the application of machine learning methods for the inference of biological networks. Specially, about how to treat the data sources.

As a future work, I would propose a transfer learning using the semi-supervised IOKR to infer PPI networks of other organisms such as other kinds of yeast or humans while training the model on *Saccharomyces cerevisiae*.

7. Bibliography

[1] Feizi A, Österlund T, Petranovic D, Bordel S, Nielsen J, *Genome-Scale Modeling of the Protein Secretory Machinery in Yeast* (PLoS ONE 8(5): e63284. doi:10.1371/journal.pone.0063284, 2013).

[2] Alberts B, Johnson A, Lewis J, et al, *Molecular Biology of the Cell* (Garland Science, 4th edition, 2002)

[3] De Las Rivas J., Fontanillo C., *Protein–Protein Interactions Essentials: Key Concepts to Building and Analyzing Interactome Networks* (PLoS Comput Biol 6(6): e1000807. doi: 10.1371/journal.pcbi.1000807, June 24, 2010).

[4] Thoms Hofmann, Bernhard Schölkopf, Alexander J. Smola, *Kernel Methods in Machine Learning* (The Annal of Statistics, 2008, Vol. 36, No. 3) 1171–1220.

[5] Jean-Philippe Vert, *Reconstruction of biological network by supervised machine learning approaches* (H. Lodhi and S. Muggleton (Eds.), Elements of Computational Systems Biology, Wiley, 2010) 189-212.

[6] Pierre Geurts, Nizar Touleimat, Marie Dutreix, Florence d'Alché-Buc, *Inferring biological networks with output kernel trees* (BMC Bioinformatics, 8 (Suppl 2):S4, 2007).

[7] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction* (Springer Series in Statistics, February 2009), chapter 3.

[8] Max Welling, *Kernel ridge Regression* (University of California, class notes).

[9] Céline Brouard, Marie Szafranski, Florence d'Alché-Buc, *Regularized Output Kernel Regression applied to protein-protein interaction network inference* (Whistler, BC, Networks Across Disciplines: Theory and Applications, 2010).

[10] Céline Brouard, Marie Szafranski, Florence d'Alché-Buc, *Semi-supervised Penalized Output Kernel Regression for Link Prediction* (ICML, 2011) 593-600.

[11] Corinna Cortes, Mehryar Mohri, Afshin Rostamizadeh, *Algorithms for Learning Kernels Based on Centered Alignment* (Journal of Machine Learning Resarch 13, 2012) 795-828.

[12] Andrew P. Bradley, *The use of the Area Under the Curve in the evaluation of machine learning algorithms* (Pattern Recognition, Vol. 30, No. 7, pp. 1997) 1145-1159

[13] Ye J., McGinnis S, & Madden T.L., *BLAST: improvements for better sequence analysis* (Nucleic Acids Res. 34:W6-W9, 2006).

[14] Heger A., Mallick S., Wilson C., Holm L., *The global trace graph, a novel paradigm for searching protein sequence database* (Bioinformaticas 23 (18), 2007), 2361-7.

[15] Arvas M., Biau G., Vert J.P., *InterProScan - An integration platform for the signature-recognition methods in InterPro* (Bioinformaticas 17 (9), 2001), 847-8.

[16] Jana Kludas, Fitsum Tamene, Juho Rousu, *Supervised and unsupervised biological network inference from multiple 'omic data.*

[17] Frank Starmer, At: http://frank.itlab.us/photo_essays/wrapper.php?nephila_2002_dna.html (Accessed on 05.06.14)

[18] Fimon006, *Protein Interaction Network for TMEM8A* (2013) At: http://commons.wikimedia.org/wiki/File:Protein_Interaction_Network_for_TMEM8A.png (Accessed on 01.06.14)

Appendix I: MATLAB Code

I. Main function

```
% New framework to implement input-output kernel regression
% Option = 1 -> Supervised setting
% Option = 2 -> Semi-supervised setting
% res_filename -> Name of the file where the results will be saved
(without extension)
% method -> 'summean': sum of the input kernels (default), 'mkl':
multiple kernel learning

function ppiPredictionFramework(option, method),

  %%%%%%%%% Fixed parameters %%%%%%%%%%%%

  % Betas
  Beta1 = 1.0;
  Beta2 = 1.0;

  % Lambdas
  lambda1 = 0.6;
  lambda2 = 0.001;

  % Cross validation
  cross_validation_limit = 10;

  % training percentage value range
  tr_perc_range = [0.1, 0.2, 0.3, 0.5, 0.8, 0.9];
  tr_perc_labels = [10, 20, 30, 50, 80, 90];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  binarize = @(x,y) x>y;
  isequpos = @(x,y) x==y && y==1;
  isequneg = @(x,y) x==y && y==0;

  %%%% DATA Top14 %%%%%%%%%%%%%%%
  featureNames =
{'expression','localization','blasts_2012','FPrintScan_2012','Gene3
D_2012', 'gtgs_new_red', 'HMMPanther_2012', 'PatternScan_2012',
'HMMPfam_2012', 'HMMPIR_2012', 'ProfileScan_2012',
'protein_clusters', 'HMMSmart_2012', 'superfamily_2012'};
  feat_id = [1 2 3 4 5 6 7 8 9 10 11 12 13 14];
  featureNames(feat_id);

  selectLabels = 'SecrModel';

  % LABELS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  load(['labels' selectLabels '.mat'], 'ppinteraction','prunique');
  % INPUT FEATURES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  load(['feature' selectLabels 'Top14.mat'], 'feats','counts',
'featnames_new');
  fnames = featnames_new;
  % Binary version of the features
  featsBin = feats;
  if ~isempty(find(feats(:)>0 & feats(:)<1)),
    featsBin(find(featsBin(:)>0 & featsBin(:)<1))=1;
  end

  for tr_perc_index=1:size(tr_perc_range,2),
    training_percentage = tr_perc_range(tr_perc_index)

    %% Cross validation
    for cv=1:cross_validation_limit,
        fprintf('Processing iteration %d.\n', cv);

        % size(ppinteraction,1) -> Number of files = Number of
proteins
        [trset,tsset] = createFold(size(ppinteraction,1),
training_percentage);

        labels = ppinteraction([trset; tsset],[trset; tsset]); %
Sorting the matrix
        features = feats([trset; tsset],:); % Sorting the feature
matrix
        featuresBin = featsBin([trset; tsset],:); % Sorting the
binary feature matrix

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %                     OUTPUT KERNEL                   %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Compute the Degree Matrix
        labels_aux = labels(1:size(trset,1),1:size(trset,1));
        % Laplacian unnormalized
        L = diag(sum(labels_aux)) - labels_aux;
        % Difussion output kernel matrix
        Diff_Kernel = expm(-Beta1*L);
        % Normalize
        Diff_Kernel =  Diff_Kernel ./ (sqrt(diag(Diff_Kernel)) *
sqrt(diag(Diff_Kernel))');

        % Center Laplacian unnormalized matrix
        L_center = L - repmat(mean(L,1),size(L,1),1)...
            - repmat(mean(L,2),1,size(L,1))...
            + repmat(mean(L(:)),size(L,1),size(L,1));

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %                     INPUT KERNEL                    %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if(strcmp(method, 'mkl')),
            % MultipleKernelLearning
            [KKAll, rcorr_aux, KKsingle_ds] = mk_learning(feat_id,
counts, features, featuresBin,
L_center(1:size(trset,1),1:size(trset,1)));
```

```
            rcorr(cv,:) = rcorr_aux;
        else,
            [KKAll, rcorr_aux, KKsingle_ds] = input_mk(feat_id,
counts, features, featuresBin,
L_center(1:size(trset,1),1:size(trset,1)));
            rcorr(cv,:) = rcorr_aux;
        end


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %                COMPUTE PREDICTIONS                  %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if option == 1,
          % SUPERVISED SETTING %%%%%%%%%%%%%%%%%%%%%%%%%%
          A = supervised_setting(trset, KKAll, lambda1);
        else,
          % SEMI-SUPERVISED SETTING %%%%%%%%%%%%%%%%%%%%%
          A = semi_supervised_setting(trset, labels, KKAll,
lambda1, lambda2);
        end

        % Predictions
        predictions = A' * Diff_Kernel * A;


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %                EVALUATE CLASSIFIER                  %
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        % TRAINING/TEST
        %%%%%%%%%%%%%%%%%%%%
        [AUC, ROC, bthresh, accuracy] =
evaluate_trts(ppinteraction, trset, labels, predictions);
        AUC_trts_matrix(cv) = AUC;
        ROC_trts_matrix(cv) = {ROC};
        bthresh_v_trts_matrix(cv) = bthresh;
        accuracy_trts_matrix(cv) = accuracy;


        % TEST/TEST
        %%%%%%%%%%%%%%%%%%%%
        [AUC, ROC, bthresh, accuracy] =
evaluate_tsts(ppinteraction, trset, labels, predictions);
        AUC_tsts_matrix(cv) = AUC;
        ROC_tsts_matrix(cv) = {ROC};
        bthresh_v_tsts_matrix(cv) = bthresh;
        accuracy_tsts_matrix(cv) = accuracy;


        % TRAINING/TEST AND TEST/TEST
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        [AUC, ROC, bthresh, accuracy] =
evaluate_trts_and_tsts(ppinteraction, trset, labels, predictions);
        AUC_comb_matrix(cv) = AUC;
        ROC_comb_matrix(cv) = {ROC};
        bthresh_v_comb_matrix(cv) = bthresh;
        accuracy_comb_matrix(cv) = accuracy;


        % Each data source alone
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for ds=1:size(feat_id,2),

            if option == 1,
              %%%%%%%%% SUPERVISED SETTING %%%%%%%%%%%%%%
              A = supervised_setting(trset, KKsingle_ds(:,:,ds),
lambda1);
            else,
              %%%%%%%% SEMI-SUPERVISED SETTING %%%%%%%%%%
              A = semi_supervised_setting(trset, labels,
KKsingle_ds(:,:,ds), lambda1, lambda2);
            end

            % Predictions
            predictions = A' * Diff_Kernel * A;

            % TRAINING/TEST
            %%%%%%%%%%%%%%%%%%%%
            [AUC, ROC, bthresh, accuracy] =
evaluate_trts(ppinteraction, trset, labels, predictions);
            AUC_trts_matrix_ds(cv, ds) = AUC;

            % TEST/TEST
            %%%%%%%%%%%%%%%%%%%%
            [AUC, ROC, bthresh, accuracy] =
evaluate_tsts(ppinteraction, trset, labels, predictions);
            AUC_tsts_matrix_ds(cv, ds) = AUC;

            % TRAINING/TEST AND TEST/TEST
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            [AUC, ROC, bthresh, accuracy] =
evaluate_trts_and_tsts(ppinteraction, trset, labels, predictions);
            AUC_comb_matrix_ds(cv, ds) = AUC;

        end

    end; % for cv

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                    SAVE RESULTS                     %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if(strcmp(method, 'mkl')),
        if option == 1,
            resultfile = [ 'MKL_SUPERVISED_RESULTS_PATH'
int2str(tr_perc_labels(tr_perc_index)) ];
        else,
            resultfile = [ 'MKL_SEMISUPERVISED_RESULTS_PATH'
int2str(tr_perc_labels(tr_perc_index)) ];
        end
    else,
        if option == 1,
            resultfile = [ 'SUM_SUPERVISED_RESULTS_PATH'
int2str(tr_perc_labels(tr_perc_index)) ];
        else,
            resultfile = [ 'SUM_SEMISUPERVISED_RESULTS_PATH'
int2str(tr_perc_labels(tr_perc_index)) ];
        end
    end

    % TRAINING/TEST
    %%%%%%%%%%%%%%%%%%%
    ROC_trts = squeeze(ROC_trts_matrix(:));
    AUC_trts = squeeze(AUC_trts_matrix(:));
    accuracy_trts = squeeze(accuracy_trts_matrix(:));

    ROC_trts_average = averageROC(ROC_trts);
    AUC_trts_average = sum(AUC_trts)/size(AUC_trts,1);
    accuracy_trts_avg = sum(accuracy_trts)/size(accuracy_trts,1);

    % TEST/TEST
    %%%%%%%%%%%%%%%%%%%
    ROC_tsts = squeeze(ROC_tsts_matrix(:));
    AUC_tsts = squeeze(AUC_tsts_matrix(:));
    accuracy_tsts = squeeze(accuracy_tsts_matrix(:));

    ROC_tsts_average = averageROC(ROC_tsts);
    AUC_tsts_average = sum(AUC_tsts)/size(AUC_tsts,1);
    accuracy_tsts_avg = sum(accuracy_tsts)/size(accuracy_tsts,1);

    % TRAINING/TEST AND TEST/TEST
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ROC_comb = squeeze(ROC_comb_matrix(:));
    AUC_comb = squeeze(AUC_comb_matrix(:));
    accuracy_comb = squeeze(accuracy_comb_matrix(:));

    ROC_comb_average = averageROC(ROC_comb);
    AUC_comb_average = sum(AUC_comb)/size(AUC_comb,1);
    accuracy_comb_avg = sum(accuracy_comb)/size(accuracy_comb,1);

    % Save results

    % Average the correlation scores for each data source
    for ds=1:size(feat_id,2),
        for cv=1:cross_validation_limit,
            rcorr_aux(cv) = rcorr(cv,ds);
        end
        rcorr_avg(ds) = sum(rcorr_aux)/cross_validation_limit;
    end

    save(resultfile , 'rcorr', 'rcorr_avg', 'lambda1','lambda2',...
        'AUC_trts', 'AUC_trts_average', 'ROC_trts',
'ROC_trts_average', 'accuracy_trts',...
        'AUC_tsts', 'AUC_tsts_average', 'ROC_tsts',
'ROC_tsts_average', 'accuracy_tsts',...
        'AUC_comb', 'AUC_comb_average', 'ROC_comb',
'ROC_comb_average', 'accuracy_comb');
```

```
    % Each data source alone
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for ds=1:size(feat_id,2),

        if(strcmp(method, 'mkl')),
            if option == 1,
                resultfile = ['MKL_SUP_DATASOURCE_RESULT_PATH'
                    'ppi_prediction_sup_tr_perc_'
int2str(tr_perc_labels(tr_perc_index)) '_'
featureNames{feat_id(ds)} ];
            else,
                resultfile = ['MKL_SEMISUP_DATASOURCE_RESULT_PATH'
                    'ppi_prediction_semisup_tr_perc_'
int2str(tr_perc_labels(tr_perc_index)) '_'
featureNames{feat_id(ds)} ];
            end
        else,
            if option == 1,
                resultfile = ['SUM_SUP_DATASOURCE_RESULT_PATH'
                    'ppi_prediction_sup_tr_perc_'
int2str(tr_perc_labels(tr_perc_index)) '_'
featureNames{feat_id(ds)} ];
            else,
                resultfile = ['SUM_SEMISUP_DATASOURCE_RESULT_PATH'
                    'ppi_prediction_semisup_tr_perc_'
int2str(tr_perc_labels(tr_perc_index)) '_'
featureNames{feat_id(ds)} ];
            end
        end

        % TRAINING/TEST
        %%%%%%%%%%%%%%%%%%%
        AUC_trts = squeeze(AUC_trts_matrix_ds(:,ds));

        AUC_trts_average = sum(AUC_trts)/size(AUC_trts,1);

        % TEST/TEST
        %%%%%%%%%%%%%%%%%%%
        AUC_tsts = squeeze(AUC_tsts_matrix_ds(:,ds));

        AUC_tsts_average = sum(AUC_tsts)/size(AUC_tsts,1);

        % TRAINING/TEST AND TEST/TEST
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        AUC_comb = squeeze(AUC_comb_matrix_ds(:,ds));

        AUC_comb_average = sum(AUC_comb)/size(AUC_comb,1);

        % Save results
        save(resultfile , 'lambda1', 'lambda2',...
        'AUC_trts', 'AUC_trts_average', ...
        'AUC_tsts', 'AUC_tsts_average', ...
        'AUC_comb', 'AUC_comb_average');

    end
  end % end training percentage for
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %             HELP FUNCTIONS (next sections)             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end
```

II. Input kernel

```
  % Split the samples in trainging and test sets
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [trset,tsset] = createFold(Nall, tr_percentage),

    prRand = randperm(Nall);

    Ntr = round(Nall*tr_percentage);

    trset = prRand(1:Ntr)';
    tsset = prRand(Ntr+1:end)';

  end

  % Input Multiple Kernel
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [KKAll, rcorr, KKsingle] = input_mk(feat_id, counts,
feats, featsBin, labelsC),
    % Input Kernel based on combining different data sources

    for ds=1:size(feat_id,2), % over all data sources

      % aid -> first index of feats from data source ds
      % bid -> last index of feats from data source ds
      if ds==1,
        aid = 1;
        bid = counts(ds);
      else,
        aid = sum(counts(1:ds-1))+1;
        bid = sum(counts(1:ds));
      end;

      % Take only current data source and reorder samples in
training/testing
      feat_cur = feats(:,aid:bid);
      feat_cur_bin = featsBin(:,aid:bid);

      if strcmp(featureNames{feat_id(ds)},'expression'),
        % RBF kernel, improves correlation and accuracy %%%%
        sigm = 1;

        n1sq = sum(feat_cur'.^2,1);
        n1 = size(feat_cur',2);

        D = (ones(n1,1)*n1sq)' + ones(n1,1)*n1sq -
2*feat_cur*feat_cur';

        KKsingle(:,:,ds) = exp(-D/(2*sigm^2));

        % Normalize kernel - 1-diagonal
        KKsingle(:,:,ds) = KKsingle(:,:,ds)./
(sqrt(diag(KKsingle(:,:,ds)))*sqrt(diag(KKsingle(:,:,ds)))' +
0.00000001);
```

```
      else, % Otherwise
        % Linear kernel %%%%%%%%%
        KKsingle(:,:,ds) = full(feat_cur*feat_cur');
        % Normalize kernel - 1-diagonal
        KKsingle(:,:,ds) = KKsingle(:,:,ds)./
(sqrt(diag(KKsingle(:,:,ds)))*sqrt(diag(KKsingle(:,:,ds)))' +
0.00000001);
      end;

      % Correlation
      KKdatasource = KKsingle(:,:,ds);

      % Center kernel matrix of the datasource ds
      KKdatasource = KKdatasource -
repmat(mean(KKdatasource,1),size(KKdatasource,1),1)...
          - repmat(mean(KKdatasource,2),1,size(KKdatasource,1))...
          +
repmat(mean(KKdatasource(:)),size(KKdatasource,1),size(KKdatasource
,1));

      % Compute the correlation between the data source and the
output
      KKdatasource_aux = KKdatasource(1:size(labelsC,1),
1:size(labelsC,1));
      rcorr(ds) =
sum(KKdatasource_aux(:).*labelsC(:))/(sqrt(sum(KKdatasource_aux(:).
^2))*sqrt(sum(labelsC(:).^2)));

    end; % for ds

    % Sum
    % KKAll = sum(KKsingle,3);
    % Mean
    KKAll = sum(KKsingle,3)/size(feat_id,2);

    % Normalize the input kernel matrix
    KKAll = KKAll./ (sqrt(diag(KKAll))*sqrt(diag(KKAll))' +
0.00000001);

  end

  % Multiple Kernel Learning on different data sources
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [KKAll, rcorr, KKsingle_aux] = mk_learning(feat_id,
counts, feats, featsBin, labelsC),
    % Input kernel based on combining different data sources

    for ds=1:size(feat_id,2), % over all data sources

      % aid -> first index of feats from data source ds
      % bid -> last index of feats from data source ds
      if ds==1,
        aid = 1;
        bid = counts(ds);
      else,
        aid = sum(counts(1:ds-1))+1;
        bid = sum(counts(1:ds));
      end;

      % Take only current data source and reorder samples in
training/testing
      feat_cur = feats(:,aid:bid);
      feat_cur_bin = featsBin(:,aid:bid);

      if strcmp(featureNames{feat_id(ds)},'expression'),
        % RBF kernel, improves correlation and accuracy %%%%%%%%%
        sigm = 1;

        n1sq = sum(feat_cur'.^2,1);
        n1 = size(feat_cur',2);

        D = (ones(n1,1)*n1sq)' + ones(n1,1)*n1sq -
2*feat_cur*feat_cur';

        KKsingle(:,:,ds) = exp(-D/(2*sigm^2));

        % Normalize kernel - 1-diagonal
        KKsingle(:,:,ds) = KKsingle(:,:,ds)./
(sqrt(diag(KKsingle(:,:,ds)))*sqrt(diag(KKsingle(:,:,ds)))' +
0.00000001);

      else, % Otherwise
        % Linear kernel %%%%%%%
        KKsingle(:,:,ds) = full(feat_cur*feat_cur');
        % Normalize kernel - 1-diagonal
        KKsingle(:,:,ds) = KKsingle(:,:,ds)./
(sqrt(diag(KKsingle(:,:,ds)))*sqrt(diag(KKsingle(:,:,ds)))' +
0.00000001);
      end;

      KKsingle_aux(:,:,ds) = KKsingle(:,:,ds);

      KKdatasource = KKsingle(:,:,ds);

      % Center kernel matrix of the datasource ds
      KKdatasource = KKdatasource -
repmat(mean(KKdatasource,1),size(KKdatasource,1),1)...
         - repmat(mean(KKdatasource,2),1,size(KKdatasource,1))...
         +
repmat(mean(KKdatasource(:)),size(KKdatasource,1),size(KKdatasource
,1));

      % Compute the correlation between the data source and the
output
      KKdatasource_aux = KKdatasource(1:size(labelsC,1),
1:size(labelsC,1));
      rcorr(ds) =
sum(KKdatasource_aux(:).*labelsC(:))/(sqrt(sum(KKdatasource_aux(:).
^2))*sqrt(sum(labelsC(:).^2)));

      % Weight the datasource
      KKsingle(:,:,ds) = KKsingle(:,:,ds).*rcorr(ds);
    end; % for ds
    % Mean
    KKAll = sum(KKsingle,3)/size(feat_id,2);

    % Normalize the input kernel matrix
    KKAll = KKAll./(sqrt(diag(KKAll))*sqrt(diag(KKAll))' +
0.00000001);

  end
```

III. Kernel Regression

```
  % SUPERVISED setting
  %%%%%%%%%%%%%%%%%%%%%%%
  function [A] = supervised_setting(trset, KKAll, lambda1),
    B = lambda1 * eye(size(trset,1),size(trset,1)) +
KKAll(1:size(trset,1),1:size(trset,1));
    A = B \ KKAll(1:size(trset,1),:);
  end

  % SEMI-SUPERVISED setting
  %%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [A] = semi_supervised_setting(trset, labels, KKAll,
lambda1, lambda2),

    U = zeros(size(trset,1), size(labels,1));
    U(:,1:size(trset,1)) = eye(size(trset,1));

    LKKAll = diag(sum(KKAll)) - KKAll;
    LKKAll = expm(-Beta2 * LKKAll);
    % Normalize matrix
    LKKAll = LKKAll ./ (sqrt(diag(LKKAll)) * sqrt(diag(LKKAll))');

    B = U/(lambda1 * eye(size(labels,1)) + KKAll * (U'*U) +
2*lambda2*KKAll*LKKAll);
    A = B * KKAll;

  end
```

IV. Result processing

```
  % Evaluate TR/TS
  %%%%%%%%%%%%%%%%%%%
  function [AUC, ROC, bthresh, accuracy] =
evaluate_trts(ppinteraction, trset, labels, predictions),

    Mat_test = ones(size(ppinteraction,1),size(ppinteraction,1));
    Mat_test(1:size(trset,1), 1:size(trset,1)) = 0;
    Mat_test(size(trset,1)+1:end, size(trset,1)+1:end) = 0;
    Mat_test = triu(Mat_test,1);
    indices_test = find(Mat_test == 1)';

    % Compare prediction and known labels
    [AUC, ROC, bthresh] = getAUCandROC(labels(indices_test)',
predictions(indices_test)');

    % Balanced accuracy
    accuracy = getAccuracy(labels(indices_test)',
predictions(indices_test)', bthresh);

  end

  % Evaluate TS/TS
  %%%%%%%%%%%%%%%%%%%
  function [AUC, ROC, bthresh, accuracy] =
evaluate_tsts(ppinteraction, trset, labels, predictions),

    Mat_test = ones(size(ppinteraction,1),size(ppinteraction,1));
    Mat_test(1:size(trset,1), :) = 0;
    Mat_test(:, 1:size(trset,1)) = 0;
    Mat_test = triu(Mat_test,1);
    indices_test = find(Mat_test == 1)';

    % Compare prediction and known labels
    [AUC, ROC, bthresh] = getAUCandROC(labels(indices_test)',
predictions(indices_test)');

    % Balanced accuracy
    accuracy = getAccuracy(labels(indices_test)',
predictions(indices_test)', bthresh);

  end

  % Evaluate TR/TS and TS/TS
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [AUC, ROC, bthresh, accuracy] =
evaluate_trts_and_tsts(ppinteraction, trset, labels, predictions),

    Mat_test = ones(size(ppinteraction,1),size(ppinteraction,1));
    Mat_test(1:size(trset,1), 1:size(trset,1)) = 0;
    Mat_test = triu(Mat_test,1);
    indices_test = find(Mat_test == 1)';
    % Compare prediction and known labels
    [AUC, ROC, bthresh] = getAUCandROC(labels(indices_test)',
predictions(indices_test)');

    % Balanced accuracy
    accuracy = getAccuracy(labels(indices_test)',
predictions(indices_test)', bthresh);

  end
```

V. Accuracy, ROC and AUC analysis

```
  % Get accuracy
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [accall] = getAccuracy(labels, predictions, bthresh),

    predict_label_bin = binarize(predictions,bthresh);

    TPos =
size(find(arrayfun(isequpos,predict_label_bin,labels)),1);
    TNeg =
size(find(arrayfun(isequneg,predict_label_bin,labels)),1);

    Pos = size(find(labels==1),1);
    if Pos==0,
      'loocv Warning - no positive examples!'
    end;

    Neg = size(find(labels==0),1);
    if Neg==0,
      'loocv Warning - no negative examples!'
    end;

    accall =  (0.5*TPos/Pos + 0.5*TNeg/Neg);

  end


  % Analysis using AUC and ROC
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [AUC, ROC, bthresh] = getAUCandROC(labels, predictions),

    TPR = 0; % True positive rate
    FPR = 0; % False positive rate
    TPRprev = 0;
    FPRprev = 0;
    AUC = 0;
    [pred_sort,idsort] = sort(predictions,'descend');
    labelsort = labels(idsort);

    Pos = size(find(labels==1),1);
    if Pos==0,
      'ppiFrame: Warning - no positive examples!'
    end;

    Neg = size(find(labels==0),1);
    if Neg==0,
      'ppiFrame: Warning - no negative examples!'
    end;

    %[pred_sort labelsort]
    i=1;
    lprev = -1000;
    ROC = [];
    min_distance = 100000;
```

```
 min_distance_index = -1;

  while i<=size(pred_sort,1),
      if pred_sort(i)~=lprev,
          ROC = [ROC; FPR/Neg TPR/Pos];
          AUC = AUC + calcarea(FPR,FPRprev,TPR,TPRprev);

          lprev = pred_sort(i);
          TPRprev = TPR;
          FPRprev = FPR;
      end;

      % Work out distance to point (0,1)
      distance = sqrt((0-FPR/Neg)^2+(1-TPR/Pos)^2);
      if distance < min_distance,
         min_distance = distance;
         min_distance_index = i;
      end

      if labelsort(i)==1,
          TPR = TPR+1;
      else
          FPR = FPR+1;
      end;

      i = i+1;
  end; % end for while

  % Work out distance to point (0,1)
  distance = sqrt((0-FPR/Neg)^2+(1-TPR/Pos)^2);
  if distance < min_distance,
     min_distance = distance;
     min_distance_index = i;
  end

  ROC = [ROC; FPR/Neg TPR/Pos];
  AUC = AUC + calcarea(FPR,FPRprev,TPR,TPRprev);
  AUC = AUC/(Pos * Neg);
  bthresh = pred_sort(min_distance_index);

end

% Calcule area under the ROC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A=calcarea(X1,X2,Y1,Y2),

  base = abs(X1-X2);
  height = (Y1+Y2)/2;
  A = base*height;

end

% Average ROC
%%%%%%%%%%%%%%%
function [ROCav] = averageROC(ROCset),
  ROCav = [];

  if size(ROCset,2)>size(ROCset,1),
    ROCset = ROCset';
  end;

  % Average ROC
  s = 1;
  for i=0:0.05:1,
    ROCav(s,1) = i;
    tprsum = 0;

    for k=1:size(ROCset,1)
      tprsum = tprsum+TPR_FOR_FPR(i,ROCset{k},size(ROCset{k},1));
    end;

    ROCav(s,2) = tprsum/size(ROCset,1);
    s = s+1;
  end;

end

% TPR for FPR
%%%%%%%%%%%%%%%
function [tpr] = TPR_FOR_FPR(fprsamp, ROC, npts),

  tpr = 0;
  j=1;

  while j<npts & ROC(j+1,1)<fprsamp,
    j=j+1;
  end;

  if ROC(j,1) == fprsamp,
    tpr = ROC(j,2);
  else,
    tpr = INTERPOLATE(ROC(j,:),ROC(j+1,:),fprsamp);
  end;

end

% Interpolate two ROC adjacent points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [tpr] = INTERPOLATE(roc_point1, roc_point2, fprsamp),
  % Linear interpolation
  tpr = roc_point1(2)+(roc_point2(2)-roc_point1(2))*(fprsamp-
roc_point1(1))/(roc_point2(1)-roc_point1(1));
end
```