



Kernel-based predictive modelling of drug–protein binding affinities

Anna Cichońska

- 1) University of Turku, Finland
- 2) Helsinki Institute for Information Technology HIIT, Aalto University, Finland
- 3) Institute for Molecular Medicine Finland FIMM, University of Helsinki, Finland

anna.cichonska@helsinki.fi

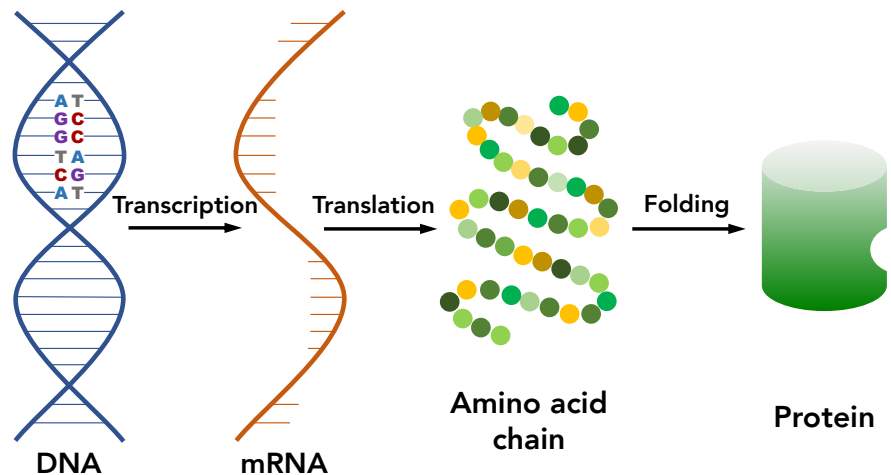
16 May 2019

Drug–protein interaction

- Drug-like chemical compounds execute their actions mainly by modulating cellular targets, including proteins, metabolites or even nucleic acids (DNA and RNA).

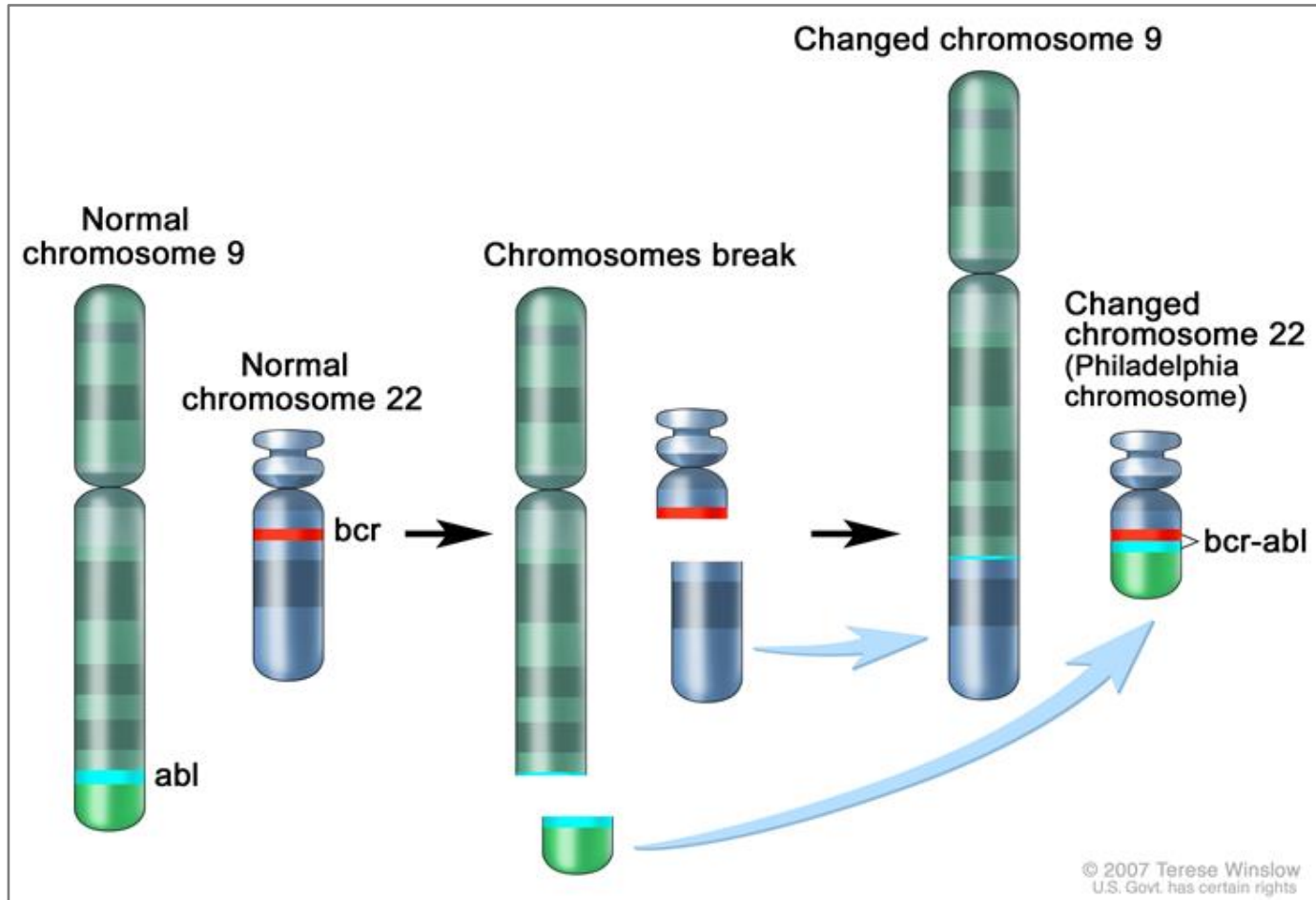
Proteins

- The largest class of drug targets.
- Proteins are assembled from amino acids using information encoded in genes.
- Perform a variety of important tasks, such as:
 - catalyzing chemical reactions (so called *enzymes*, e.g., kinases),
 - identifying and neutralizing foreign particles,
 - transporting other molecules,
 - providing structure and support for cells,& many more.



Drug-protein interaction

Imatinib-BCR-ABL



Off-target interactions

➤ Neutral

➤ Negative

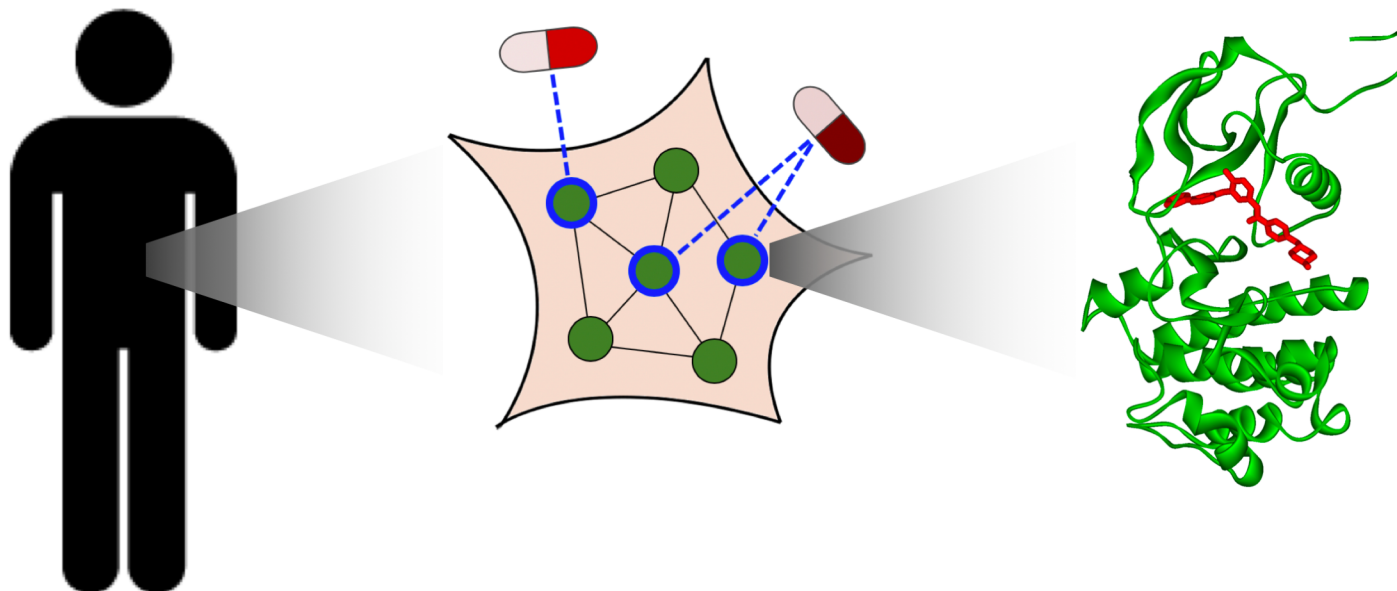
- Imatinib–c-ABL → cardiotoxic side effects.

➤ Positive

- Imatinib–KIT → treatment of gastrointestinal cancer.



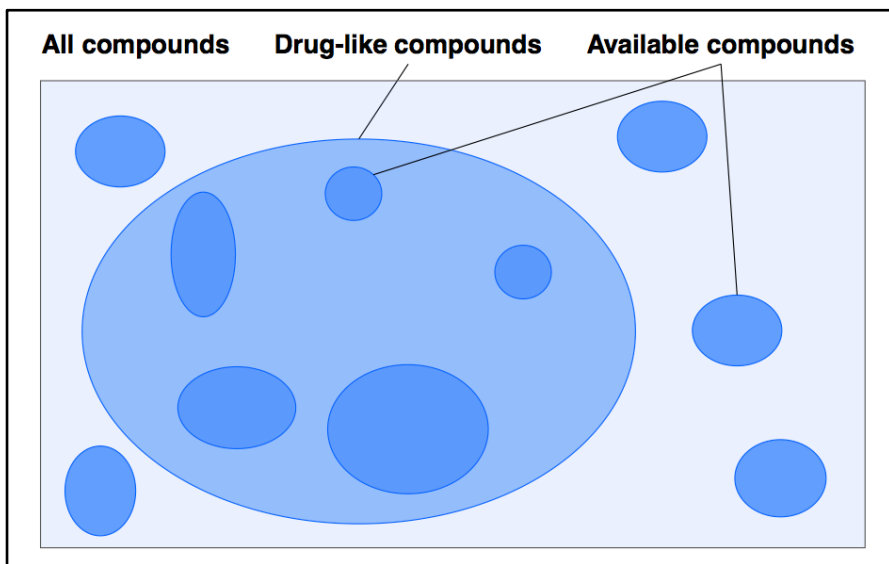
Drug-protein interaction profiling



- Expensive
- Time consuming

Enormous chemical universe

CHEMICAL SPACE



Only certain molecules have features consistent with good pharmacological properties (e.g. *Lipinski's rule of five*).

$\sim 10^{24}$!

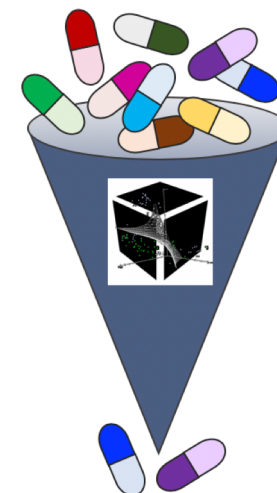
Lipinski's rule of five states that, in general, an orally-absorbed drug has no more than one violation of the following criteria:

- no more than 5 hydrogen bond donors;
- no more than 10 hydrogen bond acceptors;
- a molecular weight lower than 500 daltons;
- an octanol-water partition coefficient $\log P$ (a measure of lipophilicity) not greater than 5.

Note that the name of the rule originates from the fact that the cut-offs for all parameters are close to 5 or a multiple of 5.

Motivation for machine learning in drug discovery

- Experimental drug-protein interaction mapping is time consuming and expensive.
- Moreover, it is simply infeasible to determine all the possible drug-protein interactions in the laboratory (10^{20} - 10^{24} drug-like compounds!)
- The hypothesis is that machine learning models could provide fast, large-scale and systematic pre-screening of chemical probes, toward prioritization of the most potent interactions for further *in vitro* or *ex vivo* verification in the laboratory.



Computational drug screening

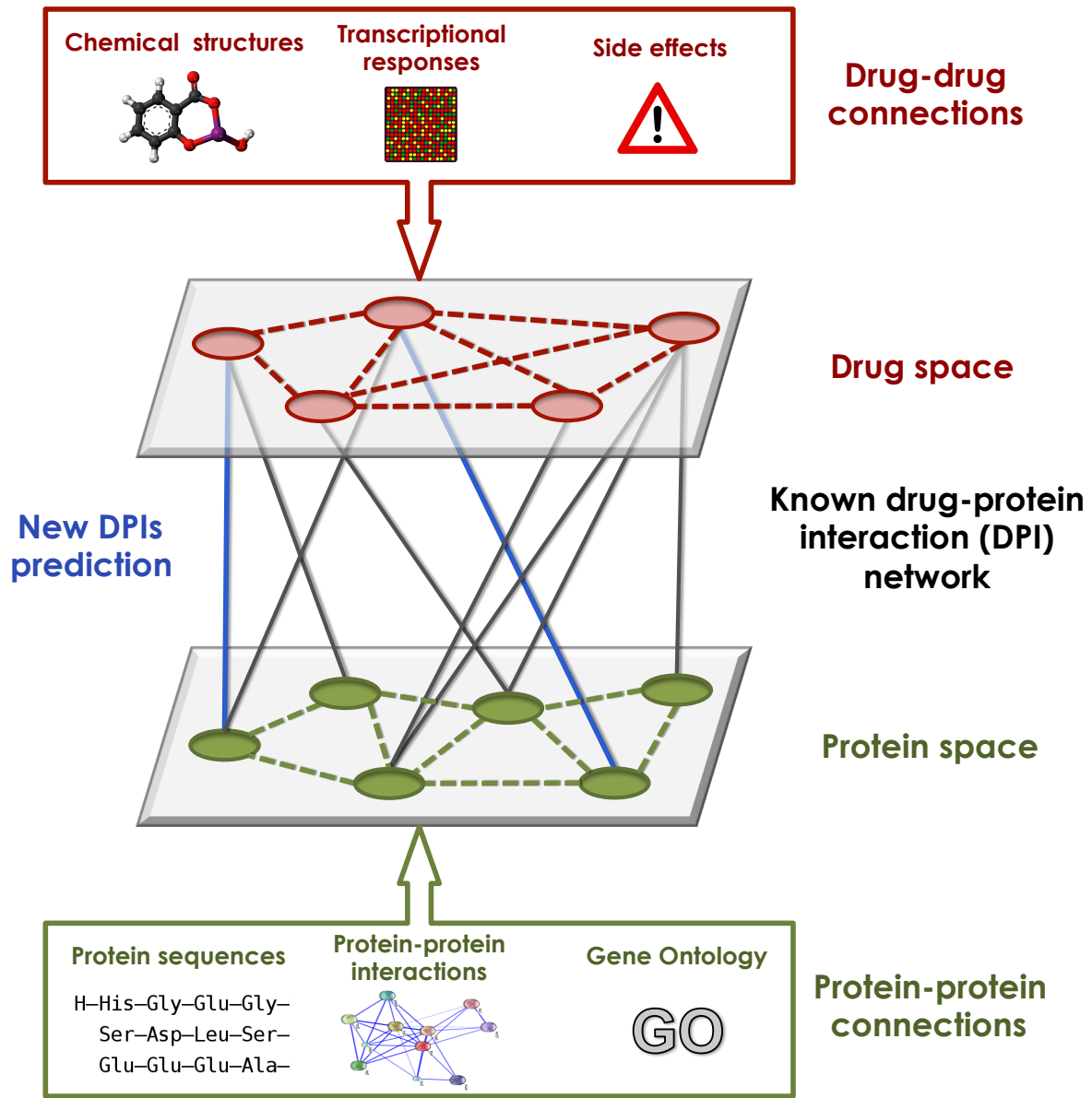


➤ Machine Learning

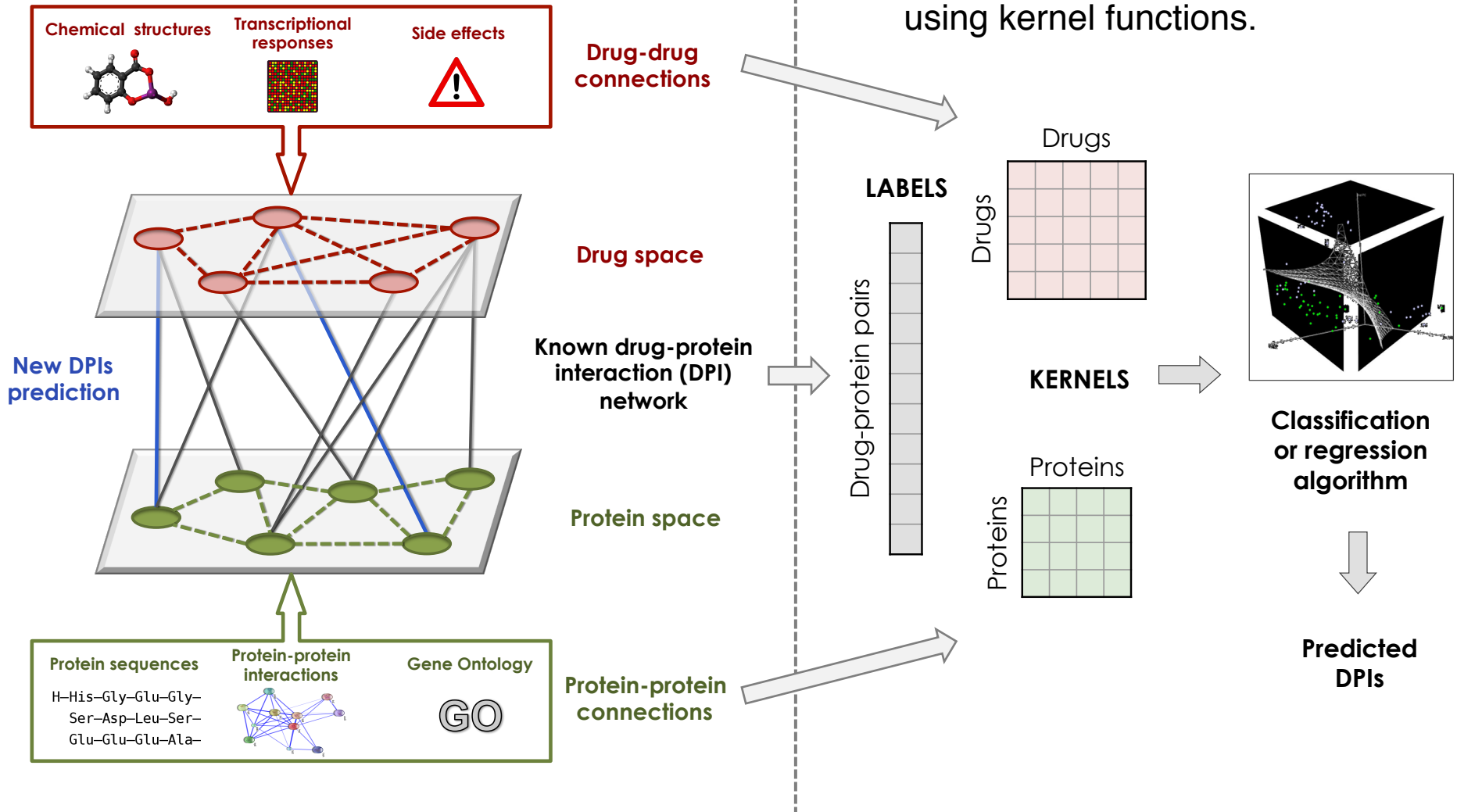
- The objective is to derive rules from the existing bioactivity data (a phase of learning from training data) in order to build predictive models that can be then applied to infer unmeasured drug-protein binding affinities (prediction phase).
- **Drug-based methods** (quantitative structure-activity relationship QSAR models)
Models trained using available bioactivity data + drug information.
- **Protein-based methods**
Models trained using available bioactivity data + protein information.
- **Systems-based methods** (proteochemometric models, pairwise models)
Models trained using available bioactivity data + both drug and protein information.
Assumption: similar drugs are likely to interact with similar proteins.

Systems-based DPI prediction methods

- Classification
interaction/no interaction
- Regression
quantitative binding affinity



Similarities between molecules can be encoded using kernel functions.



Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

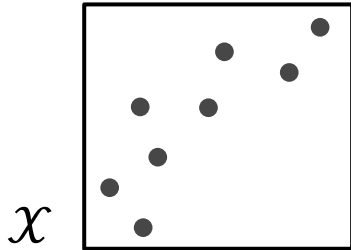
where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

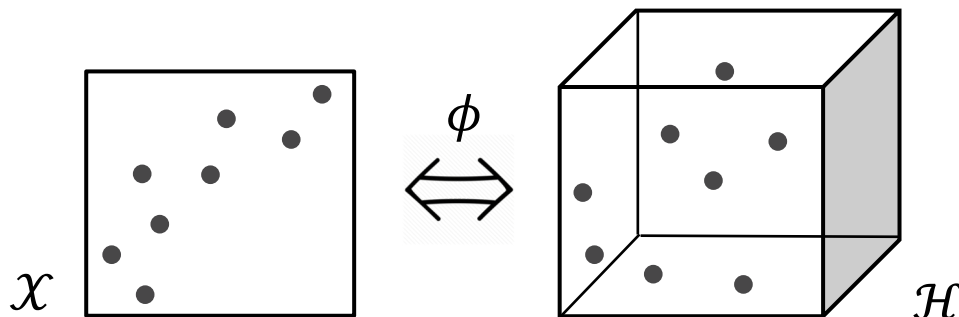


Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

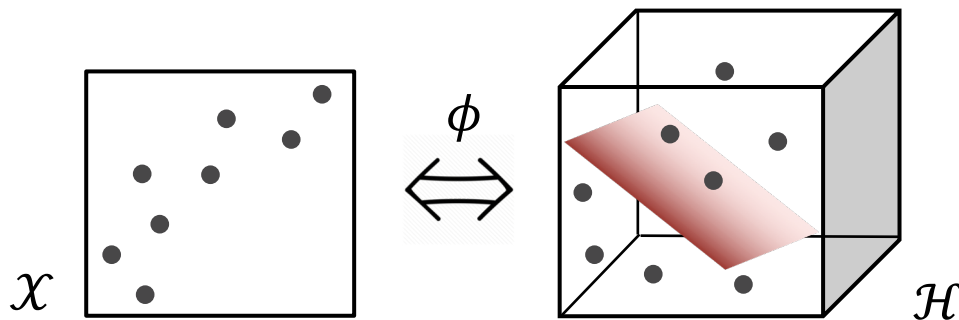


Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

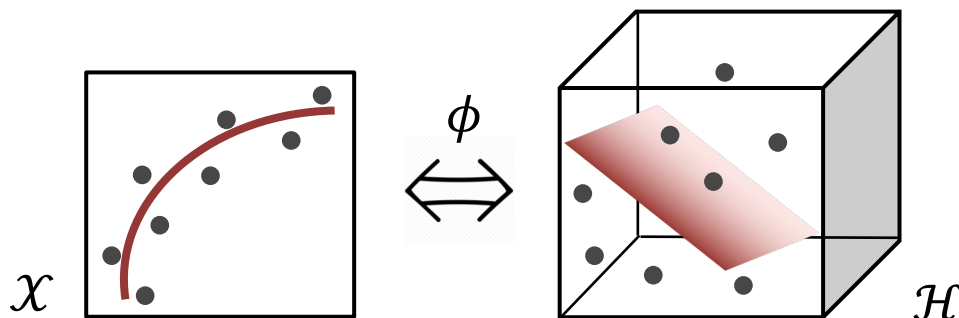


Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

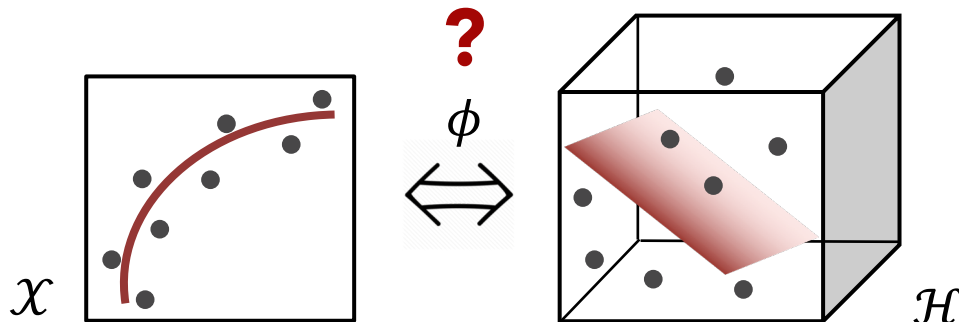


Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

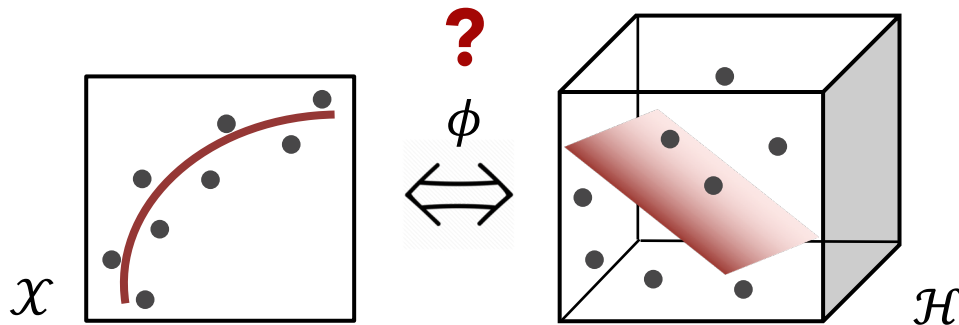


Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

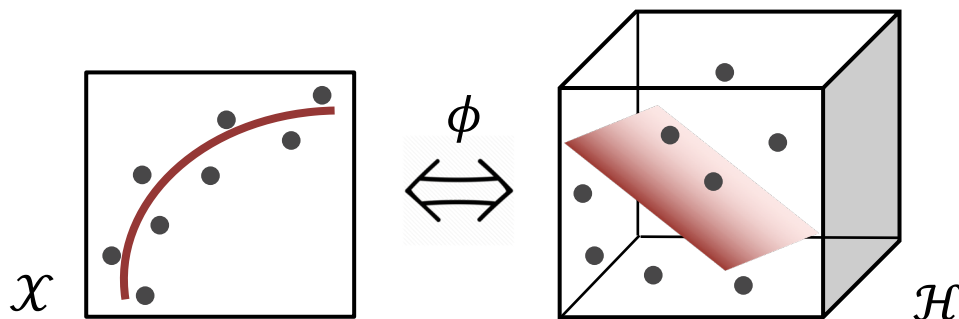


Kernels

- Kernels allow modelling **nonlinearities** in the data using well-established linear learning algorithms (in a computationally efficient manner).
- Formally, kernel is a function that for all instances $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ (e.g. drugs) satisfies

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle,$$

where ϕ denotes the mapping from the input space \mathcal{X} to an inner product high-dimensional feature space \mathcal{H} .

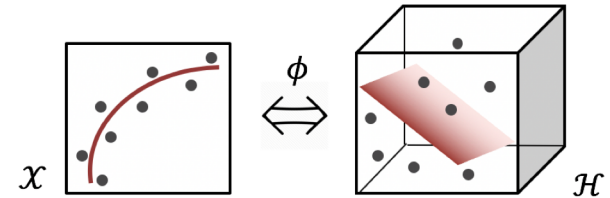


Example

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right)$$

Kernels

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$



➤ Kernel trick

It is possible to avoid the explicit computation of the mapping ϕ and define the kernel directly in terms of the original input features by replacing the inner product $\langle \cdot, \cdot \rangle$ with an appropriately chosen kernel function.

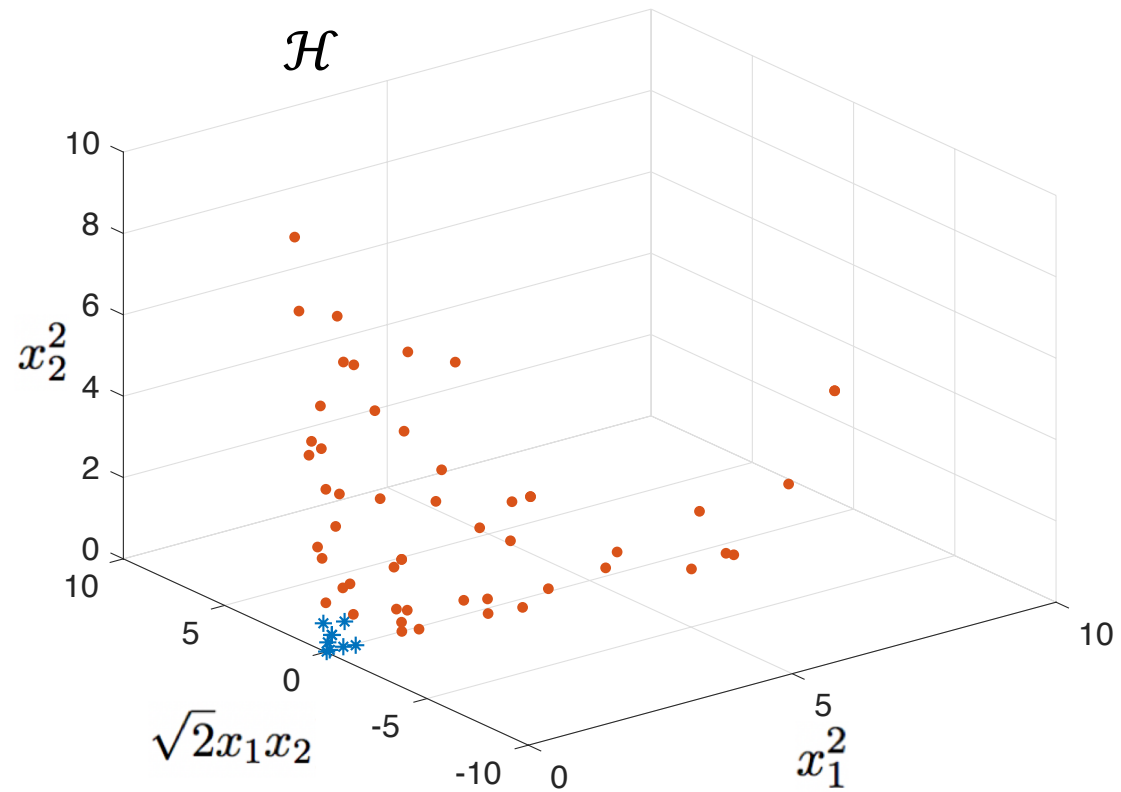
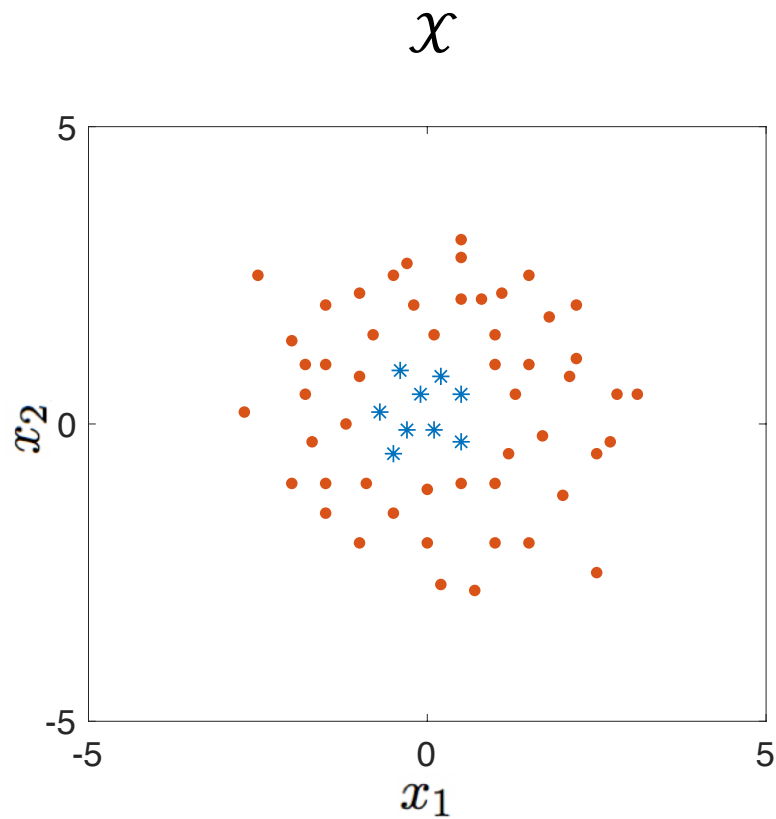
Example. Consider a two-dimensional input space together with the feature map:

$$\mathbf{x} = (x_1, x_2) \mapsto \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\begin{aligned} \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle &= \left\langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2) \right\rangle \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 = \langle \mathbf{x}, \mathbf{z} \rangle^2. \end{aligned}$$

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^2$$

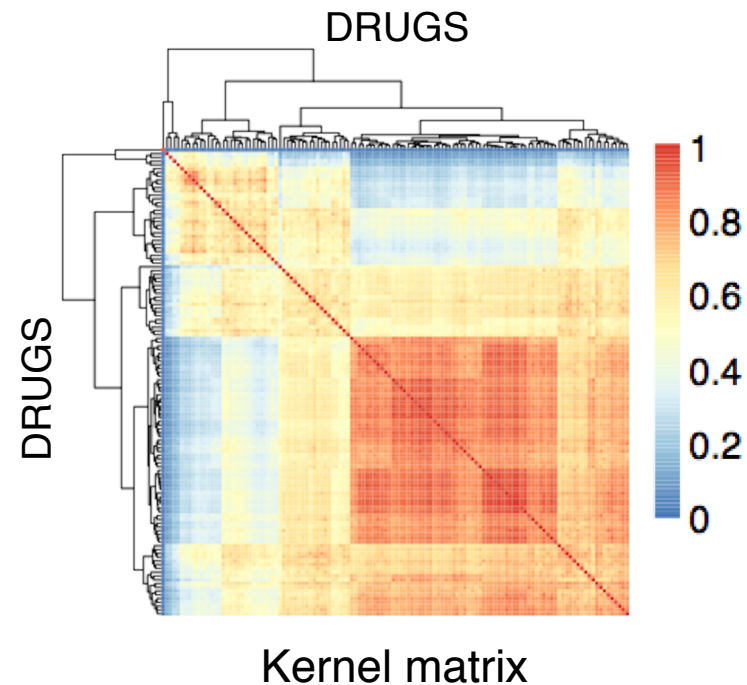
Kernels

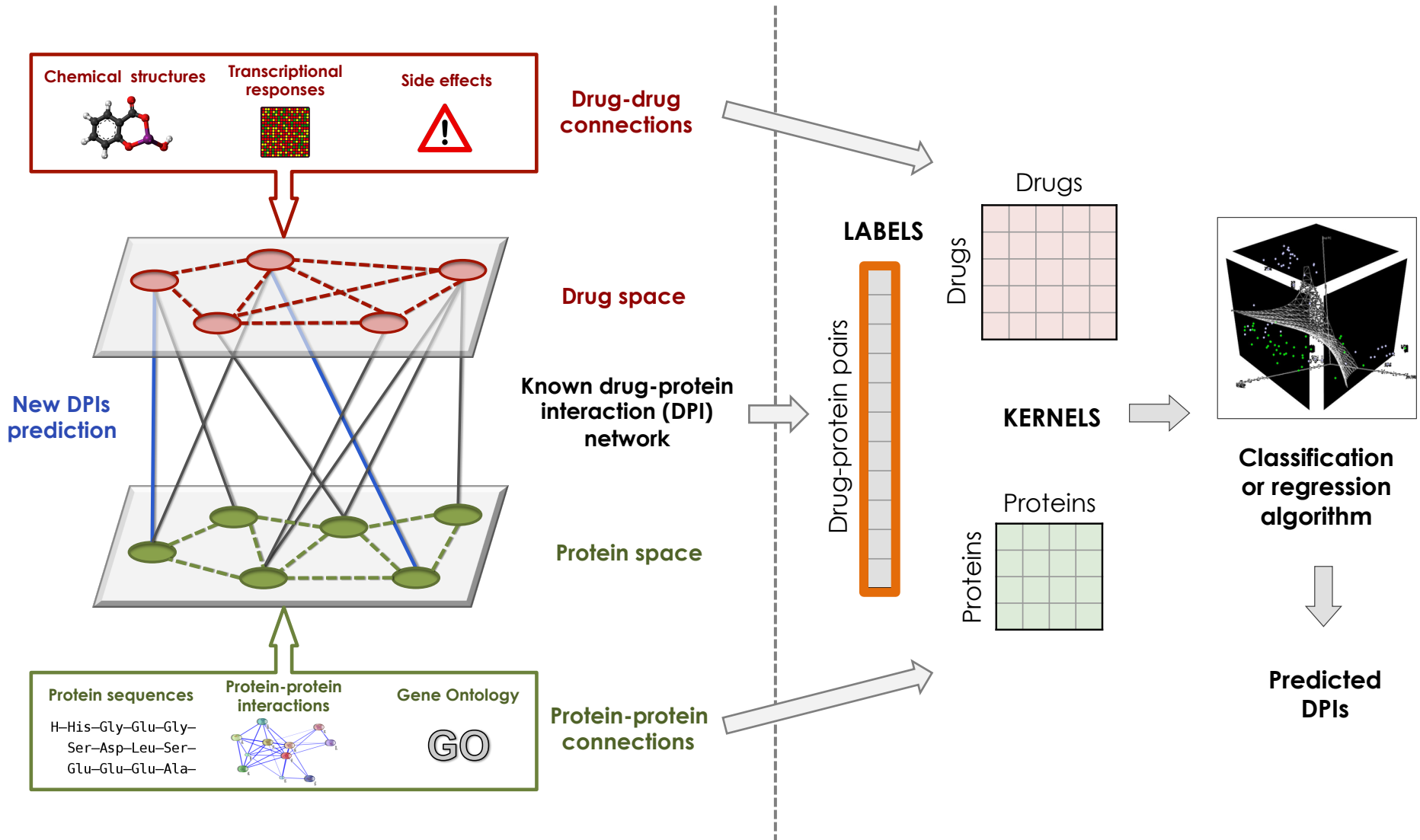


$$\mathbf{x} = (x_1, x_2) \mapsto \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

Kernels

- Kernels address the challenge of **#instances** (e.g. drugs) \ll **#features** (e.g. various chemical properties)
→ data appears only through the entries in the kernel matrix relating all pairs of instances.
- Kernels are well-suited for representing structured objects, such as molecules, that cannot always be accurately described by a standard feature vector.
- Kernel can be considered as a **similarity measure** between input instances.





Known interactions – bioactivity databases



<https://www.ebi.ac.uk/chembl/>

- Searchable and downloadable.
- Data manually extracted from the literature.
- Target Report Card, Compound Report Card.
- ~1.9 mln compounds in ChEMBL 25.



<https://pubchem.ncbi.nlm.nih.gov/>

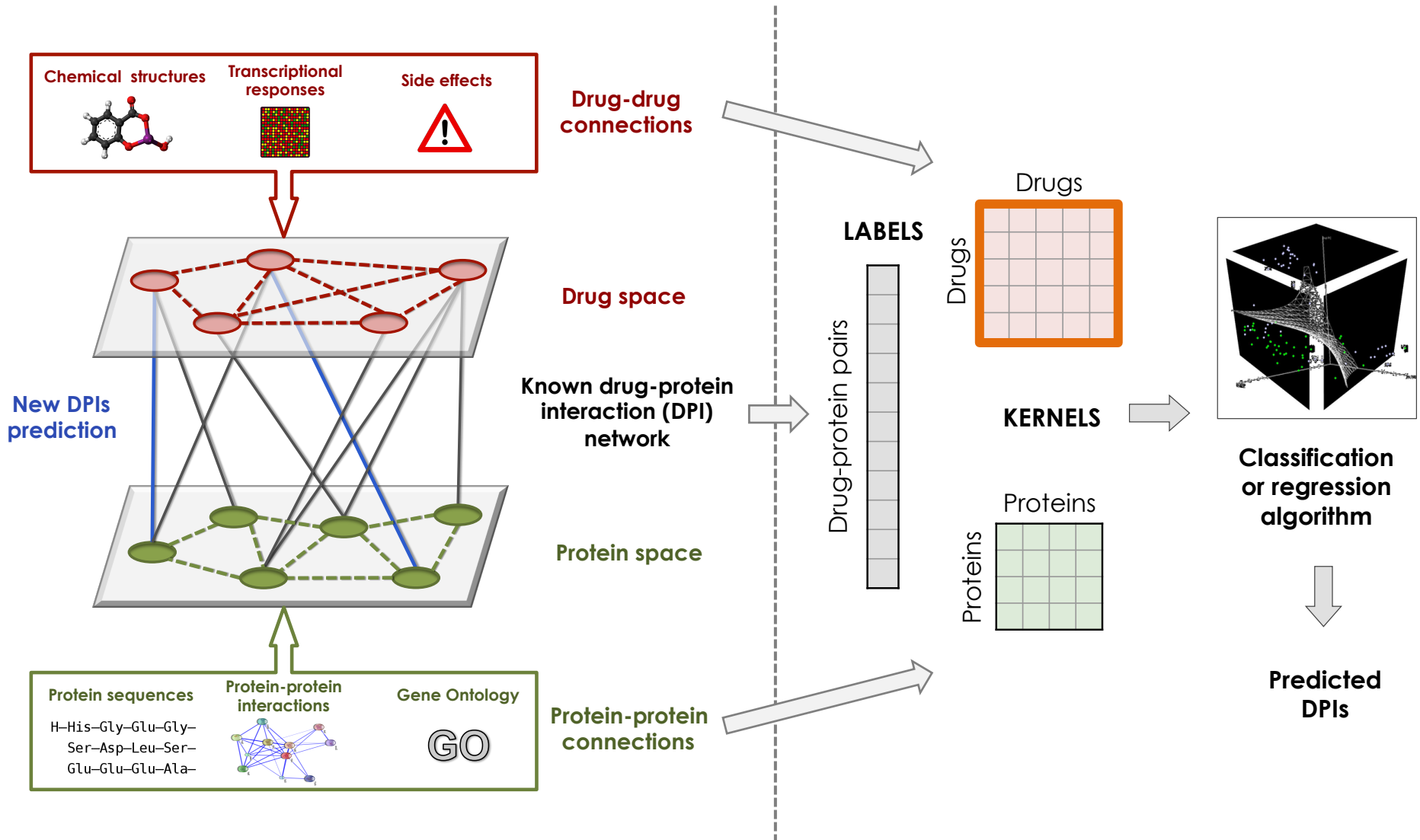
- Data generators deposit their data.
- Incorporates data from other databases, e.g. ChEMBL.
- Contains data on ~98 mln compounds.



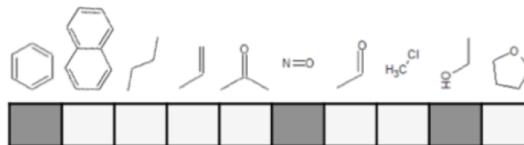
DRUGTARGET
COMMONS

<https://drugtargetcommons.fimm.fi/>

- **Crowd-sourcing platform** to improve the consensus and use of drug–target interactions.
- The end users can search, view and download bioactivity data using various compound, target and publications identifiers.
- Expert users may also submit suggestions to edit and upload new bioactivity data, as well as participate in the assay annotation and data curation processes.



Molecular fingerprint



*Chemical
space*

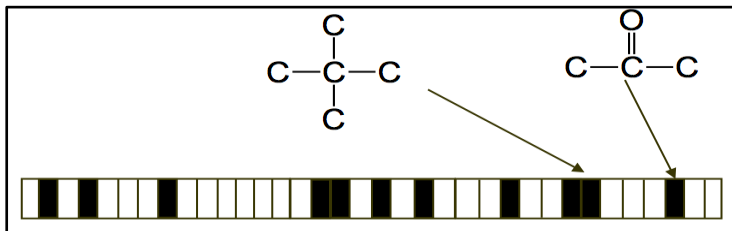
- A way of encoding the structure of a molecule.
- The most common type of fingerprint is a series of binary digits (bits) that represent the presence or absence of particular substructures in the molecule.

Example

1	1	1	0	1	1	0	1	0
2	1	1	0	1	0	0	0	0

2D fingerprints

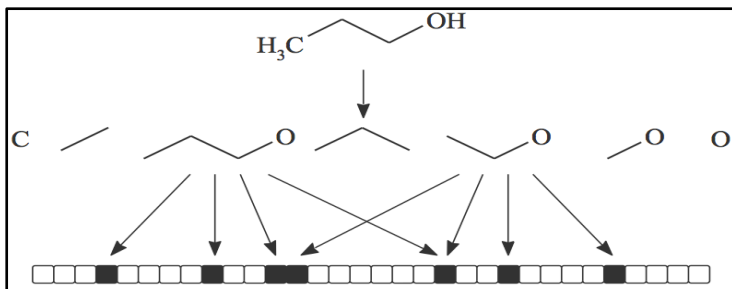
Chemical space



Dictionary-Based Fingerprints

Pre-defined fragments, each of which maps to a single bit.

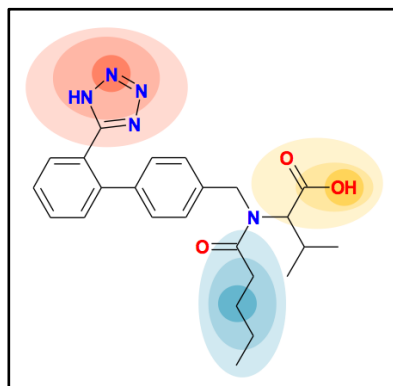
Examples: MACCS Keys, BCI.



Path-Based Hashed Fingerprints

Fragments are generated algorithmically without the need for a dictionary, e.g., all paths up to seven non-hydrogen atoms from the source atom.

Examples: Daylight, UNITY fingerprints.

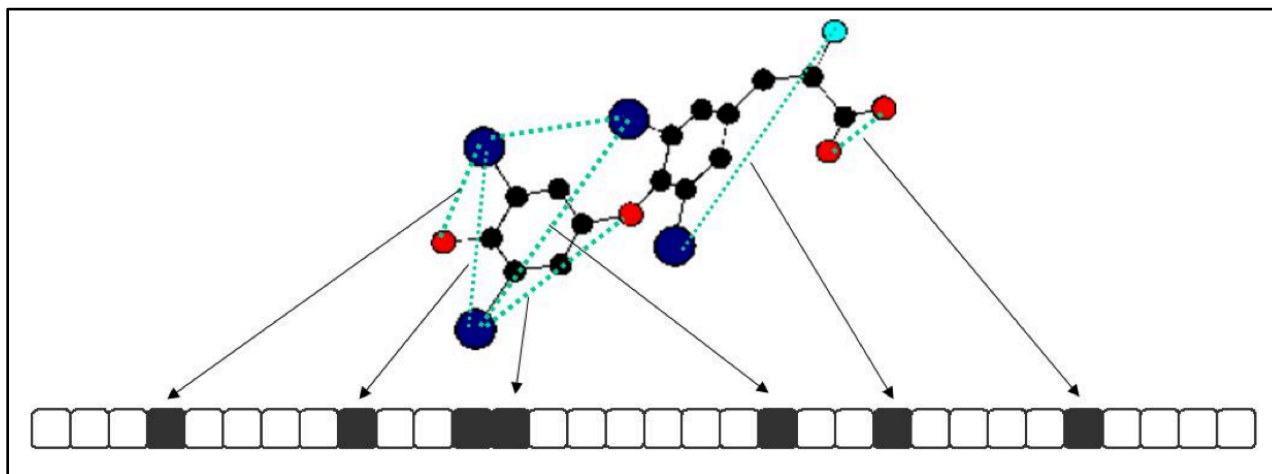


Circular Hashed Fingerprints

Each atom is represented together with its environment (neighbouring atoms as extended spheres).

Examples: ECFP2, ECFP4.

3D fingerprints



Presence or absence of geometric features,
e.g., pairs/triplets of atoms at given distance,
valence/torsion angles.

Fingerprint-based Tanimoto kernel

Chemical
space

$$K(fp_1, fp_2) = \frac{N_{fp_1, fp_2}}{N_{fp_1} + N_{fp_2} - N_{fp_1, fp_2}}$$

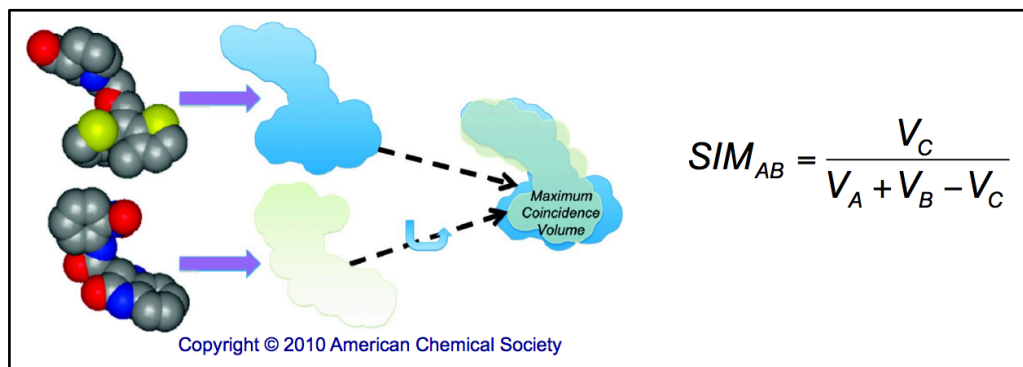
fp_i – fingerprint of the molecule i ,

N_{fp_i} – number of 1-bits in the fingerprint fp_i ,

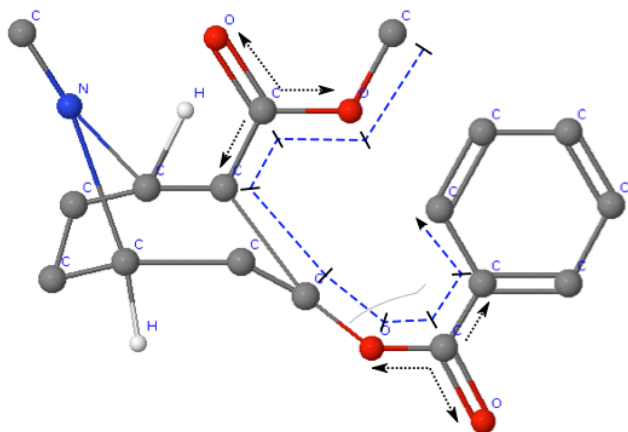
N_{fp_1, fp_2} – number of 1-bits in both fingerprints.

- Computed based on the size of common substructures of the molecules represented by the fingerprints.

3D shape-based comparison



- Atoms are represented as Gaussian functions.
- Molecules are aligned in 3D.
- Similarity score is based on the common volume.

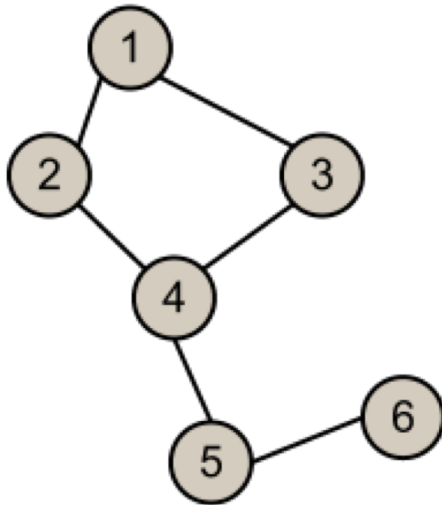


- Graph kernels allow to measure the similarity between graphs.
 - Chemical molecule can be represented as a labeled or unlabeled graph, where a node corresponds to an atom, and an edge indicates a bond between two atoms.
-
- Graph kernels can be roughly categorized into three main groups:
 - 1) graph kernels based on walks and paths,
 - 2) graph kernels based on limited-size subgraphs,
 - 3) graph kernels based on subtree patterns.
 - **Examples:** random walk kernel, shortest-path kernel, Weisfeiler-Lehman subtree kernel.

Graph

➤ A graph G is a set of nodes (vertices) V and edges E .

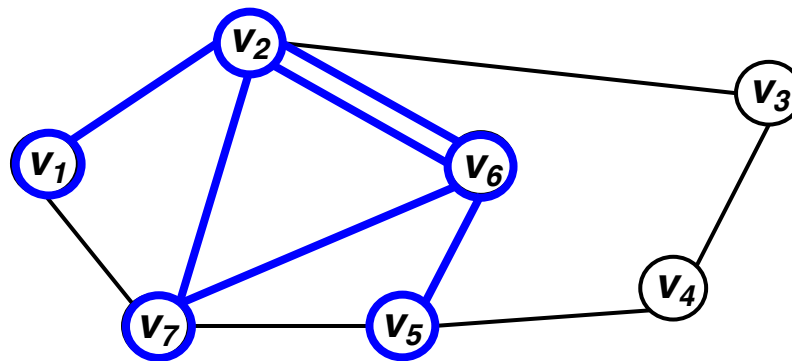
➤ The adjacency matrix \mathbf{A} of G is defined as $[A]_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$.



	①	②	③	④	⑤	⑥
①	0	1	1	0	0	0
②	1	0	0	1	0	0
③	1	0	0	1	0	0
④	0	1	1	0	1	0
⑤	0	0	0	1	0	1
⑥	0	0	0	0	1	0

Random walk

- A graph G is a set of nodes (vertices) V and edges E .
- The adjacency matrix \mathbf{A} of G is defined as $[A]_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$.
- **Walk** – a sequence of nodes, in which consecutive nodes are connected by an edge. A walk can travel over any edge and any node any number of times.



$$W = (v_1, v_2, v_6, v_7, v_2, v_6, v_5)$$

- Walks of length k can be computed by taking the adjacency matrix \mathbf{A} to the power of k . $A^k(v_i, v_j) = m \rightarrow m$ walks of length k exist between nodes v_i and v_j .

Random walk graph kernel

Chemical
space

- Random walk kernel computes the number of all pairs of matching walks in a pair of graphs.

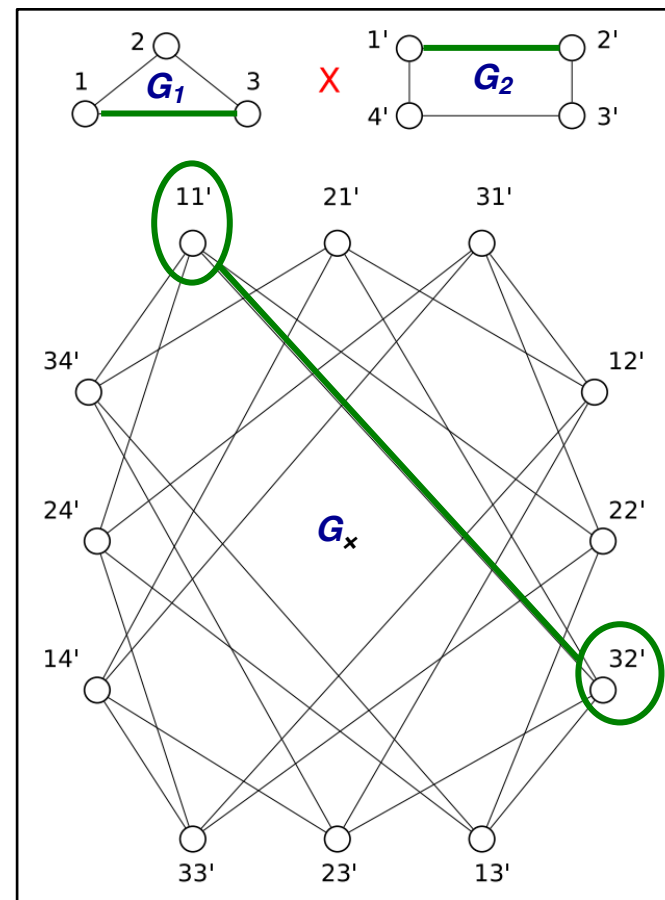
- **TRICK:** common walks of length k can be calculated from the adjacency matrix of the **product graph** G_x of two input graphs G_1 and G_2 .

- G_x is a graph over pairs of vertices from G_1 and G_2 . Two vertices in G_x are neighbours if and only if the corresponding vertices in G_1 and G_2 are both neighbours.

- **Random walk kernel**

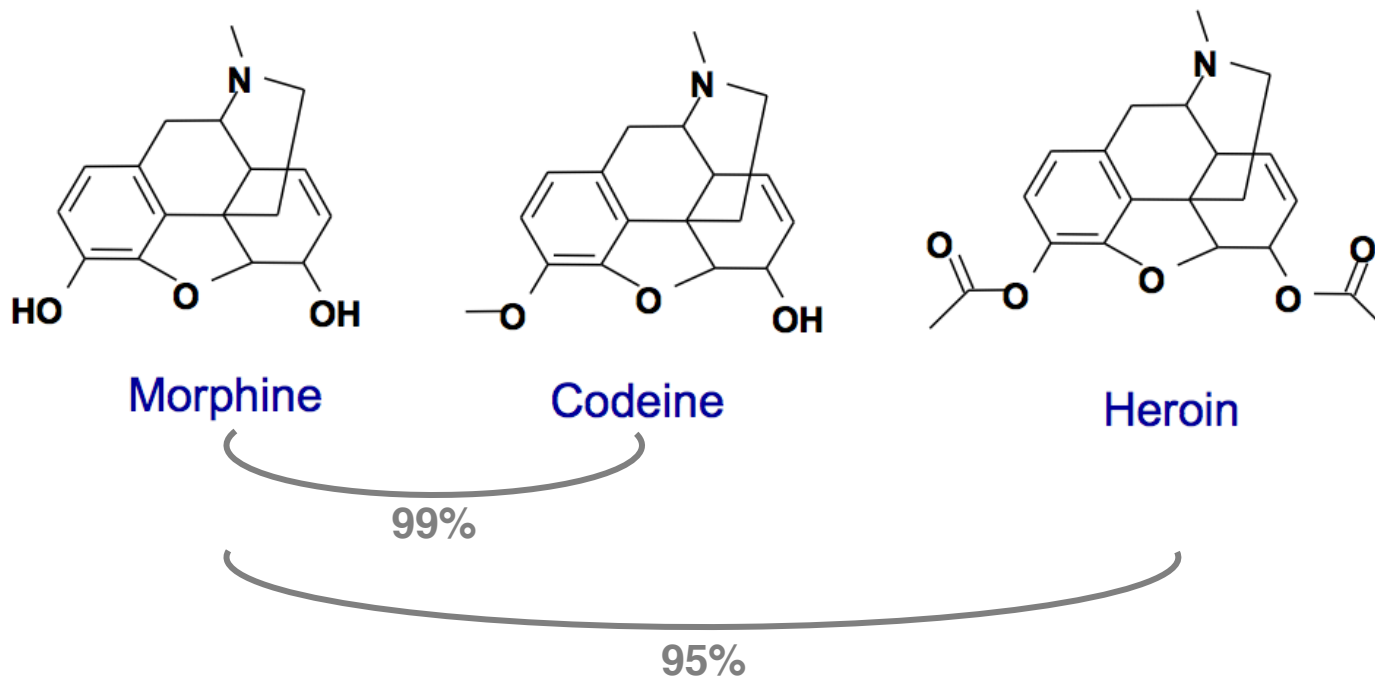
$$K_x(G_1, G_2) = \sum_{i,j=1}^{|V_x|} \left[\sum_{n=0}^{\infty} \lambda^n A_x^n \right]_{ij} = \sum_{i,j=1}^{|V_x|} [(I - \lambda A_x)^{-1}]_{ij}$$

Counts all pairs
of matching walks
of any length



Problem with using chemical structures

- Sometimes structurally similar molecules can have different properties.



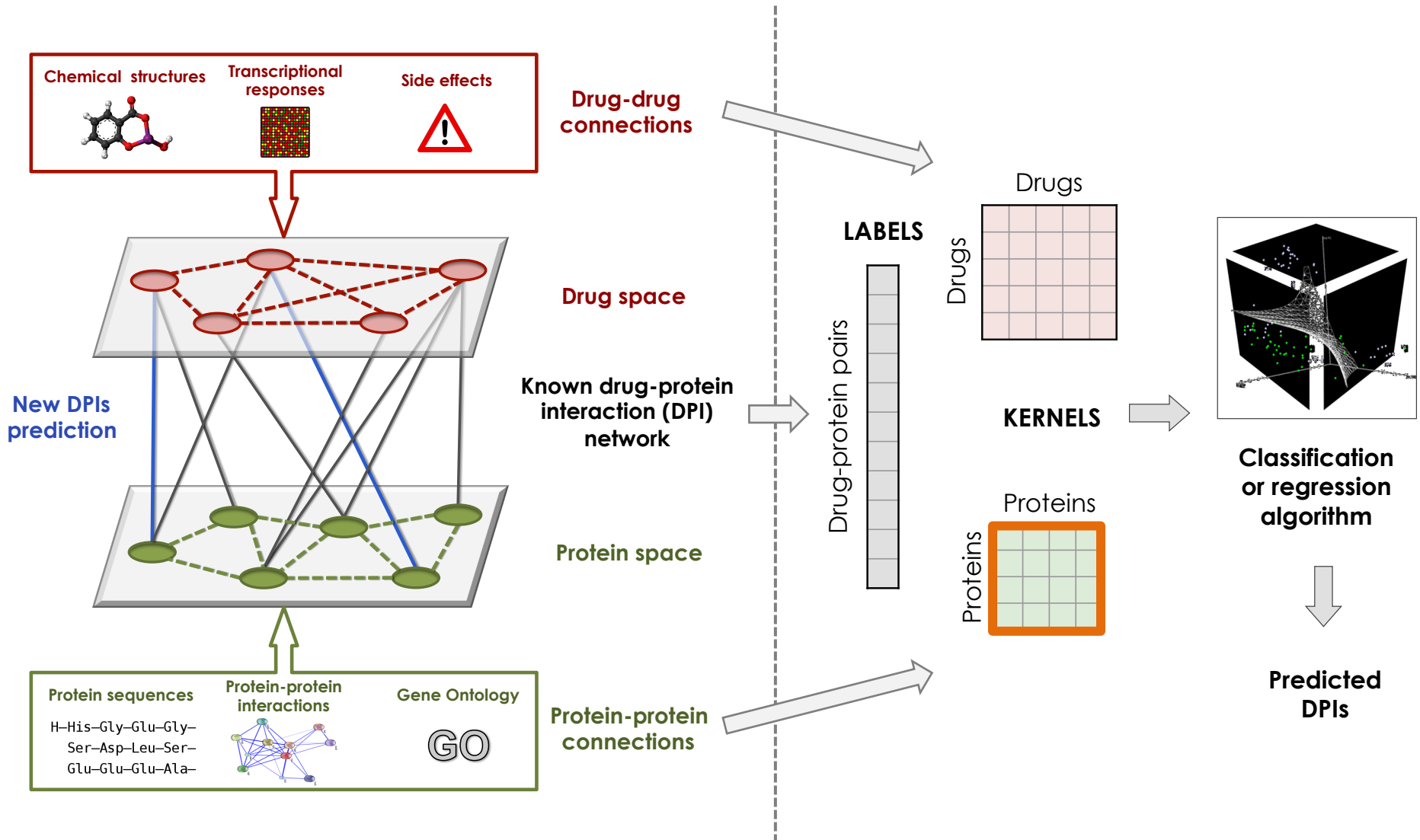
2D Tanimoto similarity

- Side effects.
- Anatomical Therapeutic Chemical (ATC) Classification System.
- Gene expression responses to drugs.
- Binding affinity profile.

ATC

Example: **Glibenclamide (A10B B01)**

A	Alimentary tract and metabolism (main anatomical group)
A10	Drugs used in diabetes (main therapeutic group)
A10B	Oral blood-glucose-lowering drugs (pharmacological subgroup)
A10B B	Sulfonamides, urea derivatives (chemical/therapeutic subgroup)
A10B B01	Glibenclamide (subgroup for chemical substance)



Amino acid sequence alignment

- **Protein sequence alignment** is a way of arranging the amino acid sequences to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences.

Cox1	PGLLLEKCHPN SIFGESMIEM-GAPFSLKGLLGNPICSPEYWKASTFGGGEVGFNLVKTAT
Cox2	PALLVEKPRPDAIFGETMVEL-GAPFSLKGLMGNPICSPQYWKPSTFGGGEVGFKIINTAS
Mpx	MGGVSEPLKRKGRVGPLLACIIIGTFRKLRDGD-----RFW----WENEGVFSMQORQA

- **Smith-Waterman (SW) algorithm**

- Performs local sequence alignment; uses dynamic programming to compare segments of all possible lengths.
- To find the optimal alignment, a scoring system including a set of specified gap penalties is used (different scoring matrices, e.g. BLOSUM, PAM).
- The algorithm assigns a score to each residue comparison between two sequences.
- Normalized similarity between two proteins p_1 and p_2 :

$$s(p_1, p_2) = \frac{SW(p_1, p_2)}{\sqrt{SW(p_1, p_1)} \sqrt{SW(p_2, p_2)}}$$

Generic String (GS) kernel

Protein
space

$$GS(\mathbf{x}, \mathbf{x}', L, \sigma_p, \sigma_c)$$

$$\stackrel{\text{def}}{=} \sum_{l=1}^L \sum_{i=0}^{|\mathbf{x}|-l} \sum_{j=0}^{|\mathbf{x}'|-l} e^{\left(\frac{-(i-j)^2}{2\sigma_p^2}\right)} e^{\left(\frac{-\|\psi^l(x_{i+1}, \dots, x_{i+l}) - \psi^l(x'_{j+1}, \dots, x'_{j+l})\|^2}{2\sigma_c^2}\right)}$$

Generic String (GS) kernel

Protein
space

$$GS(\mathbf{x}, \mathbf{x}', L, \sigma_p, \sigma_c)$$

$$\stackrel{\text{def}}{=} \sum_{l=1}^L \sum_{i=0}^{|\mathbf{x}|-l} \sum_{j=0}^{|\mathbf{x}'|-l} e^{\left(\frac{-(i-j)^2}{2\sigma_p^2}\right)} e^{\left(\frac{-\|\psi^l(x_{i+1}, \dots, x_{i+l}) - \psi^l(x'_{j+1}, \dots, x'_{j+l})\|^2}{2\sigma_c^2}\right)}$$

**Shifting
contribution
term**

Generic String (GS) kernel

Protein
space

$$GS(\mathbf{x}, \mathbf{x}', L, \sigma_p, \sigma_c)$$

$$\stackrel{\text{def}}{=} \sum_{l=1}^L \sum_{i=0}^{|\mathbf{x}|-l} \sum_{j=0}^{|\mathbf{x}'|-l} e^{\left(\frac{-(i-j)^2}{2\sigma_p^2}\right)} e^{\left(\frac{-\|\psi^l(x_{i+1}, \dots, x_{i+l}) - \psi^l(x'_{j+1}, \dots, x'_{j+l})\|^2}{2\sigma_c^2}\right)}$$

Shifting contribution term

Similarity of the amino acids in the substrings \mathbf{x} and \mathbf{x}'

Generic String (GS) kernel

$$GS(\mathbf{x}, \mathbf{x}', L, \sigma_p, \sigma_c)$$

$$\stackrel{\text{def}}{=} \sum_{l=1}^L \sum_{i=0}^{|\mathbf{x}|-l} \sum_{j=0}^{|\mathbf{x}'|-l} e^{\left(\frac{-(i-j)^2}{2\sigma_p^2}\right)} e^{\left(\frac{-\|\psi^l(x_{i+1}, \dots, x_{i+l}) - \psi^l(x'_{j+1}, \dots, x'_{j+l})\|^2}{2\sigma_c^2}\right)}$$

Shifting contribution term
Similarity of the amino acids in the substrings \mathbf{x} and \mathbf{x}'

Each type of amino acid a_k , $k = 1, \dots, K$, (e.g. Asparagine) has a corresponding feature vector $\psi(a_k)$ which defines its d properties:

$$\psi(a_k) = (\psi_1(a_k), \psi_2(a_k), \dots, \psi_d(a_k)).$$

$\psi(A)$...	$\psi(M)$		
			...			Hydrophobicity
⋮	⋮	⋮	⋮	⋮	⋮	⋮
			...			Volume
			...			Polarity

Protein sequence **A L A ... M**
} **X**

Generic String (GS) kernel

$$GS(\mathbf{x}, \mathbf{x}', L, \sigma_p, \sigma_c)$$

$$\stackrel{\text{def}}{=} \sum_{l=1}^L \sum_{i=0}^{|\mathbf{x}|-l} \sum_{j=0}^{|\mathbf{x}'|-l} e^{\left(\frac{-(i-j)^2}{2\sigma_p^2}\right)} e^{\left(\frac{-\|\psi^l(x_{i+1}, \dots, x_{i+l}) - \psi^l(x'_{j+1}, \dots, x'_{j+l})\|^2}{2\sigma_c^2}\right)}$$

Shifting contribution term
Similarity of the amino acids in the substrings \mathbf{x} and \mathbf{x}'

Each type of amino acid a_k , $k = 1, \dots, K$, (e.g. Asparagine) has a corresponding feature vector $\psi(a_k)$ which defines its d properties:

$$\psi(a_k) = (\psi_1(a_k), \psi_2(a_k), \dots, \psi_d(a_k)).$$

Given a string $\mathbf{x} = x_1, x_2, \dots, x_l$, $\psi^l(\mathbf{x})$ is its encoding function which concatenates l vectors describing each amino acid the string \mathbf{x} is composed of:

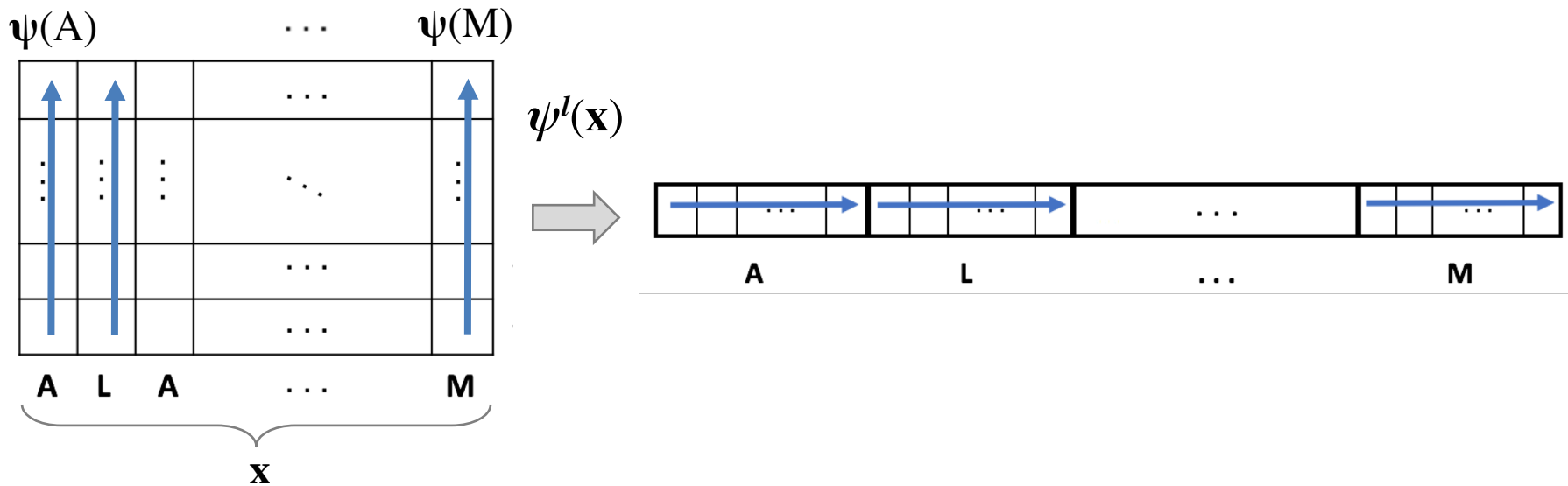
$$\psi^l(\mathbf{x}) = (\psi(x_1), \psi(x_2), \dots, \psi(x_l)).$$

Generic String (GS) kernel

$$GS(\mathbf{x}, \mathbf{x}', L, \sigma_p, \sigma_c)$$

$$\stackrel{\text{def}}{=} \sum_{l=1}^L \sum_{i=0}^{|\mathbf{x}|-l} \sum_{j=0}^{|\mathbf{x}'|-l} e^{\left(\frac{-(i-j)^2}{2\sigma_p^2}\right)} e^{\left(\frac{-\|\psi^l(x_{i+1}, \dots, x_{i+l}) - \psi^l(x'_{j+1}, \dots, x'_{j+l})\|^2}{2\sigma_c^2}\right)}$$

Shifting contribution term
Similarity of the amino acids in the substrings \mathbf{x} and \mathbf{x}'



Additional information

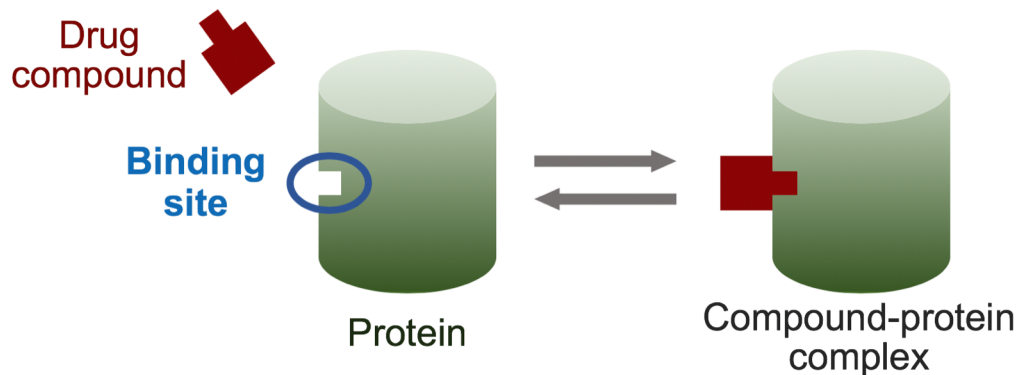
Protein space

➤ Protein binding site.

➤ Protein surface.

➤ Gene Ontology classification.

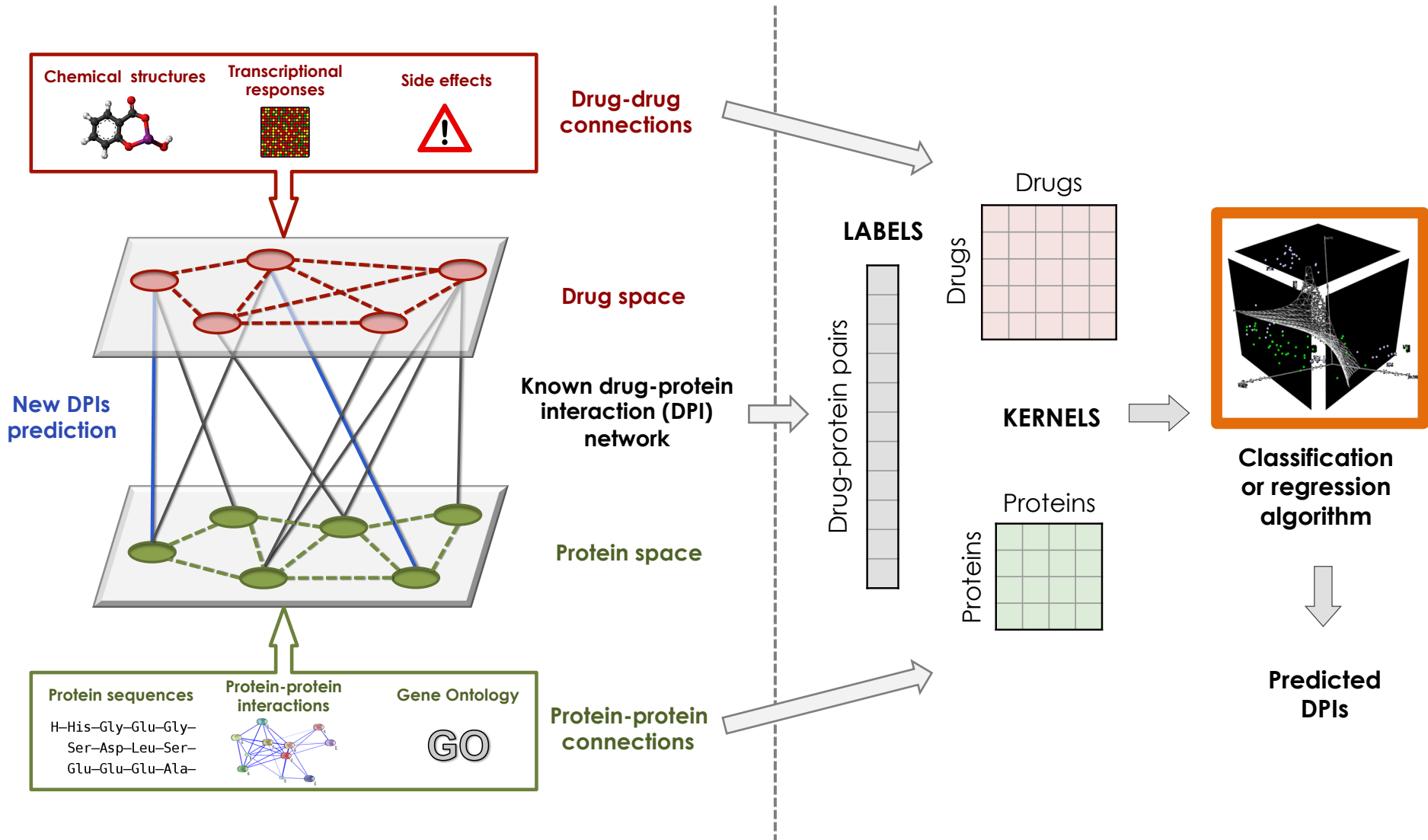
<http://geneontology.org/>



GO

Three domains:

- 1) biological processes,
- 2) cellular components,
- 3) molecular functions.



Supervised learning

*Inferring a function from
LABLED training data*

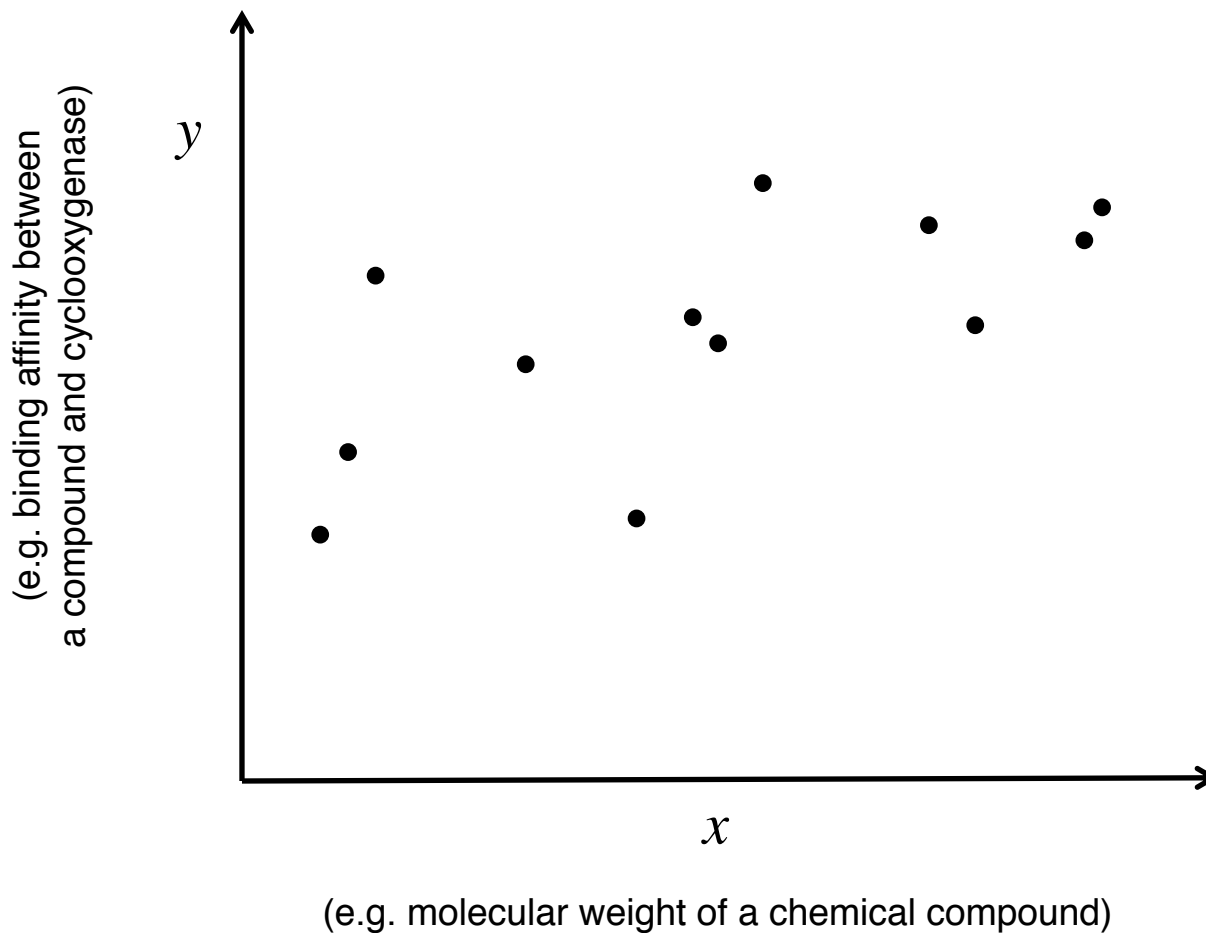
- **Classification** is the prediction of a **class label**, given attributes.
- **Regression** is the prediction of a **real number**, given attributes.

INGREDIENTS

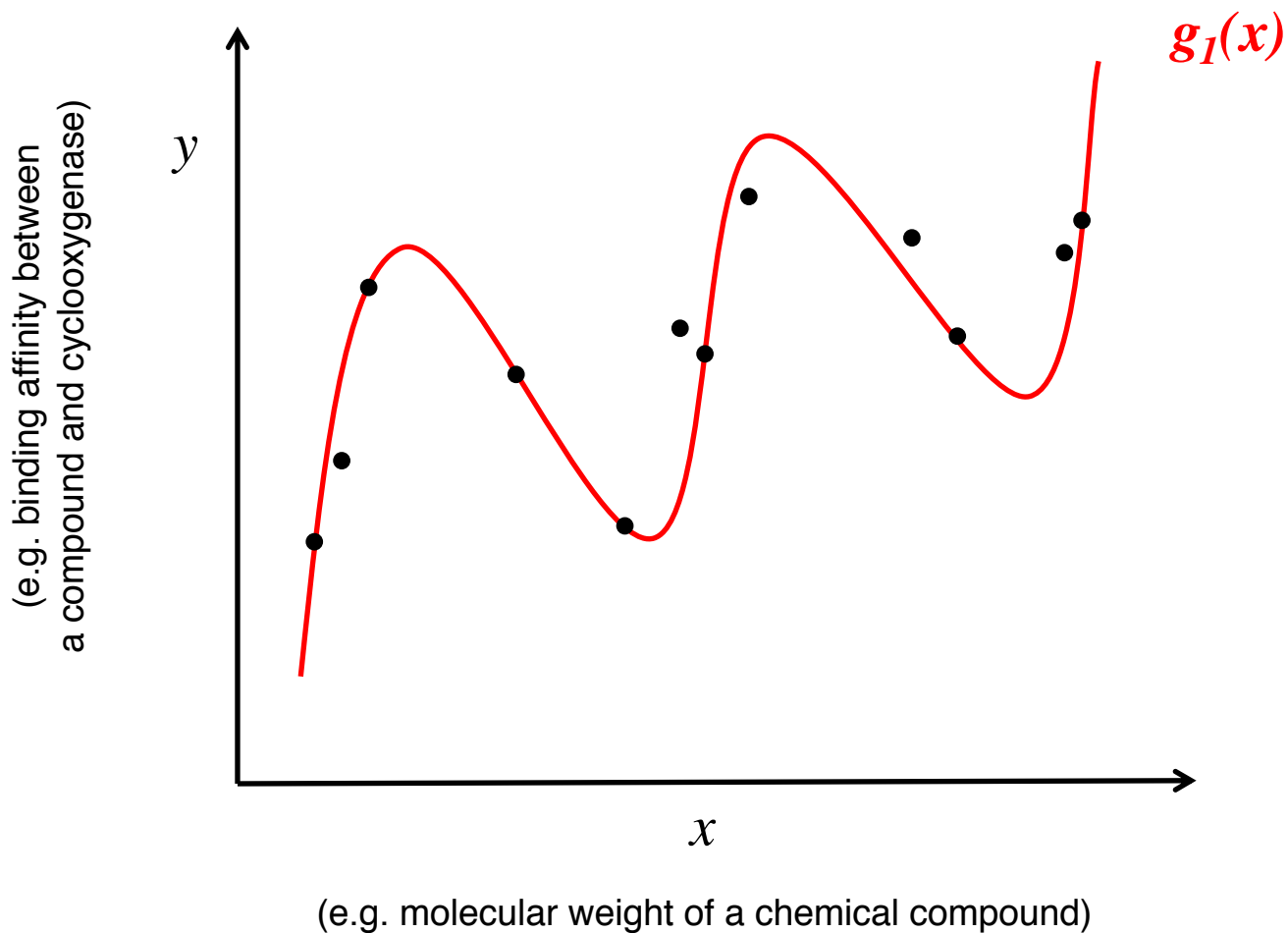
- \mathcal{X} : a space of inputs.
- \mathcal{Y} : a space of outputs.
- \mathcal{G} : a set of models mapping input to output $\mathcal{G} = \{g: \mathcal{X} \mapsto \mathcal{Y}\}$.
- Training dataset $S: \{(\mathbf{x}_i, y_i)\}$, $i=1, \dots, N$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$
sampled from an underlying unknown distribution $(\mathbf{x}, y) \sim \mathcal{D}$.
- \mathcal{L} : a loss function measuring the discrepancy between the model's predicted outputs and true outputs.

GOAL: to find a model g that minimizes the expected loss $\mathcal{L}(g(\mathbf{x}), y)$ on future instances.

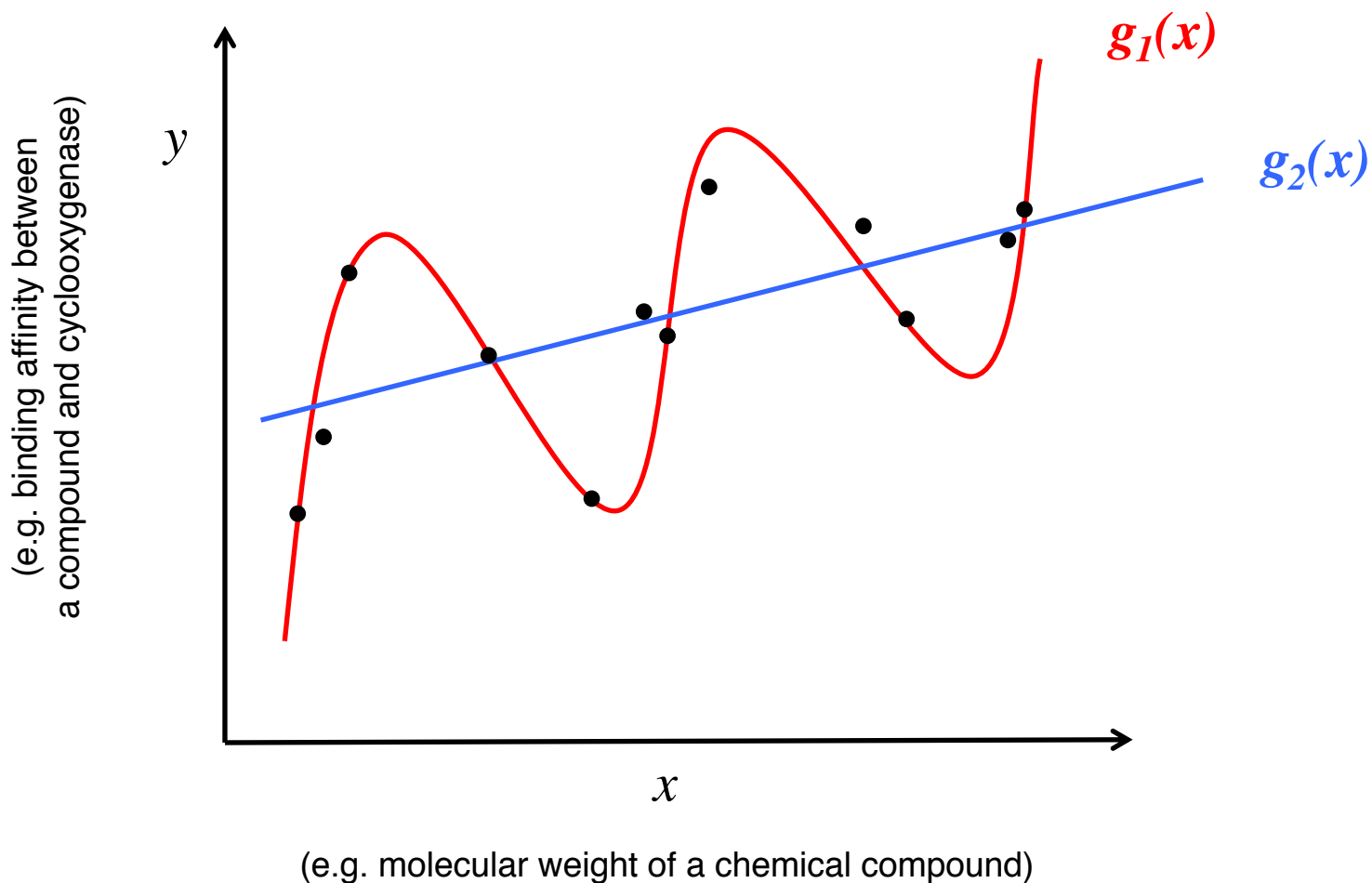
Overfitting



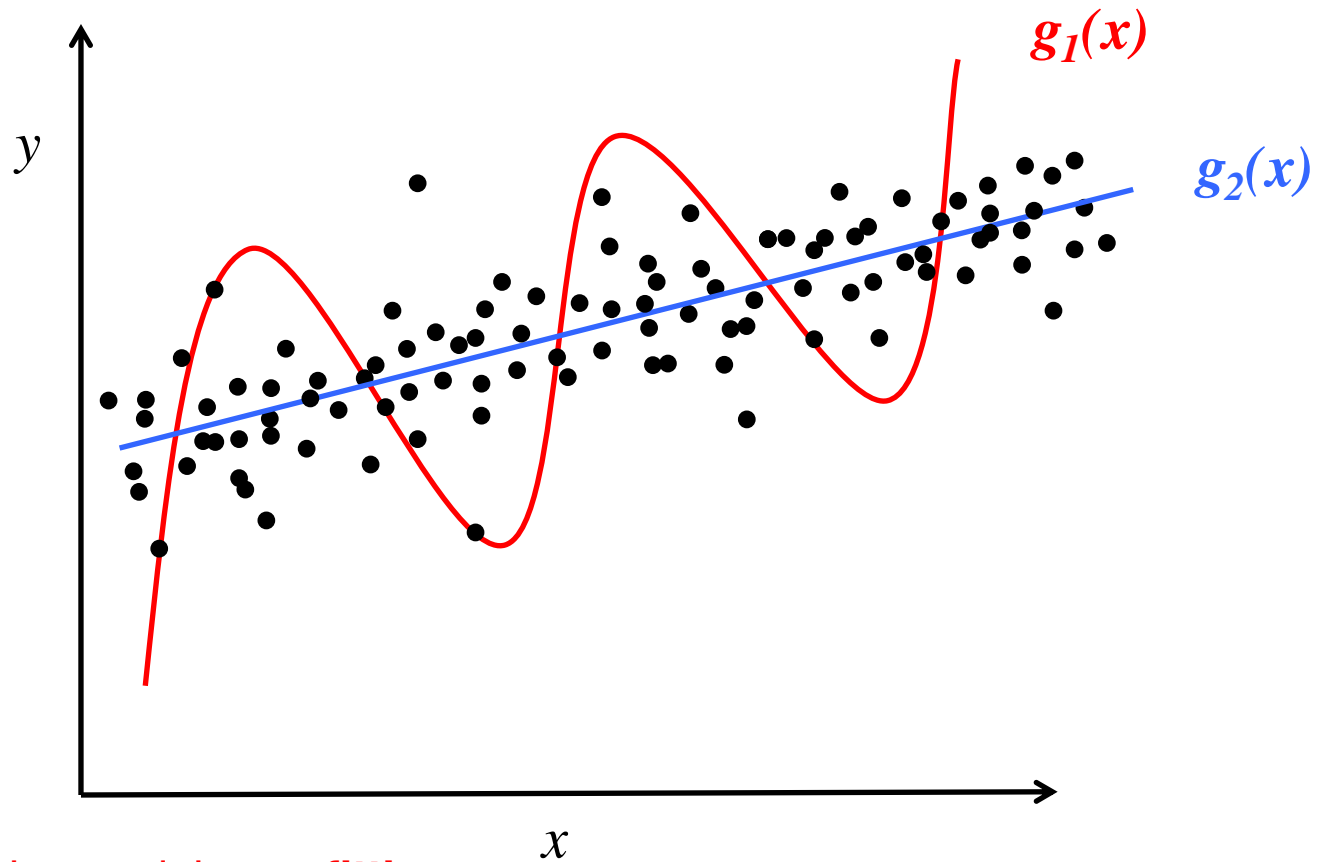
Overfitting



Overfitting



Overfitting



$g_1(x)$: more complex model, **overfitting**.

$g_2(x)$: generalizes better to new instances.

Regularization

- Regularized learning considers optimising the functions of the form:

$$\arg \min_{g \in G} \sum_{i=1}^N \mathcal{L}(g(\mathbf{x}_i), y_i) + \lambda \Omega(g)$$

Regularized empirical risk

Training error (loss),
typically, squared loss:

$$\mathcal{L}(g(\mathbf{x}_i), y_i) = (g(\mathbf{x}_i) - y_i)^2.$$

Regularizer that controls
the complexity of the model g .

- Complex model $g \rightarrow$ high value of $\Omega(g)$.
- Regularization parameter $\lambda \geq 0$ controls the balance between training error and model complexity.

Linear model

- A model in the form of a linear function $g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T \mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^p$ is the vector of model parameters to be found by minimizing $\sum_{i=1}^N \mathcal{L}(g(\mathbf{x}_i), y_i) + \lambda \Omega(g)$.
- The choice of the loss function and regularization determines the learning algorithm.

Linear model

- A model in the form of a linear function $g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^T \mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^p$ is the vector of model parameters to be found by minimizing $\sum_{i=1}^N \mathcal{L}(g(\mathbf{x}_i), y_i) + \lambda \Omega(g)$.
- The choice of the loss function and regularization determines the learning algorithm.

$\mathcal{L}(g(\mathbf{x}_i), y_i)$	$\Omega(g)$	Algorithm
$\max(0, 1 - g(\mathbf{x}_i)y_i)$, where $\mathbf{y} \in \{-1, +1\}$	$\ \mathbf{w}\ ^2 = \langle \mathbf{w}, \mathbf{w} \rangle$	Support vector machine (SVM)
$(g(\mathbf{x}_i) - y_i)^2$, where $\mathbf{y} \in \mathbb{R}^N$	$\ \mathbf{w}\ ^2 = \langle \mathbf{w}, \mathbf{w} \rangle$	Ridge regression
$(g(\mathbf{x}_i) - y_i)^2$, where $\mathbf{y} \in \mathbb{R}^N$	$\ \mathbf{w}\ _1 = \sum_{l=1}^p w_l $	Least absolute shrinkage and selection operator regression (LASSO)

Ridge regression

- Given the squared loss and quadratic regularizer, the optimization problem of ridge regression can be written as:

$$\arg \min_{\mathbf{w}} \sum_{i=1}^N (y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle)^2 + \lambda \|\mathbf{w}\|^2$$

$$\arg \min_{\mathbf{w}} \langle \mathbf{y} - \mathbf{X}\mathbf{w}, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle + \lambda \langle \mathbf{w}, \mathbf{w} \rangle, \quad \mathbf{y} \in \mathbb{R}^N, \mathbf{X} \in \mathbb{R}^{N \times p}$$

$$\frac{\delta}{\delta \mathbf{w}} (\langle \mathbf{y} - \mathbf{X}\mathbf{w}, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle + \lambda \langle \mathbf{w}, \mathbf{w} \rangle) = \mathbf{0}$$

$$\mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} - \mathbf{X}^T \mathbf{y} = \mathbf{0}$$

$$\mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_p)^{-1} \mathbf{X}^T \mathbf{y}$$

\mathbf{I}_p is a $p \times p$ identity matrix

Linear model and dual representation

- The optimal \mathbf{w} can be written as a linear combination of training examples by introducing so called *dual variable* $\alpha \in \mathbb{R}^N$:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i = \mathbf{X}^T \boldsymbol{\alpha}.$$

Linear model and dual representation

- The optimal \mathbf{w} can be written as a linear combination of training examples by introducing so called *dual variable* $\boldsymbol{\alpha} \in \mathbb{R}^N$:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i = \mathbf{X}^T \boldsymbol{\alpha}.$$

- Now, we can represent model's prediction in terms of inner products of training examples

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^N \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle$$

Linear model and dual representation

- The optimal \mathbf{w} can be written as a linear combination of training examples by introducing so called *dual variable* $\boldsymbol{\alpha} \in \mathbb{R}^N$:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i = \mathbf{X}^T \boldsymbol{\alpha}.$$

- Now, we can represent model's prediction in terms of inner products of training examples \rightarrow we can use **kernels**:

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^N \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle = \sum_{i=1}^N \alpha_i k(\mathbf{x}_i, \mathbf{x}) = \boldsymbol{\alpha}^T \mathbf{k},$$

where \mathbf{k} is a vector with kernel values between each training example \mathbf{x}_i and a test example \mathbf{x} for which the prediction is made.

Ridge regression

- Given the squared loss and quadratic regularizer, the optimization problem of ridge regression can be written as:

$$\arg \min_{\mathbf{w}} \sum_{i=1}^N (y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle)^2 + \lambda \|\mathbf{w}\|^2$$

$$\arg \min_{\mathbf{w}} \langle \mathbf{y} - \mathbf{X}\mathbf{w}, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle + \lambda \langle \mathbf{w}, \mathbf{w} \rangle, \quad \mathbf{y} \in \mathbb{R}^N, \mathbf{X} \in \mathbb{R}^{N \times p}$$

$$\frac{\delta}{\delta \mathbf{w}} (\langle \mathbf{y} - \mathbf{X}\mathbf{w}, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle + \lambda \langle \mathbf{w}, \mathbf{w} \rangle) = 0$$

$$\mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} - \mathbf{X}^T \mathbf{y} = 0$$

$$\mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_p)^{-1} \mathbf{X}^T \mathbf{y}$$

\mathbf{I}_p is a $p \times p$ identity matrix

Kernel ridge regression (KRR)

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i = \mathbf{X}^T \boldsymbol{\alpha}.$$

➤ We can rewrite equation $\mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} = \mathbf{X}^T \mathbf{y}$ in terms of \mathbf{w} to get:

$$\mathbf{w} = \lambda^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{X} \mathbf{w}) = \mathbf{X}^T \boldsymbol{\alpha}.$$

Kernel ridge regression (KRR)

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i = \mathbf{X}^T \boldsymbol{\alpha}.$$

- We can rewrite equation $\mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} = \mathbf{X}^T \mathbf{y}$ in terms of \mathbf{w} to get:

$$\mathbf{w} = \lambda^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{X} \mathbf{w}) = \mathbf{X}^T \boldsymbol{\alpha}.$$

- The solution to ridge regression in the dual space (i.e. KRR) has a closed form

$$\boldsymbol{\alpha} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

Kernel ridge regression (KRR)

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \mathbf{x}_i = \mathbf{X}^T \boldsymbol{\alpha}.$$

- We can rewrite equation $\mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} = \mathbf{X}^T \mathbf{y}$ in terms of \mathbf{w} to get:

$$\mathbf{w} = \lambda^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{X} \mathbf{w}) = \mathbf{X}^T \boldsymbol{\alpha}.$$

- The solution to ridge regression in the dual space (i.e. KRR) has a closed form

$$\boldsymbol{\alpha} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

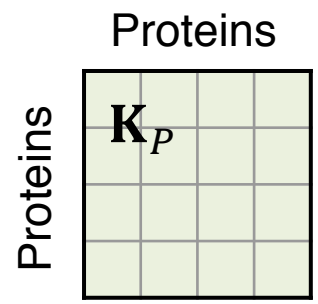
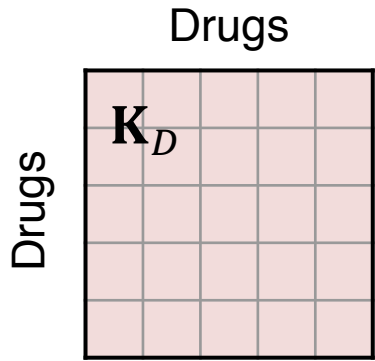
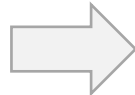
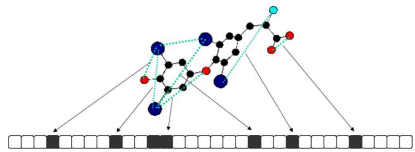
$$\mathbf{K} \in \mathbb{R}^{N \times N}$$

KRR for drug-protein binding affinity prediction

Ingredients

- A set of n_d drugs: $D = \{\mathbf{d}_1, \dots, \mathbf{d}_{n_d}\}$
- A set of n_p proteins: $P = \{\mathbf{p}_1, \dots, \mathbf{p}_{n_p}\}$
- A set of N training examples (drug-protein pairs): $X = \{(\mathbf{d}_1, \mathbf{p}_1), \dots, (\mathbf{d}_1, \mathbf{p}_{n_p}), (\mathbf{d}_2, \mathbf{p}_1), \dots, (\mathbf{d}_2, \mathbf{p}_{n_p}), \dots, \dots, (\mathbf{d}_{n_d}, \mathbf{p}_{n_p})\}$
- $N \leq n_d \times n_p$
- y_i : a real value indicating binding affinity of i^{th} drug-protein pair \mathbf{x}_i
- Pairwise kernel matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$

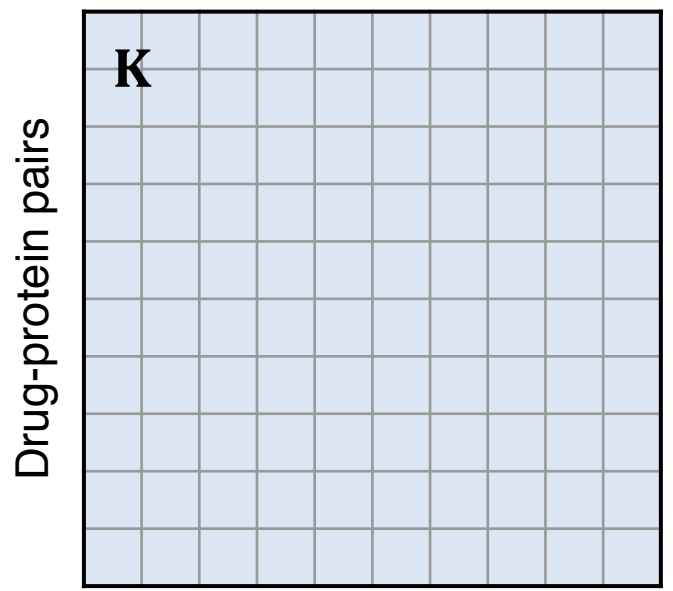
Pairwise kernel



PAIRWISE KERNEL

$$K = K_D \otimes K_P$$

Drug-protein pairs



Kronecker product

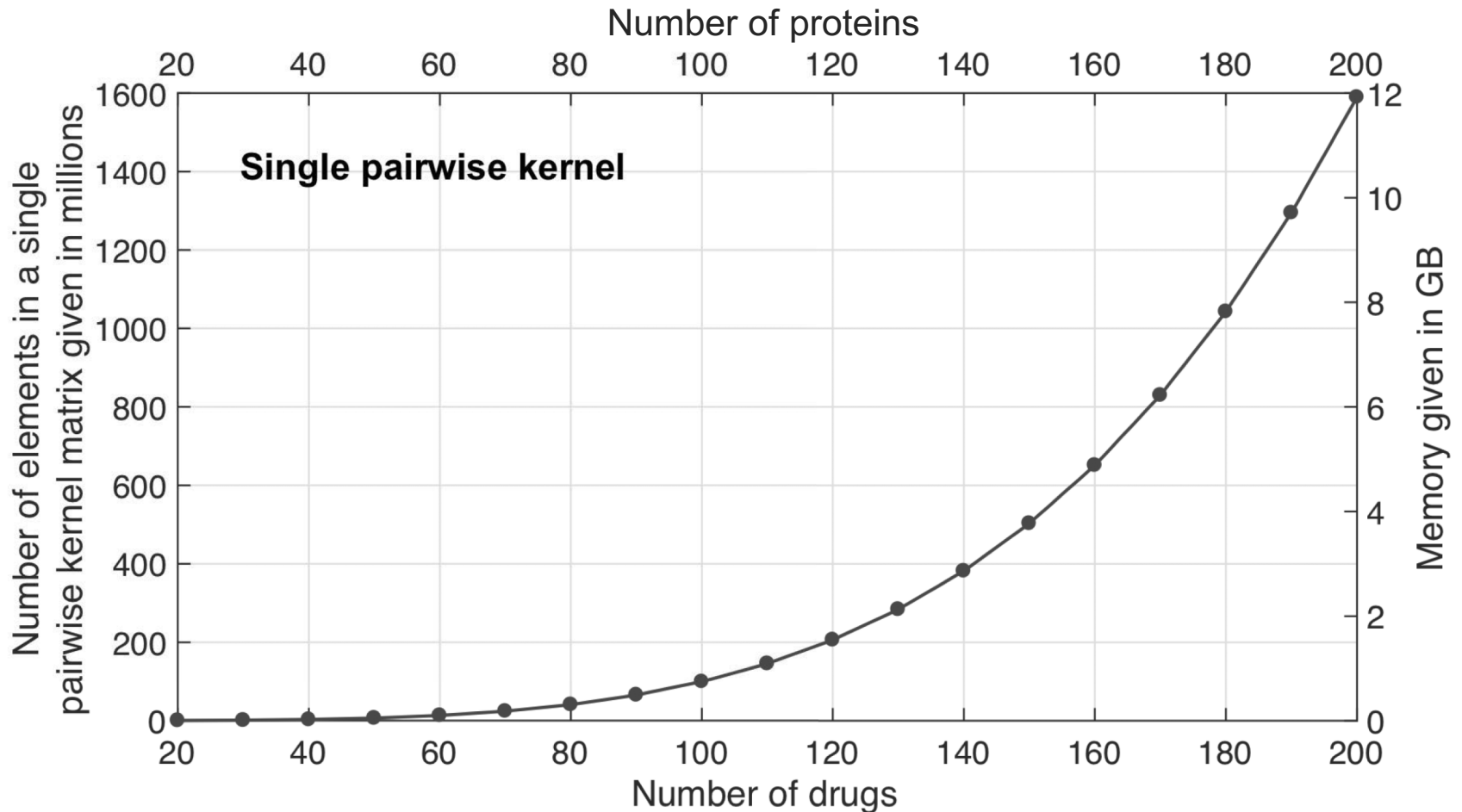
- Defined for any two matrices **B** and **C** of arbitrary size.
- Resulting matrix contains all possible products of entries of **B** and **C**.

Example

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \otimes \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \left[\begin{array}{ccc|ccc} b_{11}c_{11} & b_{11}c_{12} & b_{11}c_{13} & b_{12}c_{11} & b_{12}c_{12} & b_{12}c_{13} \\ b_{11}c_{21} & b_{11}c_{22} & b_{11}c_{23} & b_{12}c_{21} & b_{12}c_{22} & b_{12}c_{23} \\ b_{11}c_{31} & b_{11}c_{32} & b_{11}c_{33} & b_{12}c_{31} & b_{12}c_{32} & b_{12}c_{33} \\ \hline b_{21}c_{11} & b_{21}c_{12} & b_{21}c_{13} & b_{22}c_{11} & b_{22}c_{12} & b_{22}c_{13} \\ b_{21}c_{21} & b_{21}c_{22} & b_{21}c_{23} & b_{22}c_{21} & b_{22}c_{22} & b_{22}c_{23} \\ b_{21}c_{31} & b_{21}c_{32} & b_{21}c_{33} & b_{22}c_{31} & b_{22}c_{32} & b_{22}c_{33} \end{array} \right]$$

Pairwise kernel

- The size of a pairwise kernel matrix \mathbf{K} makes the model training computationally infeasible in typical applications.



Pairwise KRR – shortcut

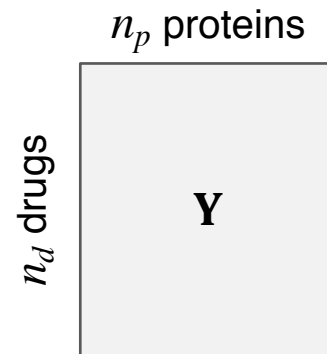
- It is possible to use algebraic properties of the Kronecker product to avoid the explicit computation of the pairwise kernel, and therefore significantly speed up the model training.

$$\begin{aligned}\alpha &= (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y} \\ &= (\mathbf{K}_D \otimes \mathbf{K}_P + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\ &= ((\mathbf{Q}_D \Lambda_D \mathbf{Q}_D^T) \otimes (\mathbf{Q}_P \Lambda_P \mathbf{Q}_P^T) + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\ &= \text{vec}(\mathbf{Q}_P \mathbf{R} \mathbf{Q}_D^T)\end{aligned}$$

$\text{vec}(\cdot)$ vectorization operator that arranges the columns of a matrix into a vector

Eigen-decomposition of the kernel matrices \mathbf{K}_D and \mathbf{K}_P

$$\text{vec}(\mathbf{R}) = (\Lambda_D \otimes \Lambda_P + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Q}_P^T \mathbf{Y}^T \mathbf{Q}_D)$$



Pahikkala T *et al.* (2013) “Efficient regularized least-squares algorithms for conditional ranking on relational data”. Machine Learning.

Pairwise KRR – shortcut

- It is possible to use algebraic properties of the Kronecker product to avoid the explicit computation of the pairwise kernel, and therefore significantly speed up the model training.

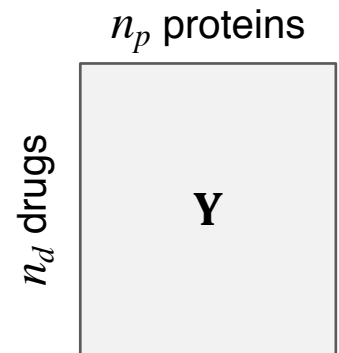
$$\begin{aligned}
 \boldsymbol{\alpha} &= (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y} \\
 &= (\mathbf{K}_D \otimes \mathbf{K}_P + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\
 &= ((\mathbf{Q}_D \boldsymbol{\Lambda}_D \mathbf{Q}_D^T) \otimes (\mathbf{Q}_P \boldsymbol{\Lambda}_P \mathbf{Q}_P^T) + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\
 &= \text{vec}(\mathbf{Q}_P \mathbf{R} \mathbf{Q}_D^T)
 \end{aligned}$$

$\text{vec}(\cdot)$ vectorization operator that arranges the columns of a matrix into a vector

Eigen-decomposition of the kernel matrices \mathbf{K}_D and \mathbf{K}_P

$$\text{vec}(\mathbf{R}) = \underbrace{(\boldsymbol{\Lambda}_D \otimes \boldsymbol{\Lambda}_P + \lambda \mathbf{I}_N)^{-1}} \text{vec}(\mathbf{Q}_P^T \mathbf{Y}^T \mathbf{Q}_D)$$

$$\text{diag}(\boldsymbol{\Lambda}_D \otimes \boldsymbol{\Lambda}_P) = \text{diag}(\boldsymbol{\Lambda}_D) \otimes \text{diag}(\boldsymbol{\Lambda}_P) = \text{vec}(\text{diag}(\boldsymbol{\Lambda}_P) \text{diag}(\boldsymbol{\Lambda}_D)^T)$$



Pairwise KRR – shortcut

- It is possible to use algebraic properties of the Kronecker product to avoid the explicit computation of the pairwise kernel, and therefore significantly speed up the model training.

$$\begin{aligned}
 \boldsymbol{\alpha} &= (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y} \\
 &= (\mathbf{K}_D \otimes \mathbf{K}_P + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\
 &= ((\mathbf{Q}_D \boldsymbol{\Lambda}_D \mathbf{Q}_D^T) \otimes (\mathbf{Q}_P \boldsymbol{\Lambda}_P \mathbf{Q}_P^T) + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\
 &= \text{vec}(\mathbf{Q}_P \mathbf{R} \mathbf{Q}_D^T)
 \end{aligned}$$

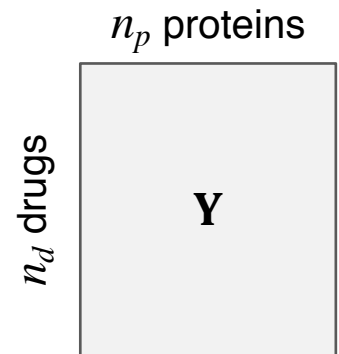
$\text{vec}(\cdot)$ vectorization operator that arranges the columns of a matrix into a vector

Eigen-decomposition of the kernel matrices \mathbf{K}_D and \mathbf{K}_P

$$\text{vec}(\mathbf{R}) = \underbrace{(\boldsymbol{\Lambda}_D \otimes \boldsymbol{\Lambda}_P + \lambda \mathbf{I}_N)^{-1}} \text{vec}(\mathbf{Q}_P^T \mathbf{Y}^T \mathbf{Q}_D)$$

$$\text{diag}(\boldsymbol{\Lambda}_D \otimes \boldsymbol{\Lambda}_P) = \text{diag}(\boldsymbol{\Lambda}_D) \otimes \text{diag}(\boldsymbol{\Lambda}_P) = \text{vec}(\text{diag}(\boldsymbol{\Lambda}_P) \text{diag}(\boldsymbol{\Lambda}_D)^T)$$

The above works if \mathbf{Y} has no missing values, i.e., $N = n_d \times n_p$ (small number of missing values can be imputed as a pre-processing step).



Pairwise KRR – shortcut

- It is possible to use algebraic properties of the Kronecker product to avoid the explicit computation of the pairwise kernel, and therefore significantly speed up the model training.

$$\begin{aligned}
 \boldsymbol{\alpha} &= (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y} \\
 &= (\mathbf{K}_D \otimes \mathbf{K}_P + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\
 &= ((\mathbf{Q}_D \boldsymbol{\Lambda}_D \mathbf{Q}_D^T) \otimes (\mathbf{Q}_P \boldsymbol{\Lambda}_P \mathbf{Q}_P^T) + \lambda \mathbf{I}_N)^{-1} \text{vec}(\mathbf{Y}) \\
 &= \text{vec}(\mathbf{Q}_P \mathbf{R} \mathbf{Q}_D^T)
 \end{aligned}$$

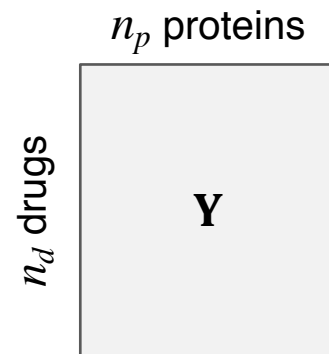
$\text{vec}(\cdot)$ vectorization operator that arranges the columns of a matrix into a vector

Eigen-decomposition of the kernel matrices \mathbf{K}_D and \mathbf{K}_P

$$\text{vec}(\mathbf{R}) = \underbrace{(\boldsymbol{\Lambda}_D \otimes \boldsymbol{\Lambda}_P + \lambda \mathbf{I}_N)^{-1}} \text{vec}(\mathbf{Q}_P^T \mathbf{Y}^T \mathbf{Q}_D)$$

$$\text{diag}(\boldsymbol{\Lambda}_D \otimes \boldsymbol{\Lambda}_P) = \text{diag}(\boldsymbol{\Lambda}_D) \otimes \text{diag}(\boldsymbol{\Lambda}_P) = \text{vec}(\text{diag}(\boldsymbol{\Lambda}_P) \text{diag}(\boldsymbol{\Lambda}_D)^T)$$

The above works if \mathbf{Y} has no missing values, i.e., $N = n_d \times n_p$ (small number of missing values can be imputed as a pre-processing step).



Multiple Kernel Learning (MKL)

- Classical kernel-based algorithms rely on a single kernel – the view resulting in the highest predictive performance is considered the best one.
- Risk of losing some important information by dropping all the other views.
- Ideally, one would like to learn the importance of each kernel matrix in a given task, and then use a weighted combination of them:

$$\mathbf{K}_\mu = \sum_{l=1}^L \mu_l \mathbf{K}^{(l)}$$

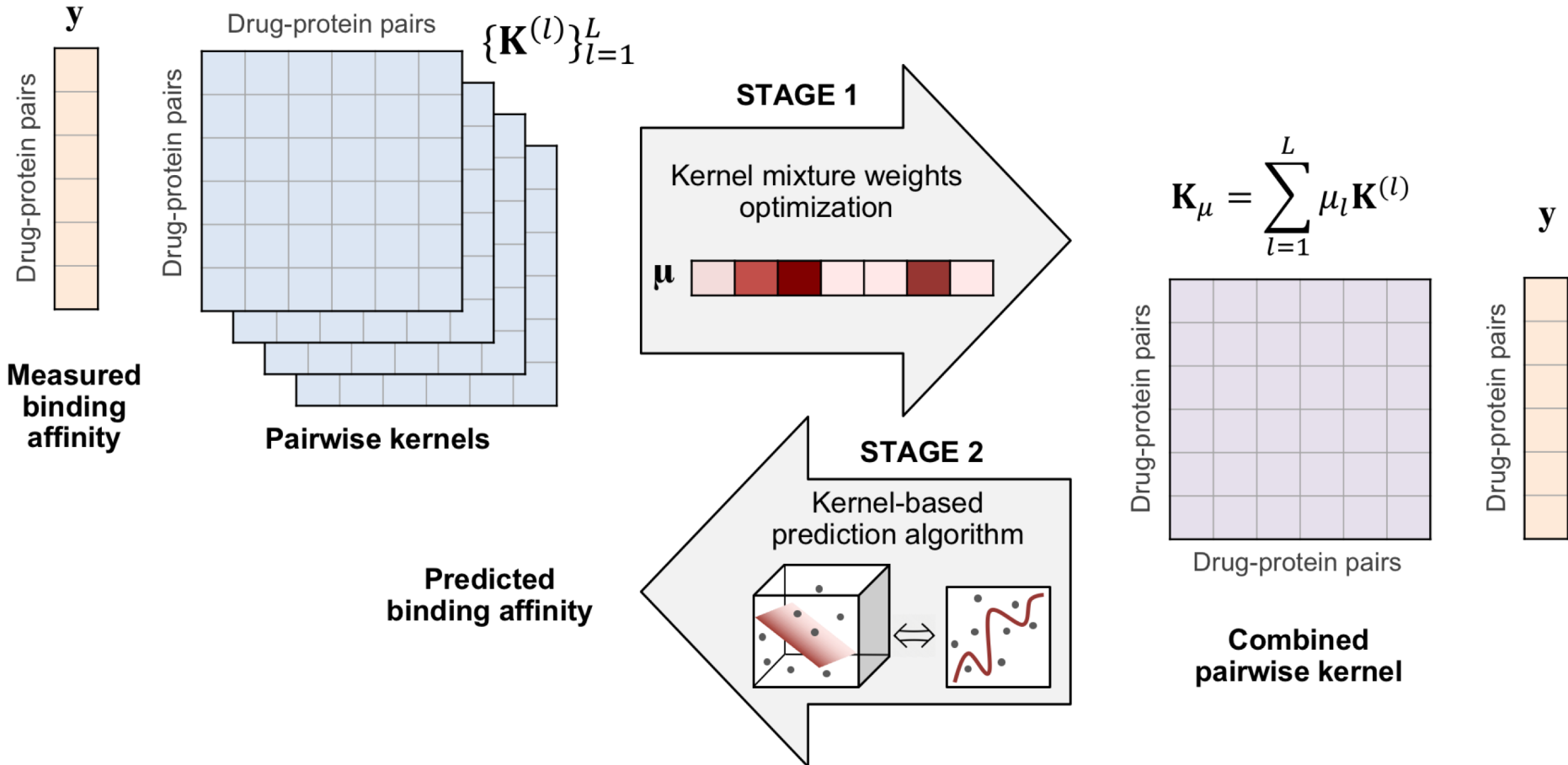
Multiple Kernel Learning (MKL)

- Classical kernel-based algorithms rely on a single kernel – the view resulting in the highest predictive performance is considered the best one.
- Risk of losing some important information by dropping all the other views.
- Ideally, one would like to learn the importance of each kernel matrix in a given task, and then use a weighted combination of them:

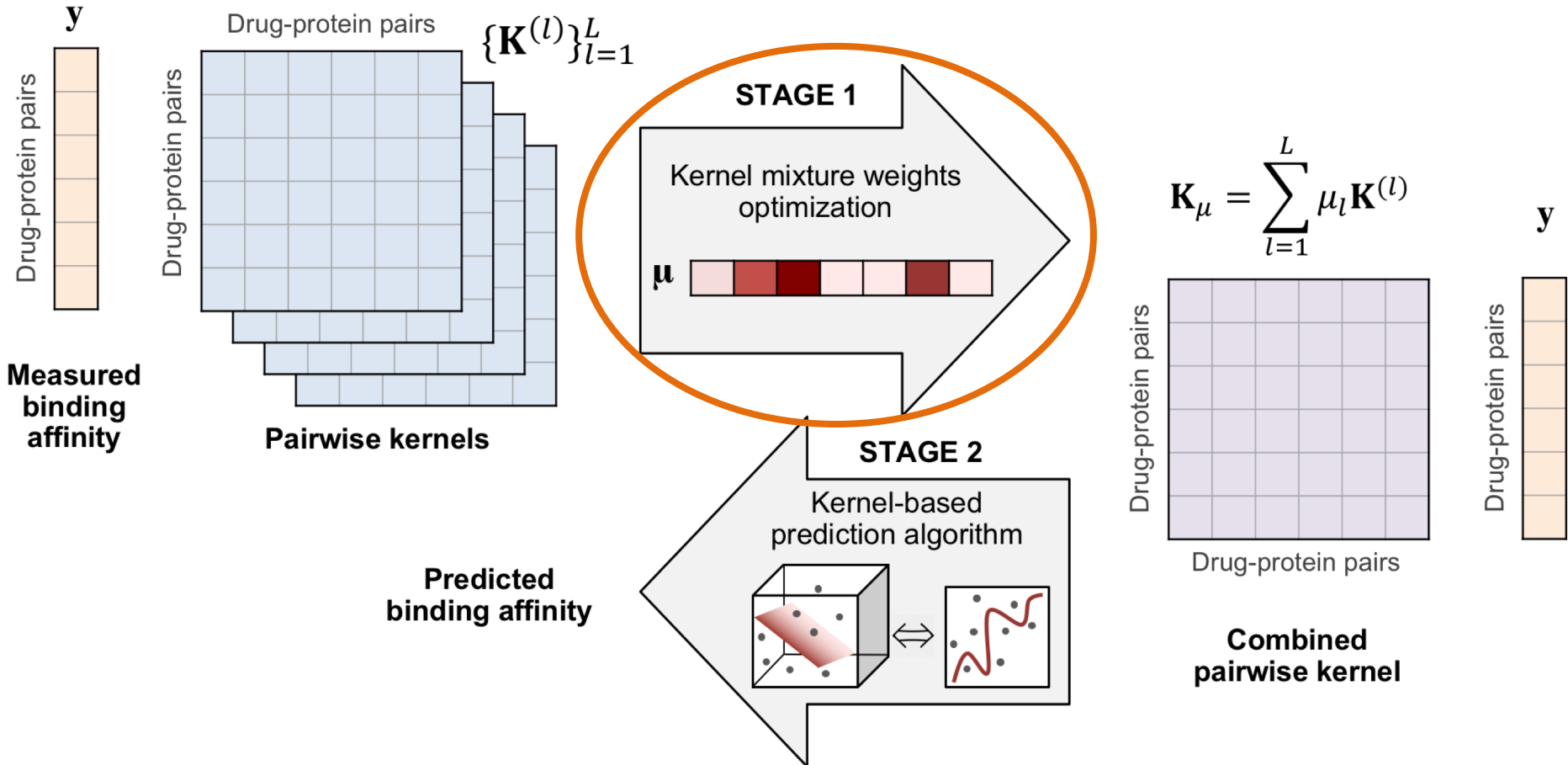
$$\mathbf{K}_\mu = \sum_{l=1}^L \mu_l \mathbf{K}^{(l)}$$

- One-stage MKL methods learn the kernel combination and prediction model parameters jointly.
- Two-stage MKL methods find the optimal kernel weights before subsequent phase of learning a classifier or regressor.

Two-stage MKL



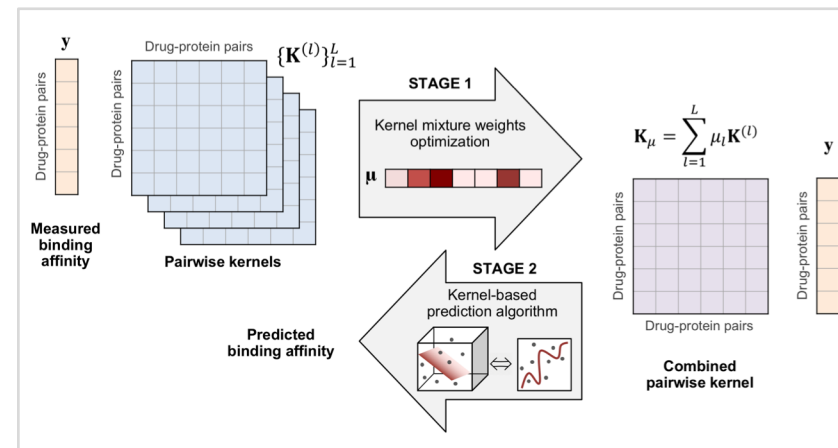
Two-stage MKL



Two-stage MKL

ALIGNF

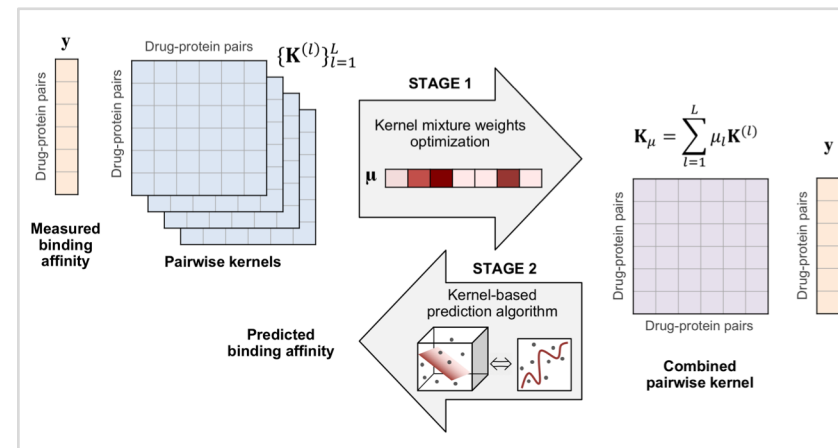
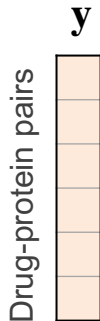
Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



Two-stage MKL

ALIGNF

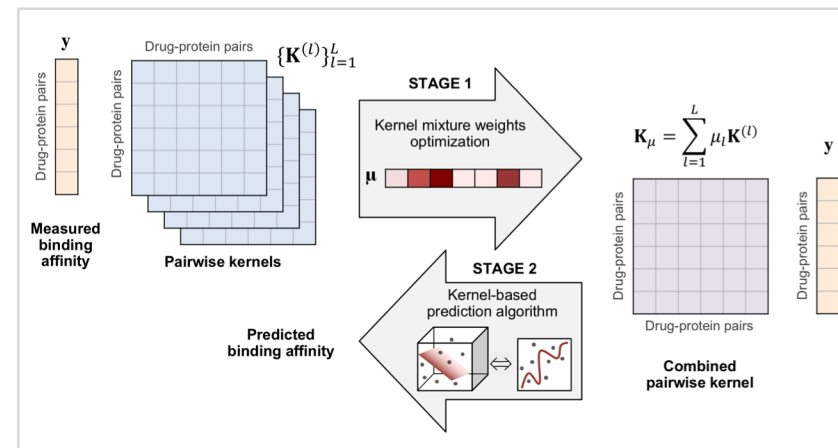
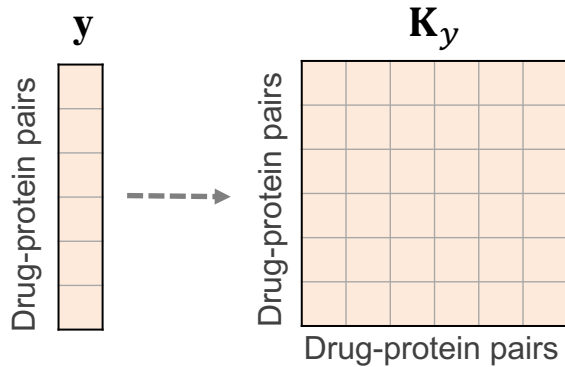
Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



Two-stage MKL

ALIGNF

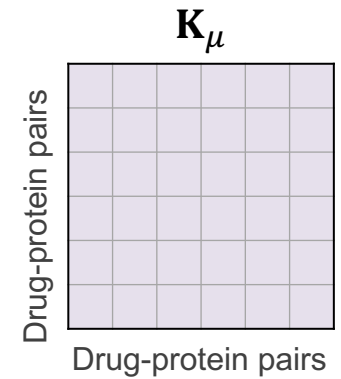
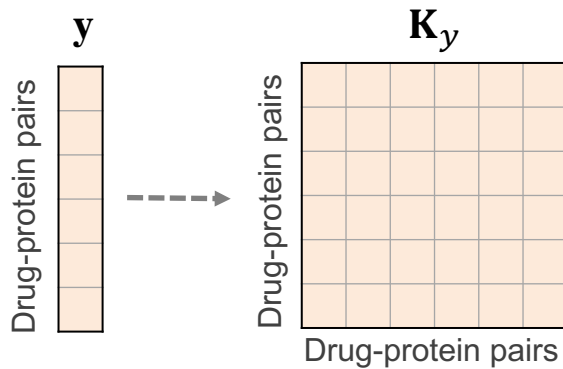
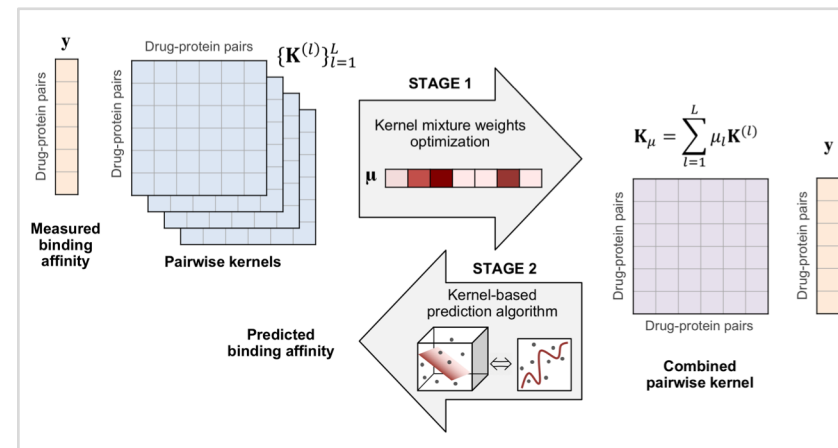
Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



Two-stage MKL

ALIGNF

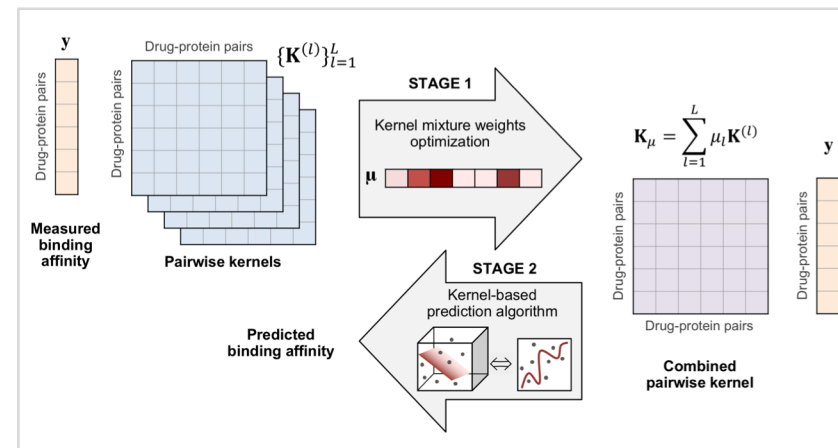
Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



Two-stage MKL

ALIGNF

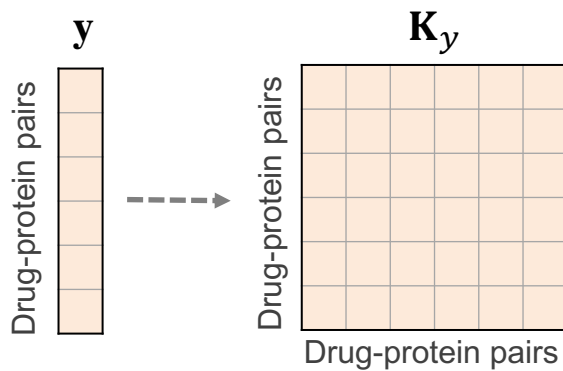
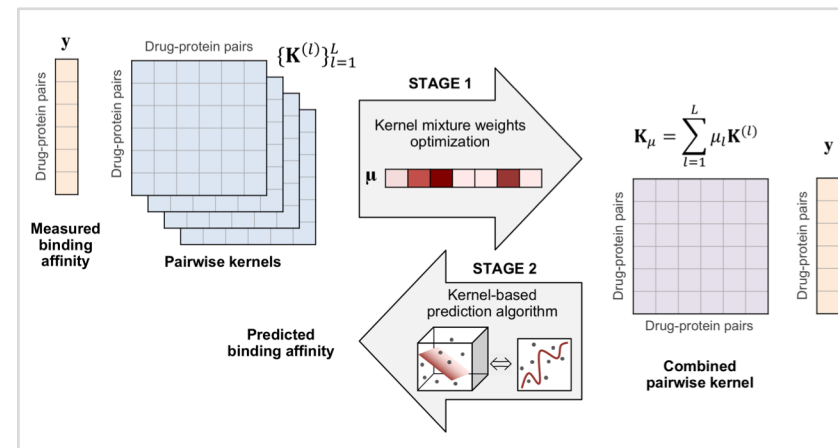
Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



Two-stage MKL

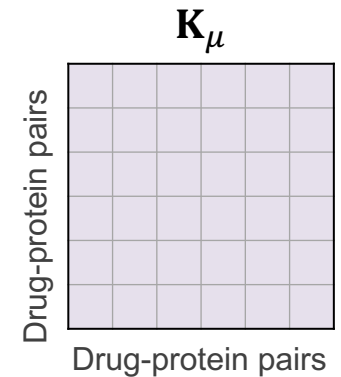
ALIGNF

Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



$$\arg \max_{\boldsymbol{\mu}} \hat{A}(\mathbf{K}_\mu^c, \mathbf{K}_y) = \max_{\boldsymbol{\mu}} \frac{\langle \mathbf{K}_\mu^c, \mathbf{K}_y \rangle_F}{\|\mathbf{K}_\mu^c\|_F},$$

subject to: $\|\boldsymbol{\mu}\|_2 = 1, \boldsymbol{\mu} \geq 0$.

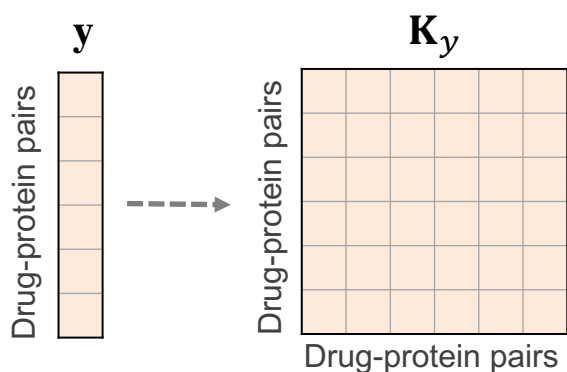
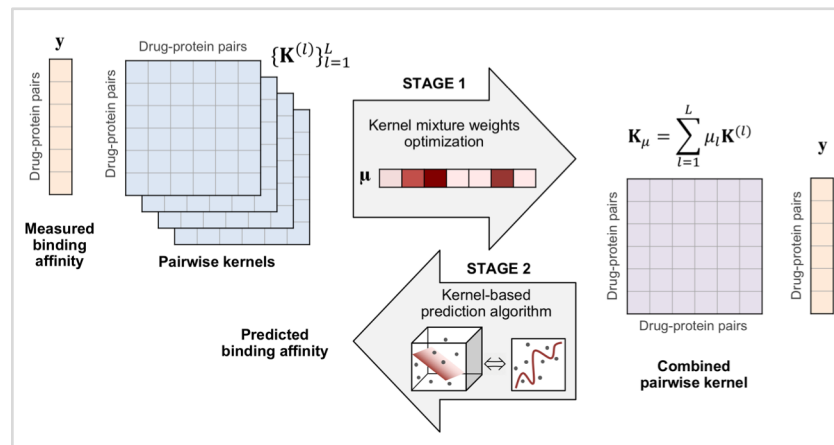


$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \text{vec}(\mathbf{A})^T \text{vec}(\mathbf{B})$$

Two-stage MKL

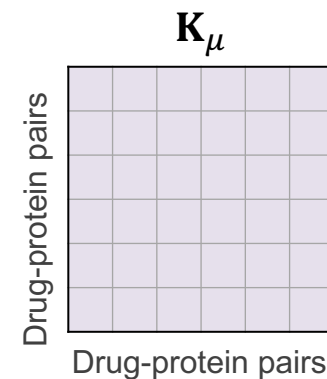
ALIGNF

Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



$$\arg \max_{\boldsymbol{\mu}} \hat{A}(\mathbf{K}_\mu^c, \mathbf{K}_y) = \max_{\boldsymbol{\mu}} \frac{\langle \mathbf{K}_\mu^c, \mathbf{K}_y \rangle_F}{\|\mathbf{K}_\mu^c\|_F},$$

subject to: $\|\boldsymbol{\mu}\|_2 = 1, \boldsymbol{\mu} \geq 0$.



$$\mathbf{K}^c = \mathbf{C}\mathbf{K}\mathbf{C}$$

$$\mathbf{C} = \left[\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{N} \right]$$

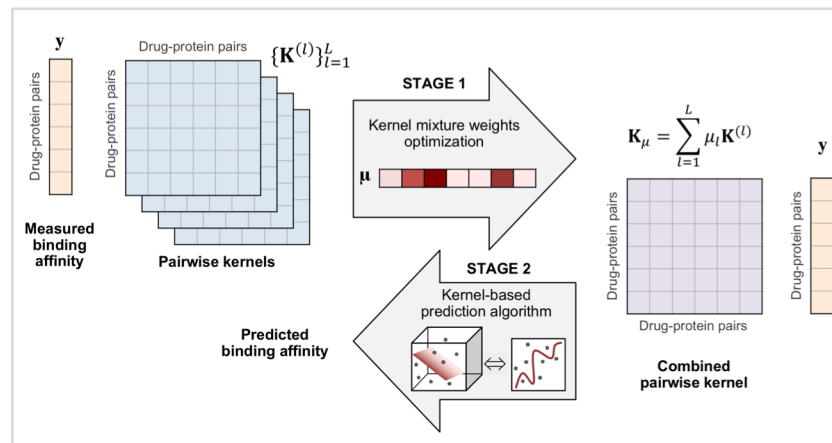
The sum of the rows (columns) of \mathbf{K}^c yields the zero vector $\mathbf{K}^c\mathbf{1} = \mathbf{0}$ ($\mathbf{1}^T\mathbf{K}^c = \mathbf{0}^T$).

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \text{vec}(\mathbf{A})^T \text{vec}(\mathbf{B})$$

Two-stage MKL

ALIGNF

Kernel mixture weights are determined by maximising the centered alignment between the combined kernel \mathbf{K}_μ and the response kernel \mathbf{K}_y .



$$\arg \max_{\mu} \hat{A}(\mathbf{K}_\mu^c, \mathbf{K}_y) = \max_{\mu} \frac{\langle \mathbf{K}_\mu^c, \mathbf{K}_y \rangle_F}{\|\mathbf{K}_\mu^c\|_F},$$

$$\text{subject to: } \|\mu\|_2 = 1, \mu \geq 0.$$

The above optimization problem can be solved via a simple **quadratic programming**:

$$\min_{\mathbf{v} \geq 0} \mathbf{v}^T \mathbf{M} \mathbf{v} - 2\mathbf{v}^T \mathbf{a},$$

$$(\mathbf{a})_i = \langle \mathbf{K}^{c(i)}, \mathbf{K}_y \rangle_F, \quad i = 1, \dots, L,$$

$$(\mathbf{M})_{ij} = \langle \mathbf{K}^{c(i)}, \mathbf{K}^{c(j)} \rangle_F, \quad i, j = 1, \dots, L.$$

Optimal kernel weights are given by $\mu^* = \frac{\mathbf{v}^*}{\|\mathbf{v}^*\|}$, where \mathbf{v}^* is the solution to the above QP.

MKL with pairwise kernels

10 drug kernels
12 protein kernels

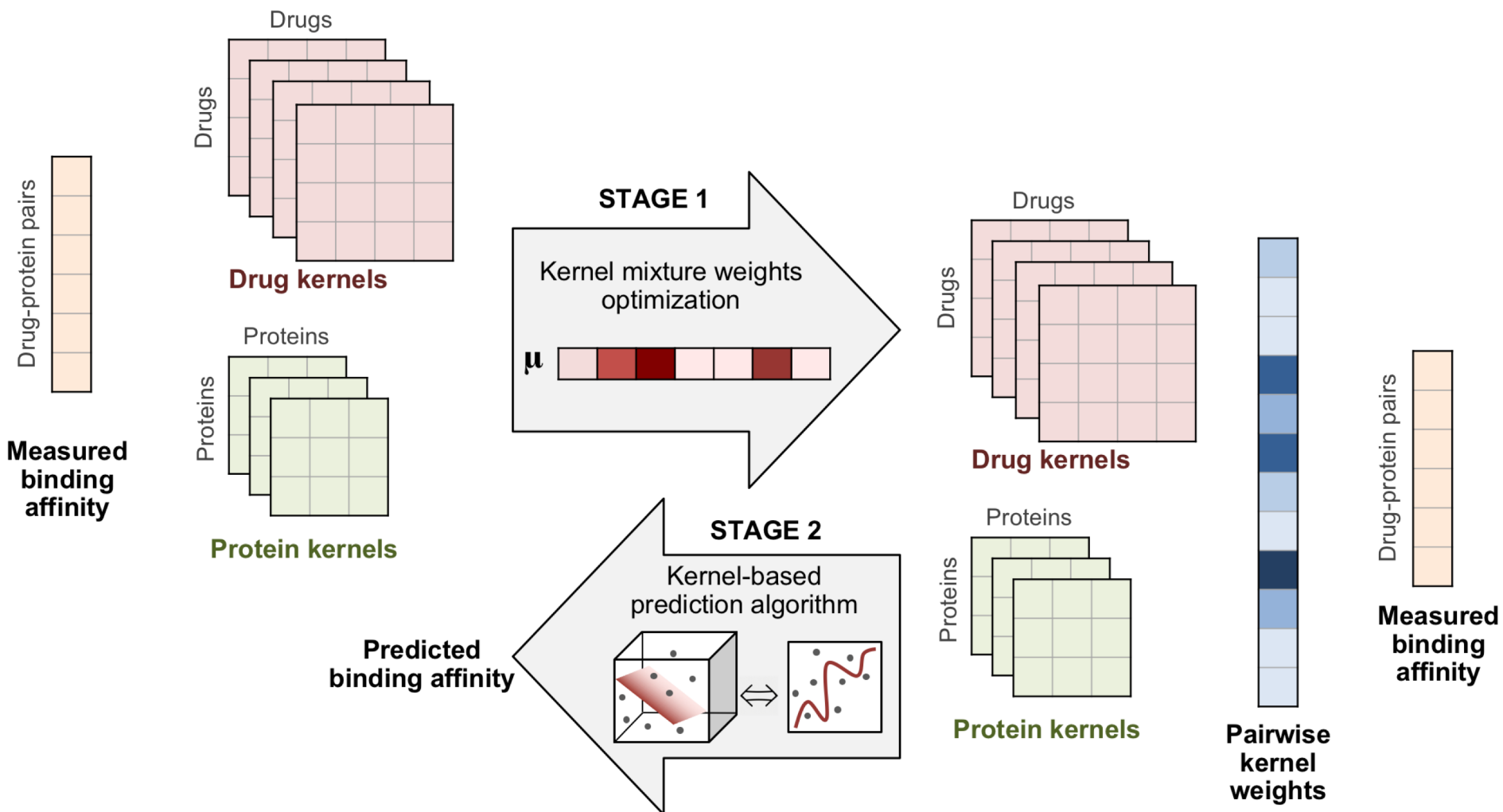
Again, the immense size of pairwise kernel spaces makes the model training infeasible in practical applications.

Number of drugs	Number of proteins	Memory [GB]	Time [h]
		ALIGNF	ALIGNF
50	50	9.810	2.976
60	60	20.290	7.797
70	70	37.750	17.678
80	80	64.000	37.691
90	90	103.180	77.408
100	100	156.890	145.312
110	110	229.670	>168.000 ^a
120	120	>256.000 ^b	>>168.000

^aProgram did not complete within 7 days (168h).

^bProgram did not run given 256GB of memory.

pairwiseMKL (i.e. ALIGNNF for pairwise kernels)



pairwiseMKL

Number of drugs	Number of proteins	Memory [GB]		Time [h]	
		ALIGNF	pairwiseMKL	ALIGNF	pairwiseMKL
50	50	9.810	0.001	2.976	0.003
60	60	20.290	0.001	7.797	0.005
70	70	37.750	0.043	17.678	0.057
80	80	64.000	0.044	37.691	0.069
90	90	103.180	0.046	77.408	0.087
100	100	156.890	0.048	145.312	0.106
110	110	229.670	0.050	>168.000 ^a	0.118
120	120	>256.000 ^b	0.053	>>168.000	0.123

^aProgram did not complete within 7 days (168h).

^bProgram did not run given 256GB of memory.

pairwiseMKL

- **Bottleneck** in using ALIGNF with pairwise kernels is the centering of the kernel, required by the algorithm

$$\mathbf{K}^c = \mathbf{C}(\mathbf{K}_D \otimes \mathbf{K}_P)\mathbf{C}$$

pairwiseMKL

- **Bottleneck** in using ALIGNF with pairwise kernels is the centering of the kernel, required by the algorithm

$$\mathbf{K}^c = \mathbf{C}(\mathbf{K}_D \otimes \mathbf{K}_P)\mathbf{C}$$

- **Key contribution:** factorized form for the centering operator

$$\mathbf{C} = \left[\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{N} \right] = \sum_{q=1}^2 \mathbf{Q}_D^{(q)} \otimes \mathbf{Q}_P^{(q)}$$

pairwiseMKL

- **Bottleneck** in using ALIGNF with pairwise kernels is the centering of the kernel, required by the algorithm

$$\mathbf{K}^c = \mathbf{C}(\mathbf{K}_D \otimes \mathbf{K}_P)\mathbf{C}$$

- **Key contribution:** factorized form for the centering operator

$$\mathbf{C} = \left[\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{N} \right] = \sum_{q=1}^2 \mathbf{Q}_D^{(q)} \otimes \mathbf{Q}_P^{(q)}$$

- Now, the quantities (inner products) required by ALIGNF can be computed without explicitly building the huge pairwise kernels

$$(\mathbf{M})_{ij} = \langle \mathbf{K}^{c(i)}, \mathbf{K}^{c(j)} \rangle_F = \sum_{q=1}^2 \sum_{r=1}^2 \text{tr}(\mathbf{Q}_D^{(q)} \mathbf{K}_D^{(i)} \mathbf{Q}_D^{(r)} \mathbf{K}_D^{(j)}) \text{tr}(\mathbf{Q}_P^{(q)} \mathbf{K}_P^{(i)} \mathbf{Q}_P^{(r)} \mathbf{K}_P^{(j)})$$

$$(\mathbf{a})_i = \langle \mathbf{K}^{c(i)}, \mathbf{K}_y \rangle_F = \langle \mathbf{y}, \mathbf{h} \rangle, \text{ where}$$

$$\mathbf{h} = \sum_{q=1}^2 \sum_{r=1}^2 \text{vec} \left((\mathbf{Q}_P^{(q)} \mathbf{K}_P^{(i)} \mathbf{Q}_P^{(r)}) \mathbf{Y} (\mathbf{Q}_D^{(q)} \mathbf{K}_D^{(i)} \mathbf{Q}_D^{(r)}) \right) \quad \mathbf{y} = \text{vec}(\mathbf{Y})$$

pairwiseMKL

- In multiple kernel learning for classification tasks, it is usual to choose the **response kernel** of the form:

$$(\mathbf{K}_y)_{ij} = y_i y_j = \begin{cases} +1, & \text{if } y_i = y_j \\ -1, & \text{if } y_i \neq y_j \end{cases}$$

pairwiseMKL

- In multiple kernel learning for classification tasks, it is usual to choose the **response kernel** of the form:

$$(\mathbf{K}_y)_{ij} = y_i y_j = \begin{cases} +1, & \text{if } y_i = y_j \\ -1, & \text{if } y_i \neq y_j \end{cases}$$

- This works in binary classification, since positive and negative classes are perfectly separated.

pairwiseMKL

- In multiple kernel learning for classification tasks, it is usual to choose the **response kernel** of the form:

$$(\mathbf{K}_y)_{ij} = y_i y_j = \begin{cases} +1, & \text{if } y_i = y_j \\ -1, & \text{if } y_i \neq y_j \end{cases}$$

- This works in binary classification, since positive and negative classes are perfectly separated.
- However, it fails completely with real values, as large numbers get large kernel values, and small numbers get small kernel values.

$$\begin{aligned} y_i = y_j = 1 & \Rightarrow y_i y_j = 1 \\ y_i = 1, y_j = 1000 & \Rightarrow y_i y_j = 1000 \end{aligned}$$

pairwiseMKL

- In multiple kernel learning for classification tasks, it is usual to choose the **response kernel** of the form:

$$(\mathbf{K}_y)_{ij} = y_i y_j = \begin{cases} +1, & \text{if } y_i = y_j \\ -1, & \text{if } y_i \neq y_j \end{cases}$$

- This works in binary classification, since positive and negative classes are perfectly separated.
- However, it fails completely with real values, as large numbers get large kernel values, and small numbers get small kernel values.

$$\begin{aligned} y_i = y_j = 1 &\Rightarrow y_i y_j = 1 \\ y_i = 1, y_j = 1000 &\Rightarrow y_i y_j = 1000 \end{aligned}$$

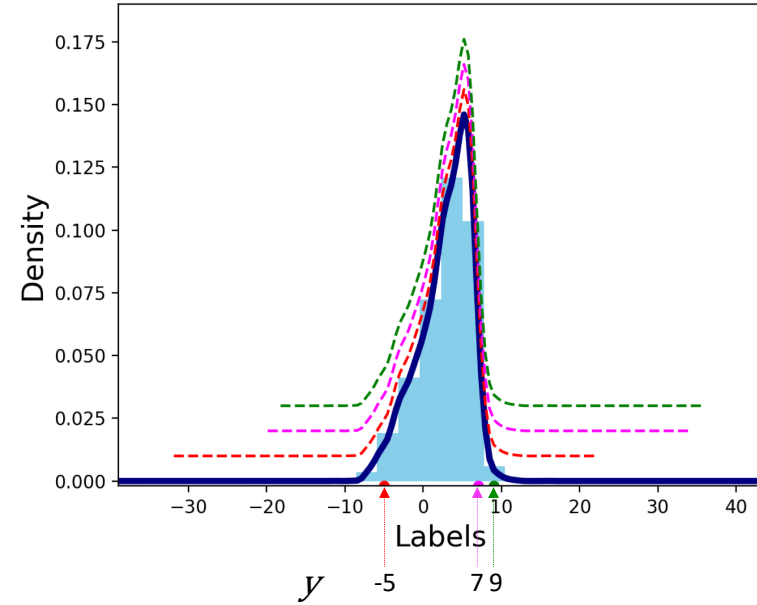
- The Gaussian kernel would work better as it is translation invariant.

$$(\mathbf{K}_y)_{ij} = \exp\left(-\frac{\|y_i - y_j\|^2}{2\sigma^2}\right)$$

- However, the factorized centering procedure requires explicit representation of the response matrix \mathbf{Y} ($\mathbf{y} = \text{vec}(\mathbf{Y})$).

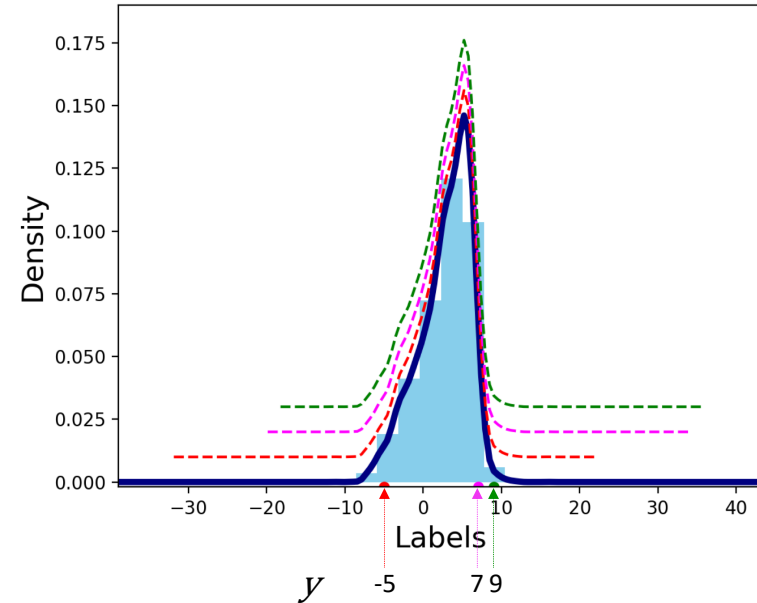
pairwiseMKL

- We start fitting a mixture of Gaussians onto the frequency histogram of the response variable, obtaining a density $f(b)$ for each bin b .



pairwiseMKL

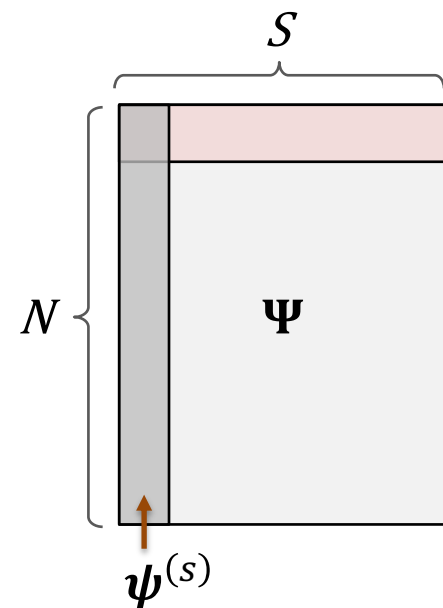
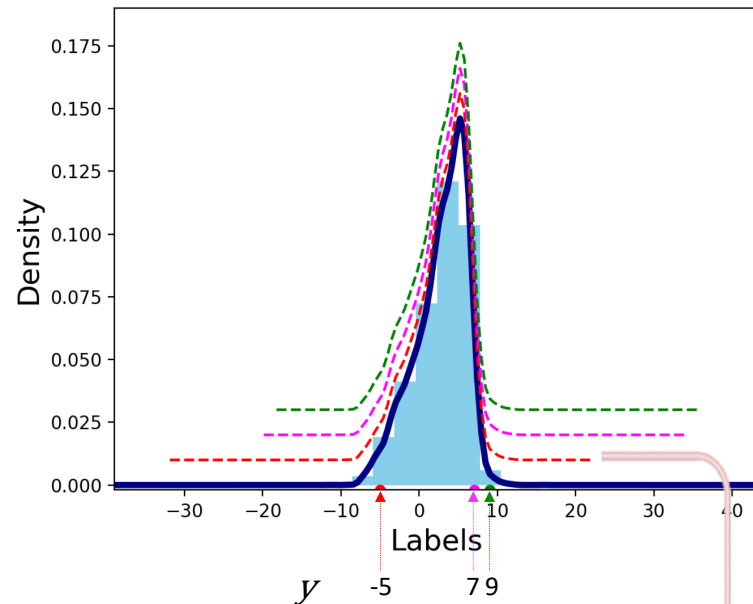
- We start fitting a mixture of Gaussians onto the frequency histogram of the response variable, obtaining a density $f(b)$ for each bin b .
- For each value y , a window of S bins around it is defined: $(b_y, b_y + 1, \dots, b_y + S - 1)$



pairwiseMKL

- We start fitting a mixture of Gaussians onto the frequency histogram of the response variable, obtaining a density $f(b)$ for each bin b .
 - For each value y , a window of S bins around it is defined:
- $$(b_y, b_y + 1, \dots, b_y + S - 1)$$
- **Feature vector for y** is read off the bin densities, and normalized.

$$\psi(y) = (f(b_y), f(b_y + 1), \dots, f(b_y + S - 1))$$



pairwiseMKL

- We start fitting a mixture of Gaussians onto the frequency histogram of the response variable, obtaining a density $f(b)$ for each bin b .
- For each value y , a window of S bins around it is defined: $(b_y, b_y + 1, \dots, b_y + S - 1)$

- **Feature vector for y** is read off the bin densities, and normalized.

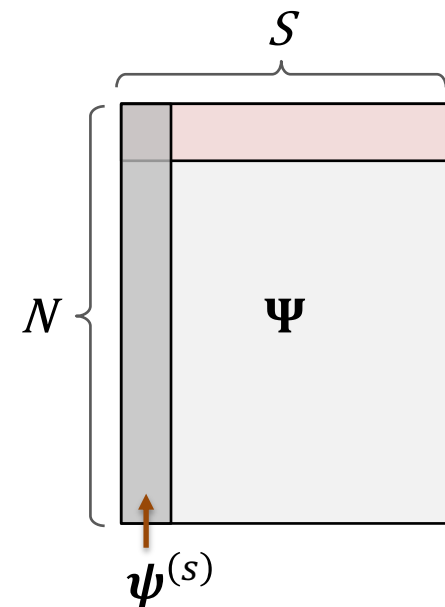
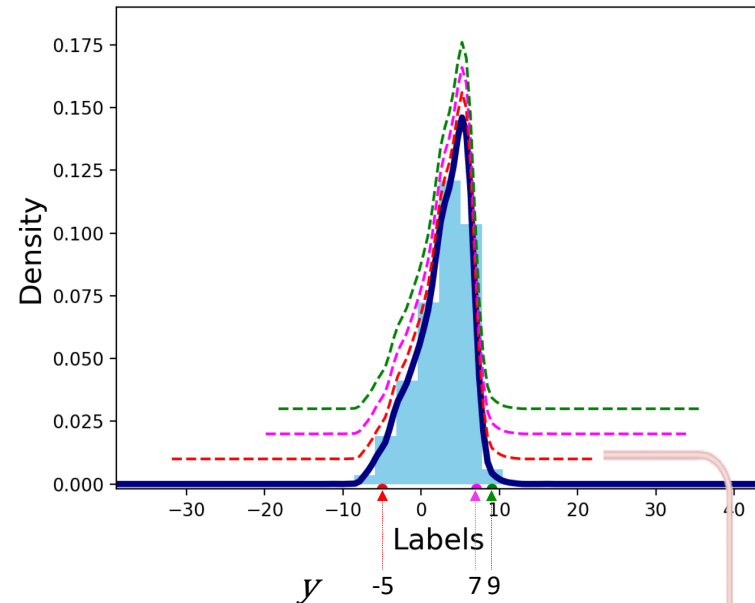
$$\psi(y) = (f(b_y), f(b_y + 1), \dots, f(b_y + S - 1))$$

- **Kernel:** sum of products of S bin densities.

$$\mathbf{K}_y = \sum_{s=1}^S \psi^{(s)} \psi^{(s)T}$$

$$\psi^{(s)} = (\psi^{(s)}(y_1), \dots, \psi^{(s)}(y_N))$$

- Intuitively, the kernel measures the alignment of the original density f with f shifted by $b_{yi} - b_{yj}$ bins.



pairwiseMKL

- We start fitting a mixture of Gaussians onto the frequency histogram of the response variable, obtaining a density $f(b)$ for each bin b .
- For each value y , a window of S bins around it is defined: $(b_y, b_y + 1, \dots, b_y + S - 1)$

- **Feature vector for y** is read off the bin densities, and normalized.

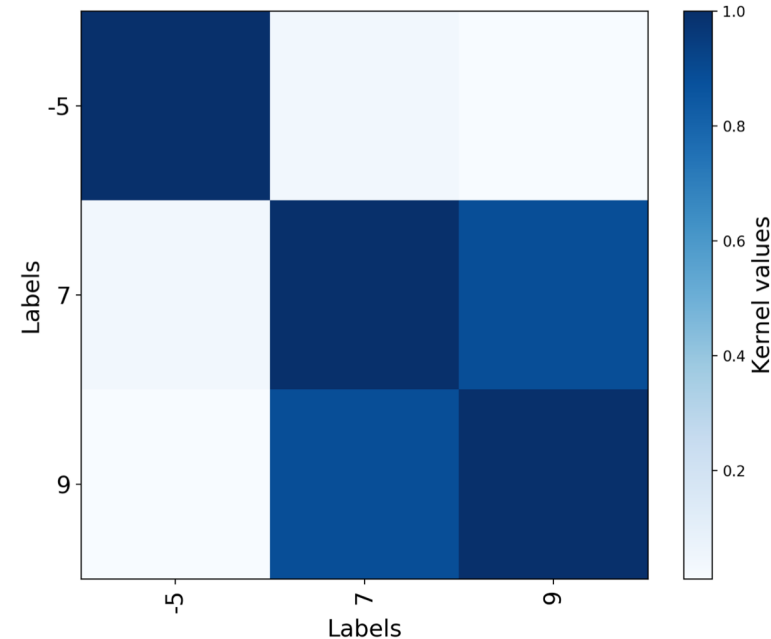
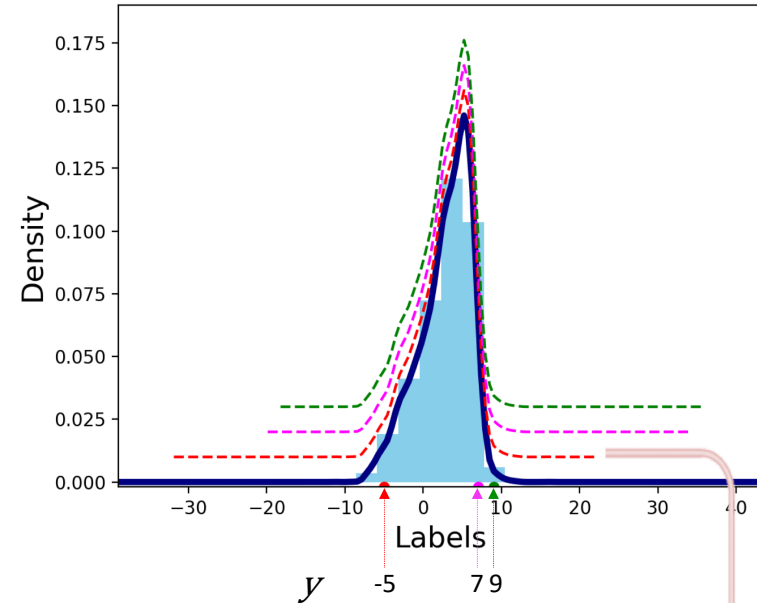
$$\psi(y) = (f(b_y), f(b_y + 1), \dots, f(b_y + S - 1))$$

- **Kernel:** sum of products of S bin densities.

$$\mathbf{K}_y = \sum_{s=1}^S \psi^{(s)} \psi^{(s)T}$$

$$\psi^{(s)} = (\psi^{(s)}(y_1), \dots, \psi^{(s)}(y_N))$$

- Intuitively, the kernel measures the alignment of the original density f with f shifted by $b_{yi} - b_{yj}$ bins.



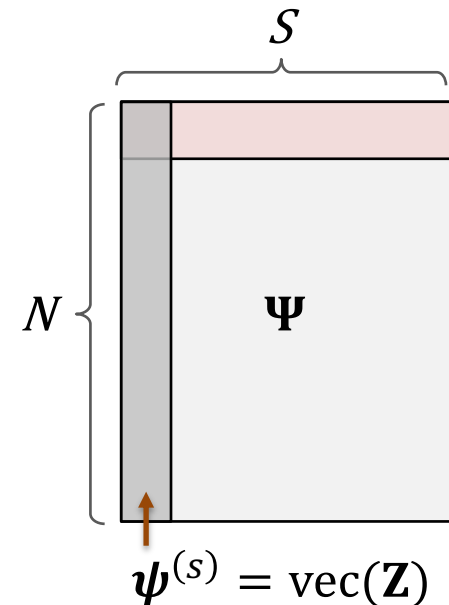
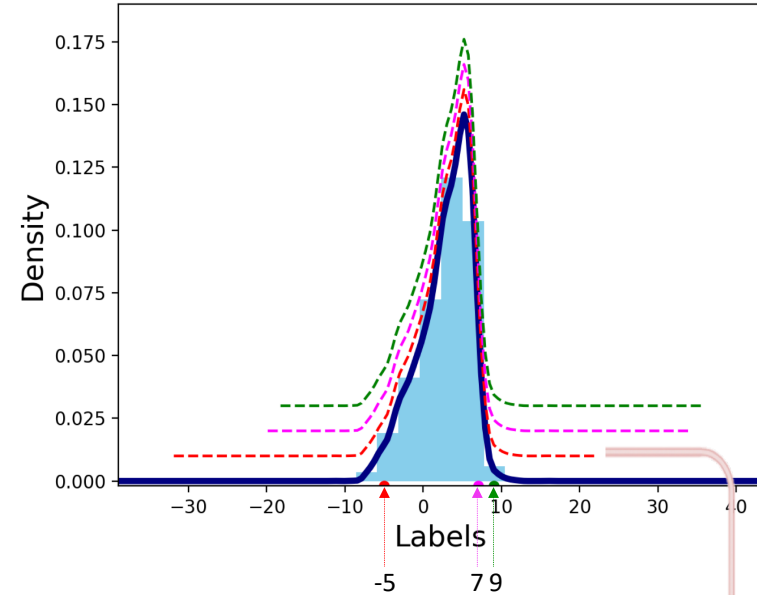
pairwiseMKL

$$\mathbf{K}_y = \sum_{s=1}^S \boldsymbol{\psi}^{(s)} \boldsymbol{\psi}^{(s)T}$$

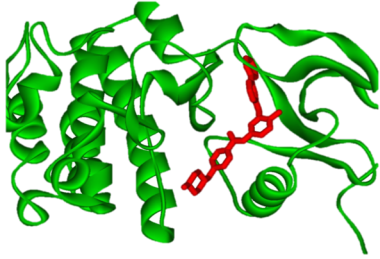
- We can now compute the centered kernel alignment between each input kernel and the Gaussian response kernel:

$$(\mathbf{a})_i = \langle \mathbf{K}^{c(i)}, \mathbf{K}_y \rangle_F = \sum_{s=1}^S \langle \boldsymbol{\psi}^{(s)}, \mathbf{w} \rangle,$$

$$\mathbf{w} = \sum_{q=1}^2 \sum_{r=1}^2 \text{vec} \left((\mathbf{Q}_P^{(q)} \mathbf{K}_P^{(i)} \mathbf{Q}_P^{(r)}) \mathbf{Z} (\mathbf{Q}_D^{(q)} \mathbf{K}_D^{(i)} \mathbf{Q}_D^{(r)}) \right)$$



pairwiseMKL: application example

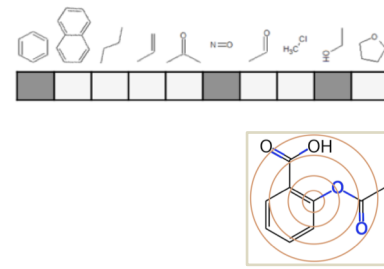


- Bioactivity data:
drug–target interaction map comprising **167995** pIC_{50} values between 2967 kinase inhibitors and 226 kinases (Merget et al., 2016).
- 10 drug kernels x 312 kinase kernels
→ **3120 pairwise kernels**.

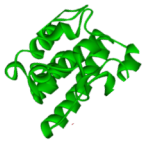


Drug kernels

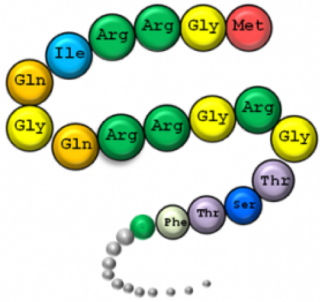
10 fingerprint-based Tanimoto kernels



- **Kd-circular** Extended connectivity 1024-bit fingerprint (ECFP6).
- **Kd-estate** 79-bit fingerprint corresponding to Estate substructures.
- **Kd-ext** Path-based, hashed 1024-bit fingerprint taking into account ring systems.
- **Kd-graph** Path-based, hashed 1024-bit fingerprint considering connectivity.
- **Kd-hybr** Path-based, hashed 1024-bit fingerprint considering hybridization states.
- **Kd-kr** 4860-bit fingerprint defined by Klekota and Roth (2008).
- **Kd-maccs** 166-bit fingerprint based on MACCS structural keys.
- **Kd-PubCh** 881-bit fingerprint defined by PubChem.
- **Kd-sp** 1024-bit fingerprint based on the shortest paths between atoms taking into account ring systems and charges.
- **Kd-std** Path-based, hashed 1024-bit fingerprint.



Protein kinase kernels



- Full amino acid sequences.
- Amino acid sub-sequences of kinase domains.
- Amino acid sub-sequences of ATP binding pockets.

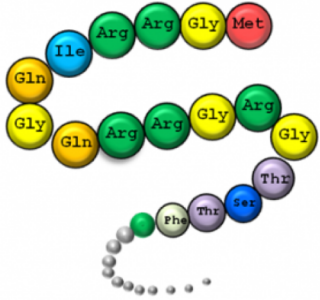
GO profile composed of:

- 415 GO terms from *molecular function* domain;
- 3926 GO terms from *biological process* domain;
- 352 GO terms from *cellular component* domain.

GO



Protein kinase kernels



- Full amino acid sequences.
 - Amino acid sub-sequences of kinase domains.
 - Amino acid sub-sequences of ATP binding pockets.
- Smith-Waterman (SW) kernel.
 - Generic string (GS) kernel (Giguère et al., 2013).

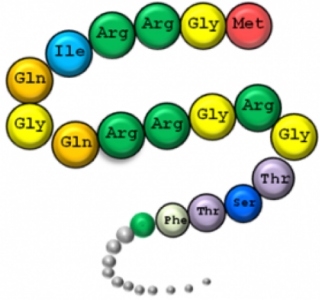
GO profile composed of:

- 415 GO terms from *molecular function* domain;
- 3926 GO terms from *biological process* domain;
- 352 GO terms from *cellular component* domain.

GO



Protein kinase kernels



- Full amino acid sequences.
 - Amino acid sub-sequences of kinase domains.
 - Amino acid sub-sequences of ATP binding pockets.
- Smith-Waterman (SW) kernel.
 - Generic string (GS) kernel (Giguère et al., 2013).

GO profile composed of:

- 415 GO terms from *molecular function* domain;
- 3926 GO terms from *biological process* domain;
- 352 GO terms from *cellular component* domain.

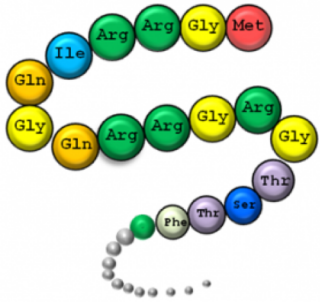
Gaussian kernel.

GO



Protein kinase kernels

312 protein kernels



- Full amino acid sequences.
- Amino acid sub-sequences of kinase domains.
- Amino acid sub-sequences of ATP binding pockets.

- Smith-Waterman (SW) kernel. **0 hyperparameters**
- Generic string (GS) kernel (Giguère et al., 2013). **3 hyperparameters**

GO profile composed of:

















- 415 GO terms from *molecular function* domain;
- 3926 GO terms from *biological process* domain;
- 352 GO terms from *cellular component* domain.

Gaussian kernel.

1 hyperparameter

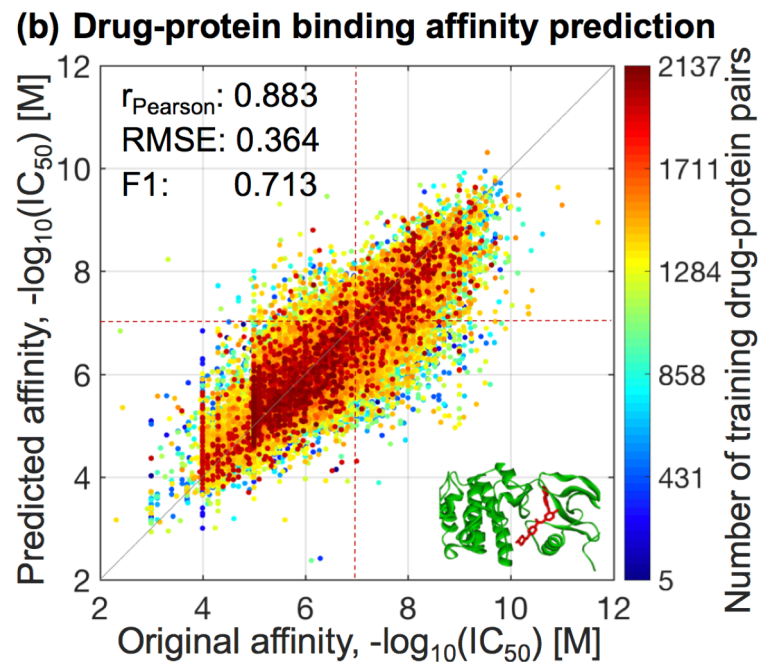
GO

pairwiseMKL: CV results

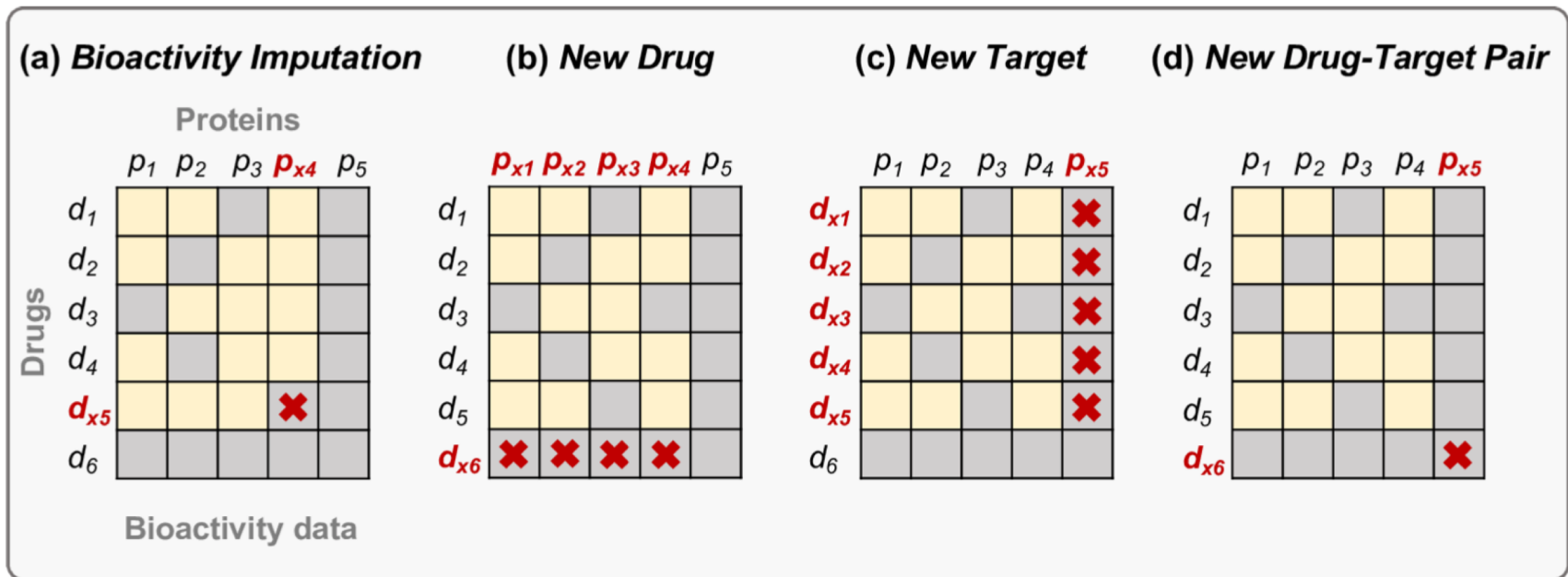
Drug kernel	⊗	Protein kernel	Pairwise kernel weight
 shortest paths		 ATP binding pocket + GS kernel	0.37
 circular		 ATP binding pocket + GS kernel	0.36
 Klekota and Roth		 ATP binding pocket + GS kernel	0.09
 circular		 kinase domain + GS kernel	0.07
 circular		 GO biological process + Gaussian kernel	0.06
 circular		 GO cellular component + Gaussian kernel	0.02
 circular		 kinase domain + SW kernel	0.02
 circular		 full sequence + GS kernel	0.01




Selected pairwise kernels:
8 of 3120

Cichonska A *et al.* (2018) “Learning with multiple pairwise kernels for drug bioactivity prediction”.
Bioinformatics.




Pairwise prediction scenarios



-  Experimentally-measured binding affinity
-  Unmeasured binding affinity
-  Unmeasured binding affinity to be predicted

- d Drug
- p Protein
- (d_x, p_x) Query drug-protein pair, the binding affinity of which is to be predicted

Cichonska A *et al.* (2017) “Computational-experimental approach to drug-target interaction mapping: A case study on kinase inhibitors”. PLOS Computational Biology.



IDG-DREAM Drug Kinase Binding Prediction Challenge

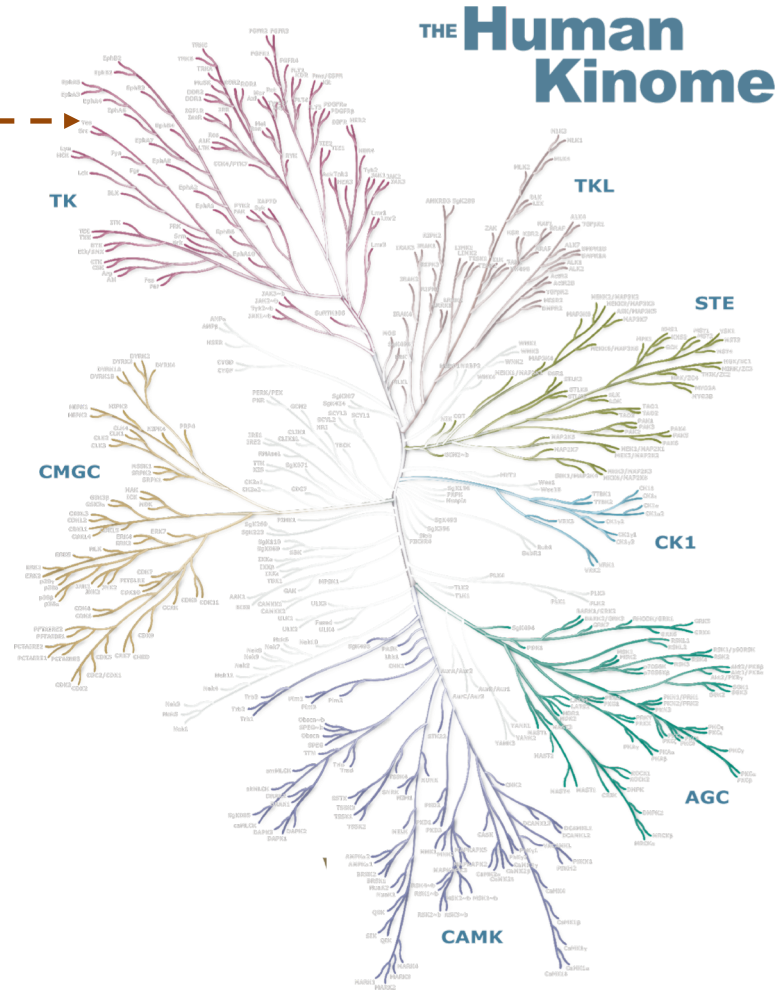
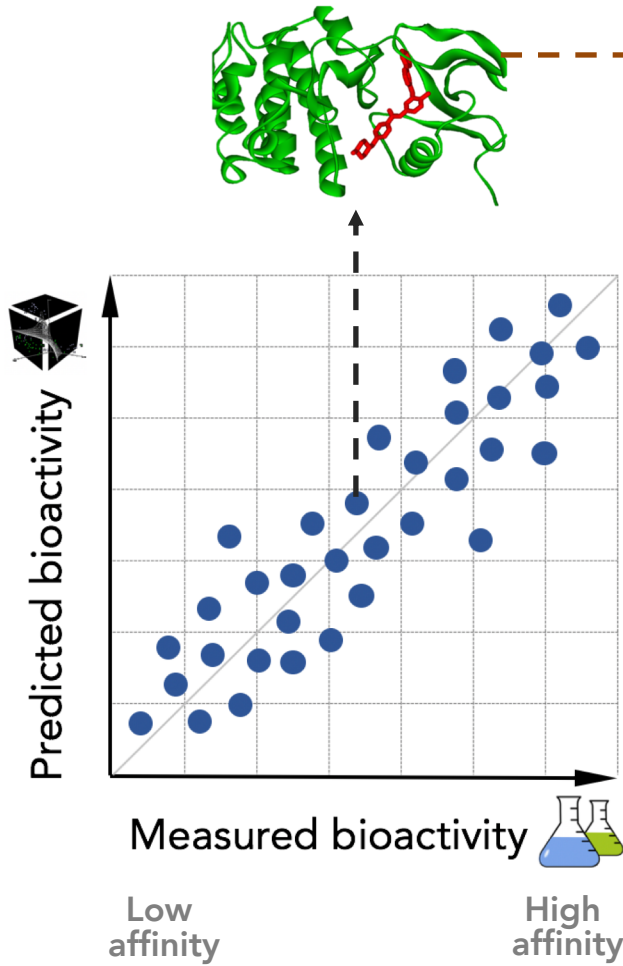


www.synapse.org/DrugKinaseChallenge

OVERALL AIMS

- To evaluate machine learning models as systematic tools for guiding drug–protein mapping efforts to prioritize most potent and selective agents for further experimental evaluation.
- The participating teams were challenged with three overall questions:
 - **What are the best machine learning approaches for predicting drug-protein binding affinities?**
 - **What are the most predictive compound and protein features?**
 - **What are the best bioactivity data for the model training?**

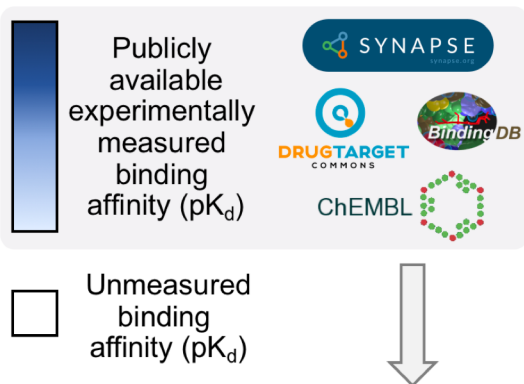
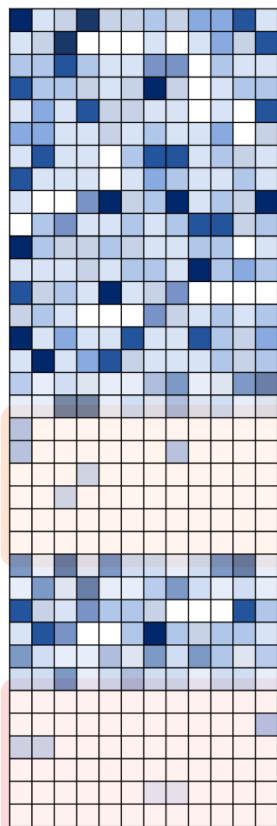
Scope



Overview of the IDG-DREAM Drug-Kinase Binding Prediction Challenge



Kinases



Model dockers and training bioactivity data from the teams made publicly available

Round 1
To predict pK_d values of 430 compound-kinase pairs

77 teams



+ 1 baseline (BL) model

Round 2
To predict pK_d values of 394 compound-kinase pairs

54 teams



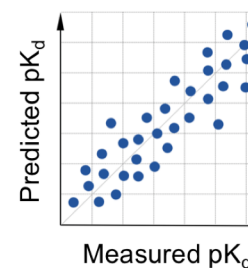
Round 1 evaluation:
15 November 2018

169 submissions

Round 2 evaluation:
18 April 2019

99 submissions

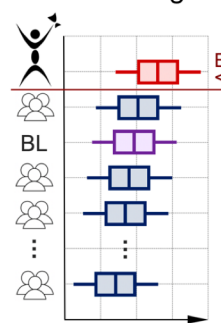
Round 1: submissions scored across six evaluation metrics but remained unranked



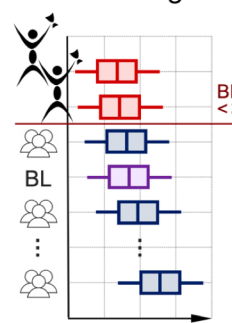
- RMSE
- Pearson correlation
- Spearman correlation
- Concordance index
- F1 score
- Average AUC

Round 2

Subchallenge 1



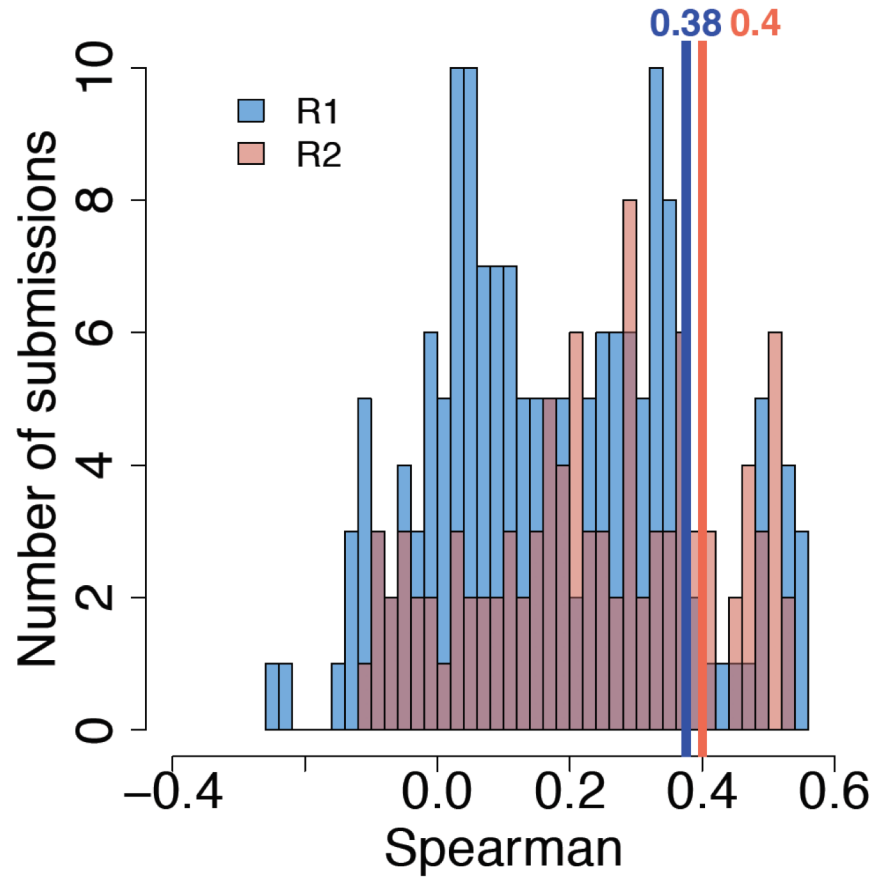
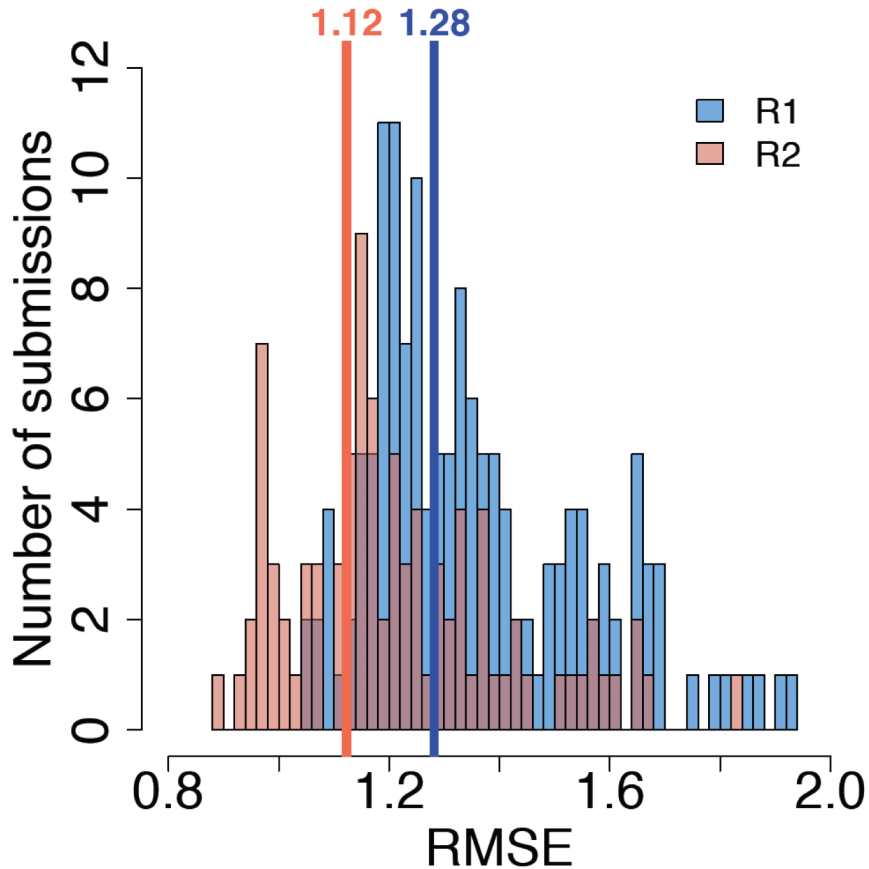
Subchallenge 2



Dissociation constant K_d indicates the concentration of a compound at which 50% of target kinase molecules exists in the compound-kinase complex.

$$pK_d = -\log_{10}K_d [M]$$

Results: Round 1 vs. Round 2



Thank you!