Tutorial 2

Compound profiling using QSAR modeling

Gilles Marcou, Dragos Horvath and Alexandre Varnek

Laboratory of Chemoinformatics, University of Strasbourg, UMR7140 UniStra/CNRS, 1 rue Blaise Pascal 67000 Strasbourg

Keywords: QSAR, Chemoinformatics, profiling, multi-task learning, dopamine.

Abstract: This tutorial is dedicated to prediction of pharmacological profiling of compounds. pharmacological profile of a given compound corresponds to ensemble of its biological activities (or any other properties). A “classical” strategy to build the compound’s profile implies obtaining individual QSAR models for each activity. There are two main drawbacks of this approach: for each individual model one should (*i*) tune the model’s parameters, and (*ii*) setup validation procedure. This problem could be efficiently resolved using multi-task learning algorithms. In this tutorial, several algorithms are considered. Their efficiency is compared to the corresponding single task (classical) approach. The tutorial describes different ways to assess the models’ performance, parameters selection for multi-tasks algorithms and interference of individual modeling tasks.

1. Introduction

Available experimental information on compound profiling is growing up since a large amount of data reach public domain. Thus, PubChem (Bolton, Wang, Thiessen, & Bryant, 2008) and ChEMBL (Gaulton, et al., 2011) databases aggregate an enormous amount of data from screening campaigns. Therefore, a given compound can be associated to an array of biological activities issued from different screening experiments. This brings a new level of data complexity which represents a new challenge to QSAR modeling.

In a classical, QSAR model (single task approach) one activity is related to chemical descriptors. This "single task" modeling process is rather well documented, see for instance (Tropsha, 2010). It generally includes the following steps: a choice of machine learning algorithm, processing of the dataset into a cross-validation loop, tuning the algorithms parameters on training set of each fold, model's validation on external test set of a given fold, models application on new data controlled by some applicability domain.

If several activities have to be modeled one same procedure is repeatedly applied to each of them. This process requires parameters optimization and a validation procedure for each task, which is computationally inefficient.

Besides, since the pioneering work of Caruana in 1997 (Caruana, 1997), it is expected that building simultaneously several models could be beneficial. If the tasks are related, simultaneous learning of the data may lead to the models of better performances. A whole class of machine learning algorithms - *multi-task learning* (MTL) – has been developed and described in the data mining literature.

Unlike single task learning (STL) algorithms, MTL have two particular features: (*i*) models for all tasks are built simultaneously and (*ii*) each task interferes with other tasks. For instance, it is straightforward to generalize the equations of multi-linear regression, using a Frobenius norm, so that instead of a vector of weights, the final model represents a matrix of weights. However, this would be equivalent to a learning the same number of independent single task multi-linear models. To avoid that, a MTL algorithm includes a communication between the individual learning tasks.

There exist different ways to implement MTL. The most popular one is algorithm of neural networks with multiple neurons in the output layer, see for instance (Varnek, Gaudin, Marcou, Baskin, Panday, & Tetko, 2009). However, it seems that a MTL version exists for many other machine-learning methods. Here, we focus on linear models obtained using the Lasso algorithm which provides with different options to perform MTL.

The MTL paradigm introduces the problem of measuring predictive performance the models and its comparison with the STL algorithm? These questions are intimately linked to the parameters optimization.

Finally, the MTL paradigm introduces a new degree of freedom into the modeling process. It is expected that MTL should be beneficial to *related* tasks. In practice which tasks could be considered as being related? How one can decide which tasks should be modeled together or considered separately?

Although, this tutorial won't bring a definite answer to these questions, it will illustrate them and, in some cases, will propose some simple solutions.

2. Starting with MATLAB and MALSAR

In this tutorial we use the MATLAB environment (MATLAB and Statistics Toolbox Release 2014a, 2014) containing MALSAR (Muti-tAsk Learning via StructurAl Regularization) package . (Zhou, Chen, & Ye, 2012) implementing 25 original implementations of MTL.

Install MATLAB

The editor of MATLAB, "Mathworks" provides us with demo licenses. Only the basic MATLAB system is needed; no additional toolboxes are required.

In order to download the software for your platform you have to follow instructions below:

1. Create an account on Mathworks website: http://www.mathworks.fr
2. Use this account:

login g.marcou@unistra.fr / password gil$500

3. In the section Support/Download products, chose the version R2014A of Matlab for your platform and download it.

4. Download the file licence.dat.
5. Follow the instructions to install MATLAB on your platform. During the installation procedure you can give the location of the file license.dat. Alternatively, you can copy this file in the installation directory of MATLAB.
Important note for Mac and Linux users: MATLAB can sometime crash with the following error:
Error using (…) loading error: dlopen: cannot load any more object with static TLS
This error is due to a particular initialization of some shared libraries loaded by MATLAB. In some systems only small number of libraries can be simultaneously loaded. Unfortunately the built-in doc system uses a lot of libraries, therefore the use og WEB documentation instead built-in documentation is strongly recommended. Otherwise MATLAB proposes a patch: http://www.mathworks.de/support/bugreports/961964.

MALSAR

MALSAR is a set of tools within MATLAB which provides with MTL algorithm for regression, classification and clustering. It is shipped with pre-compiled libraries for Windows 32 / 64 bits and for Mac Intel 64 bits. For other system a compilation procedure is needed that will require development tools.

To install MALSAR, proceed as follows:
1. Load MALSAR:
http://www.public.asu.edu/~jye02/Software/MALSAR/downloads/MALSAR1.1.zip
2. Create a directory called MALSAR and unzip the downloaded archive into this directory. If you have a Windows 32/64 bit system or a Mac Intel 64 bit system, installation is finished.
3. For others, launch MATLAB, add the MALSAR directory you created, and all subdirectories to the MATLAB search path (see below *Exercise 0: Introduction to the MATLAB system*) and run the command INSTALL.M into the **Command Window**. If there are no error messages, the installation is complete.

Additional functions are provided on the USB key of the workshop in the directory CS3-2014/Tutorials/QSAR_Profiling/Scripts/. They are also available for download on the web site of the conference: http://tiny.url/CS3-2014-Tuto3.

3. Dataset

The dataset studied in this tutorial is composed of 2965 affinity data (pKi) for 1597 compounds for the dopamine receptor family of GPCRs (Brown, Okuno, Marcou, Varnek, & Horvath, 2014). Affinity measures are provided for D1, D2, D3, D4 and D5 subfamilies. For each compound affinity data are available for one or several subfamilies in the following proportion:

- One dopamine subfamily: 41%
- Two subfamilies: 38%
- Three subfamilies: 17%
- Four subfamilies: 3%
- Five subfamilies: 1%

With respect to subfamilies, the affinity measures are distributed as follows:

Dopamine subfamily	**D1**	**D2**	**D3**	**D4**	**D5**
Number of compounds	272	1325	846	424	98

Table 1. Repartition of compounds across all 5 dopamine subfamilies for this dataset.

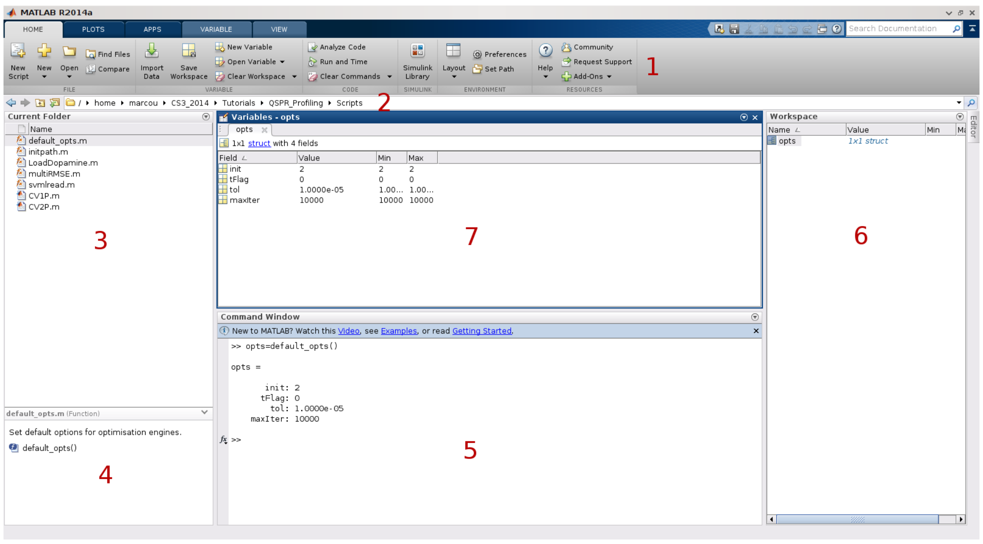
Beside, these five subfamilies are clustered into two groups, D1 and D5, on the one hand, and D2, D3 and D4, on the other hand.

4. Exercise 0: Introduction to the MATLAB system

The Matlab main window is divided into 7 areas (Figure 1):

The main menu banner, area 1. This menu give access to a number of common actions like creating a new script, managing variables into the memory, access to plotting tools, interface with MATLAB toolboxes, edit/run/debug scripts and functions.

- *The file and folder management, area 2,3 and 4*. For MATLAB, any script and function must be stored into a file using the .m termination. They can be used into the interface only if MATLAB knows where to find them. Therefore, managing files and folder is critical in order to tell MATLAB which scripts and functions are available. The area 2 is a breadcrumb used to navigate the user's directory tree. The area 3 displays the content of the current folder. By right clicking on a folder, it is possible to add it to the search path of MATLAB. By clicking on a file, in this area, if it is recognized as a MATLAB script or function, additional information are displayed into the area 4.
- *The Command Window, area 5*. Most commands will be entered into this area. If a command produces printing of results, they will be displayed also into this area. All commands can be re-called using the up arrow of the keyboard. Note that keyboard shortcuts are system dependent in MATLAB; for instance copy is Ctrl-W, cut is Alt-W and paste is Ctrl-Y on Linux and it is the usual Ctrl-X and Ctrl-V respectively, on Windows.
- *The Workspace, area 6*. This area registers all variables that are currently in memory. The type of the variable and some indications about its content are displayed. By clicking on a variable, it opens the area 7.
- *The Variables editor, area 7*. Any variable of the workspace can be visualized and its content can be edited using this tool.



Set default options for optimisation engines.
default_opts()

1x1 struct with 4 fields

Field	Value	Min	Max
init	2	2	2
tFlag	0	0	0
tol	1.0000e-05	1.00...	1.00...
maxIter	10000	10000	10000

New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
>> opts=default_opts()

opts = 

       init: 2
      tFlag: 0
        tol: 1.0000e-05
    maxIter: 10000

>>
```

Name	Value	Min	Max
opts	1x1 struct		

Figure 1. View of the main interface of MATLAB. It is divided typically into a main menu banner (1), a folder and file management tool (2,3,4), a command frame (5), a workspace frame (6) and a variable frame(7).

MATLAB uses 9 major types of data: Numeric, Characters and Strings, Categorical Arrays, Tables, Structures, Cell Arrays, Function Handles, Map Containers and Times Series. Only those underlined will be intensively used.

All manipulations consist in a set of commands aimed to reach a particular goal. We believe that it is not reasonable to type commands during the tutorial. Instead, each command of the exercises will be explained and the expected result will be described. The user is expected to copy the commands and to paste them into the Command Window. All commands are present into the folder Exercises into files named `Exo*.m`. For instance commands for exercise 1 are into the file `Exo1.m`. For this reason it is recommended to start MATLAB from this folder or to locate this folder immediately. In each file, a line containing only the % character separates commands that should be copy-pasted. Some commands that are too long to be executed during the tutorial are commented out. Instead, the output variables of the commands are loaded into memory. All files containing some of variables are located into the folder `Precomputed`.

Note that command auto-completion is proposed by pressing the tab key.

Goal of the exercise: Read the SVM files of each dopamine subtypes and store the corresponding tasks.

Algorithm: The SVM file format is a sparse encoding used to store molecular descriptors values associated with a scalar property. Each line of the file corresponds to a molecule. A sample line is given below:

```
7.89 1:600 3:104 4:168 5:96 16:4 20:4
```

The first column is the value of the property, here 7.89. The following columns have all the same structure, an integer being the ID of the molecular descriptor, a column character and the value of this descriptor for this molecule. Usually, a companion file gives the link between the ID of the descriptor and its description.

The function reads an SVM file, line by line, fills a matrix of descriptors such that each line correspond to a molecule and each column to a descriptor and fills a column vector which elements are the property values for each molecule. The property vector and the molecular matrix define a task. A data structure shall contain all tasks.

Step-by-step instructions:

`addpath(genpath('<YourPath>/CS3_2014/Tutorials/QSPR_Profiling'));` `addpath(genpath('<YourPath>/MALSAR'));`	The command `genpath` returns a path string to all folders and subfolders below its argument. Then `addpath` uses it to add them to the search path of MATLAB. Now, all the scripts and functions needed in the following exercises are available, including MALSAR. In the `Scripts` folder, you should edit the function `initpath` to use it for reinitializing your search path if it is lost by accident during the exercises. Note that if you mistype your path here, matlab will not complain but the exercises will not work.
`edit('svmlread');`	The editor window opens showing the source code of the function `svmlread`. This function read a file in SVM format and returns `Y`, a vector of property values and the matrix `X` which rows are molecules and columns are descriptors.

The file `svmlread.m` is open into the editor window. A MATLAB script file is executed by typing the name of the file (without `.m`). A file containing a MATLAB function must be have the same name as the function and the first line must contains the function directive. The comment lines (starting with a %) at the beginning of the function are interpreted and are displayed to the user as help message. Note the use of squared braces `[]` to initialize tables/matrices, the use of round braces `()` to access given elements of tables and matrices and the use of the column character : to select sub-matrices.

`[Y,X]=svmlread('Dopamine-D1.svm');`	Read the file `Dopamine-D1.svm` and returns the corresponding matrices `X` and `Y`.
`X=full(X);`	The previous function returns a sparse matrix for `X`, but MALSAR methods need a full matrix representation.

`X=zscore(X);`	The molecular descriptors are normalized.
`X = [X,ones(size(X, 1), 1)];`	A column vector of 1 (`ones( ,1)`) with the same number of rows as `X` (`size(X,1)`) is added to the right of the matrix `X`. This adds a constant to the models.

In fact, we need to store property vectors and molecular descriptors for a defined list of Dopamine families. For the following it is needed to get vectors of tasks as cell arrays: each element of the vector shall contain the property vector or the corresponding molecular descriptors matrix. Each cell of a cell array can accommodate any kind of MATLAB data type: in particular it can contains vectors or matrices.

`aDopa=[1,2,3,4,5];`	A table containing the list of Dopamine files that must be read.
`nDopa=length(aDopa);`	The number of Dopamine files to read.
`X=cell(1,nDopa); Y=cell(1,nDopa);`	`X` and `Y` are now one-line cell arrays with `nDopa` columns.
`for t=1:nDopa`	Loop to parse the `aDopa` table.

`    i=aDopa(t);`
`    [Yl,Xl]=svmlread(strcat('Dopamine-D',int2str(i),'.svm'));`
	The variable `i` is the ID of the dopamine to process. The file name is computed by concatenating strings using `scat`. Then the file is read and molecular descriptors are returned with the corresponding property values.

`    Xl=full(Xl);`
`    Xl=zscore(Xl);`
`    Xl = [Xl ones(size(Xl, 1), 1)];`
	The descriptor matrix is normalized and a constant is added.

`    X{t}=Xl; Y{t}=Yl;`	The current task is added to the `X` and `Y` cell arrays.
`end`	End of the task loop.

5. Exercise 1: Single task learning

Goals of the exercise:

- Build a sparse multi-linear model on a training set
- Apply the model on a test set and measure predictive performances
- Automatize the process for all 5 dopamine targets.

Algorithm:

The main algorithm illustrated here is the LASSO (Least Absolute Selection and Shrinkage Operator) (Tibshirani, 1996) algorithm. It consists in searching a column vector of weight W minimizing the objective function $f_{\{Y,X\}}$ based on the a vector Y containing the target property values and the matrix X, of which each line corresponds to a compound and each column to molecular descriptor :

$$f_{\{Y,X\}}(W) = |Y - XW|_F^2 + \rho_1 |W|_1 \qquad (1)$$

The Frobenius norm is noted $|\cdot|_F$ and the symbol $|\cdot|_1$ indicates the l_1–norm. The parameter ρ_1 controls the regularization term of the objective function and is called either a regularization parameter or a sparsity parameter. In the MATLAB code, the regularization parameter will be systematically noted `p1`.

The l1-norm makes the problem non-differentiable and it is not possible to find a linear algebra expression that solves it. The main interest of the method is that it imposes parsimony to the solution: it acts like an integrated variable selection procedure and limit the number of non-zero parameter weight.

Step-by-step instructions:

If needed run the script `Exo0.m` in order to have the cell arrays `X` and `Y` correctly set with the dopamine activity learning tasks.

`edit('default_opts.m');`	Opens the file `default_opts.m` into the editor window.

The MALSAR (J. Zhou, 2014) methods uses a structured variable called opts that control several aspects of the algorithms. This variable is composed of the following fields:

`opts.max_iter`	Maximum iteration step number of the optimization procedure (default 1000)
`opts.tol`	Tolerance on the optimized value (default 10^{-3})
`opts.tFlag`	Termination condition of the optimization procedure (default 1): 0. Absolute value difference of the optimized value is inferior to the tolerance. 1. Relative absolute value difference of the optimized value is inferior to the tolerance. 2. Absolute value of the optimized value is inferior to the tolerance 3. Run `maxIter` optimization procedure iterations.
`opts.W0`	Starting point of the optimization of the vector of weight W (see opts.init)
`opts.C0`	Starting point of the optimization of the intercept C for Logistic Loss (see opts.init)
`opts.init`	Specifies how to initialize W or C before the optimization procedure starts (default 2): 0. Uses guess values inferred from the data. 1. Uses the user defined values `opts.W0` or `opts.C0`. This is useful to restart a run. 2. Set the `W0` to a 0 vector and `C0` to 0.
`opts.pFlag`	Need the MATLAB parallel Toolbox. Enable Map-Reduce (default False)
`opts.pSeg_num`	Need the MATLAB parallel Toolbox and not yet available. Set the number of total parallel segmentations.

In this tutorial, the optimizations are performed until the objective function value changes less than 0.0001. This setting will produce relatively fast computations, but the convergence might be a bit weak. However, all results can be reproduced with much more stringent and computationally demanding optimization parameters.

`opts=default_opts;`	Optimization settings for MALSAR.

`prc=0.3;`
`[Xtr, Ytr, Xte, Yte]=mtlSplit(X, Y, prc);`
	Divide the whole dataset into a training set (2/3 of the dataset) and a test set (1/3 of the dataset. The proportion of instances into the test set is controlled by the variable `prc`.

`p1=5;`
`[W,funcval]=Least_Lasso(Xtr(1,1), Ytr(1,1),p1,opts);`
	Build a LASSO model for dopamine D1 on training set data using a parameter value `p1=5`. Models' weights are into the vector `W`. The vector `funcval` contains

```
Yp=Xte{1,1}*W;
SSE=sum((Yp-Yte{1,1}).^2);
SST=sum((Yte{1,1}-mean(Yte{1,1})).^2);
R2=1-SSE/SST;
RMSE=sqrt(SSE/size(Yte{1,1},1));
fprintf('RMSE=%g R2=%g\n',RMSE,R2);
```

the values of the optimization function.
Apply the model on the external test set.
Compute the sum of squared errors (`SSE`), the sum of square distances to the mean (`SST`), the determination coefficient (`R2`) and the Root Mean Squared Error (`RMSE`)

The observed performances at this stage should be the following:

`RMSE=0.66; R2=0.43`

```
plot(Yp,Yte{1,1},'o',Yte{1,1},
Yte{1,1},'-');
xlabel('Estimates');
ylabel('Experiment');
xlim([7,10]);
ylim([7,10]);
```

Plot predicted values as a function of experimental values.

This first plot is shown in Figure 2.

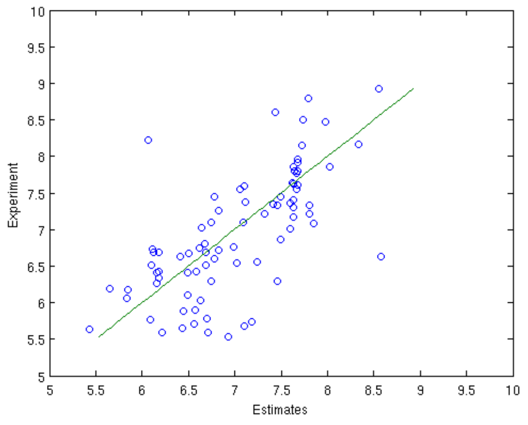


Figure 2. Experimental vs estimated values. The LASSO algorithm for Dopamine D1 and a parameter value of 5 have been used.

```
STL=cell(3,nDopa);
STL_W=cell(1,nDopa);


for t=1:nDopa
  i=aDopa(t);
  fprintf('*** Dopamine D%i\n',i);
  [STL_W{1,t},funcval]=
Least_Lasso(Xtr(1,t),Ytr(1,t),p1,opts)
;
  [STL{1,t},STL{2,t},STL{3,t}]=
stlRMSE(Xte{1,t},Yte{1,t},SLT_W{1,t});
  fprintf('RMSE=%g
R2=%g\n',STL{1,t},STL{2,t});
```

Automation of the above steps for all dopamine receptors requires storing per task statistics (`STL`) and models (`STL_W`) in cell arrays.
For each task, compute a LASSO model, using always the same parameter value (`p1=0.01`), store the model into the corresponding cell array (`STL_W`), use it and the corresponding test set to compute performances measures and store them into another cell array (`STL` stores in this order RMSE, R2 and estimates). Finally, plot the estimates

```
  figure;
  plotExpPred(Yte{1,t},STL{3,t},
strcat('Dopamine D',int2str(i)));
end
```

against the experimental values. The command `figure` makes the plot permanent.

Five new plots should appear and the corresponding performances should be displayed:

Dopamine	**RMSE**	**R2**
D1	0.66	0.43
D2	0.76	0.55
D3	0.82	0.59
D4	0.70	0.60
D5	1.08	0.23

Table 2. Typical performances of STL LASSO models on Dopamine D1 to D5 using a parameter value p=5.

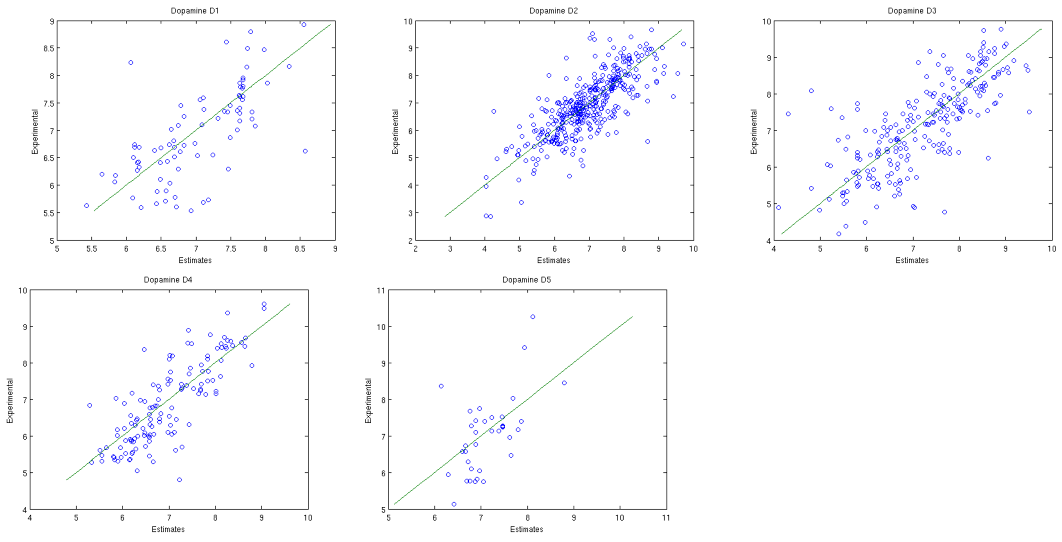


Figure 3. Typical experimental versus predicted plots for Dopamine D1 to D5 of STL LASSO models on a test set. The parameter p=5 is used.

6. Exercise 2: Multi-task learning introducing sparcity

Goal of the exercise:

- Build simultaneously all sparse multi-linear model for all dopamine regression tasks
- Apply the models on a test set and measure predictive performances
- Introduce performances measures that are relevant in the multi-task learning context

Learn all dopamine regression tasks simultaneously and present performances statistics that are relevant in the context of multi-task learning.

Algorithm:

In this exercise the same LASSO algorithm will be used. The objective function $f_{\{Y,X\}}$ based on the a vector Y containing the target property values and the matrix X is given by equation (1). However, this time W is a matrix: each column refers to a task and each line to the molecular descriptors' coefficients. The l_1-norm is a function of the matrix elements W_j^i, where *i* is row index and *j* is a column index. It is computed as follows:

$$|W|_1 = \sum_{i,j} |W_j^i| \qquad (2)$$

In this formulation, the coupling between tasks is weak because the only assumption is that all tasks share the same parsimony parameter.

Set by step instructions:

It is assumed that the previous exercises were done. If it is not the case, run the scripts `Exo0.m` and `Exo1.m`. Then, the regression tasks will be stored into the cell arrays `X` and `Y` and STL statistics will be computed for all tasks. Also, the method parameter `p1` and optimization parameters will be set.

```
[W,funcval]=Least_Lasso(Xtr,Ytr,p1
,opts);
```

Compute all models on the training set.

```
MTL_Lasso=cell(3,nDopa);
```

Create a cell array to perform an STL-like analysis of the models

```
for t=1:nDopa
  i=aDopa(t);
  fprintf('****           Dopamine
d%g\n',i);
  [RMSE,R2,Yp]=
stlRMSE(Xte{1,t},Yte{1,t},W(:,t));
  MTL_Lasso{1,t}=RMSE;
  MTL_Lasso{2,t}=R2;
  MTL_Lasso{3,t}=Yp;
  fprintf('RMSE=%g
R2=%g\n',RMSE,R2);
  figure;
  plotExpPred(Yte{1,t},Yp,
strcat('MTL     Lasso     Dopamine
D',int2str(i)));
end
```

For each task, compute the performances, store them and plot the experimental values as a function of the estimates.

The performances of the STL and MTL models are virtually identical (see **Table 2** and **Table 3**).

Dopamine	RMSE	R2
D1	0.66	0.43
D2	0.76	0.55
D3	0.82	0.59
D4	0.69	0.60
D5	1.03	0.29

Table 3. Typical performances of MTL LASSO models on Dopamine D1 to D5 using a parameter value p=5 on a test set.

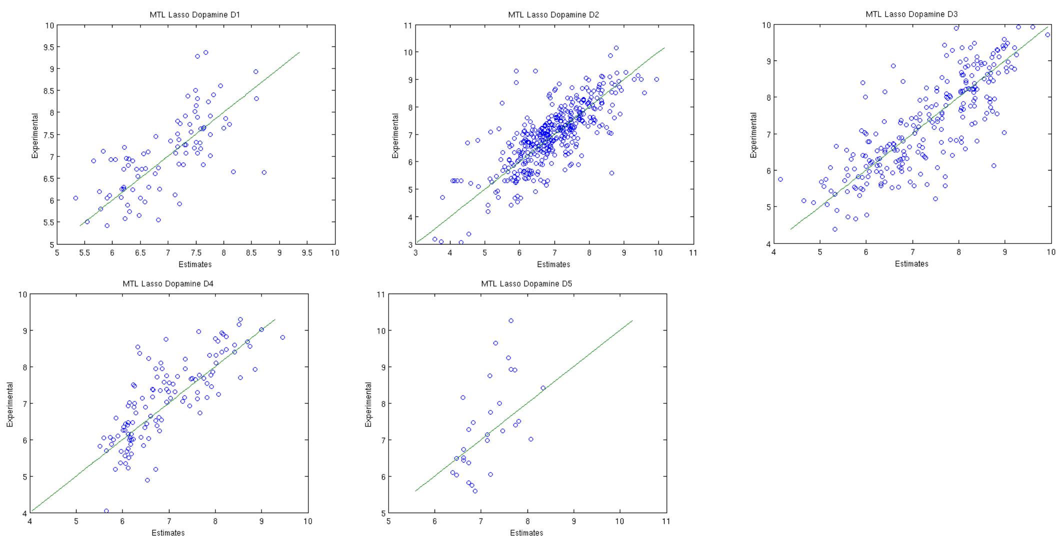


Figure 4. Typical experimental versus predicted plots for Dopamine D1 to D5 MTL LASSO models on a test set. The parameter value used is p=5.

```
v1=transpose(cell2mat(STL(2,:)));
v2=transpose(cell2mat(MTL_Lasso(2,
:)));
lDopa=transpose(arrayfun(@(x)
strcat(‘D’,   int2str(x)),   aDopa,
’UniformOutput’, false));
bar([v1,v2]);
set(gca,’xTickLabel’,lDopa);
title(‘STL  compared  to  LASSO  MTL
based on R2’);
legend(‘STL’,’MTL’);
```

Transform the models’ performances into a column vector for STL models (v1), for the MTL models (v2) and compute a vector of labels each element referring to a dopamine subfamily.

Display a barplot comparing the STL and the MTL method with appropriate labels.

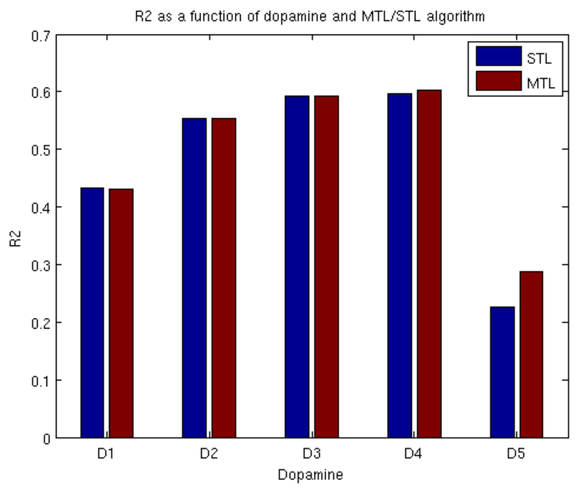


Figure 5. Comparison of R2 values for each dopamine D1 to D5 between STL models (blue) and MTL models (red).

However, the MTL procedure or the sequential building of STL models generates an array of results that are difficult to manage: there are as many plots, RMSE, R2 values, etc. as the number of tasks (for instance **Figure 3** and **Figure 4**). A common attempt to sum up the results is to use bar plots as in this example (**Figure 5**). This is adequate to manage up to about 10 tasks, but it is rapidly impossible to analyze when there are more of them.
Therefore, there it is common to find two common average performance indicators. The first one is the mean root mean squared error, $\langle RMSE \rangle_T$:

$$\langle RMSE \rangle_T = \frac{1}{T}\sum_{t=1}^{T} RMSE_t \tag{3}$$

It is an average over the number of tasks T of the per task root mean squared error $RMSE_t$.
The second one is a weighted average of the root of mean squared residual per task, according to the number of instances per tasks and the total number of instances. This indicator, $\langle RMSE \rangle_{N_t}$, is defined as follow:

$$\langle RMSE \rangle_{N_t} = \frac{\sum_{t=1}^{T} N_t . RMSE_t}{\sum_{t=1}^{T} N_t} \tag{4}$$

It is worth to remark that the $\langle RMSE \rangle_T$ is giving an equal weight to all tasks while the $\langle RMSE \rangle_{N_t}$ is giving more importance to the most populated task.

```
mRMSE=mean(cell2mat(MTL_Lasso(1,:)
));
wRMSE=0; Nall=0;
for t=1:nDopa
  Nt=size(Yte{1,t},1);
  wRMSE=wRMSE+MTL_Lasso{1,t}*Nt;
  Nall=Nall+Nt;
End
wRMSE=wRMSE/Nall;
fprintf(‘MTL: mean RMSE=%g; Wght.
Mean RMSE=%g\n’,mRMSE,wRMSE);
```

Compute the $\langle RMSE \rangle_T$ for the MTL algorithm.

Compute the $\langle RMSE \rangle_{N_t}$ and display the line of report for the MTL algorithm.

The averaged performances shall look as:

```
MTL Lasso. Mean RMSE=0.79; Wght. Mean RMSE=0.77
```

Of course, given the same number of STL models on the same targets, it is straightforward to compute a mean RMSE ($\langle RMSE \rangle_T$) and a weighted mean RMSE ($\langle RMSE \rangle_{N_t}$).

```
mRMSE=mean(cell2mat(STL(1,:)));
wRMSE=0; Nall=0;
for t=1:nDopa
  Nt=size(Yte{1,t},1);
  wRMSE=wRMSE+STL{1,t}*Nt;
  Nall=Nall+Nt;
End
wRMSE=wRMSE/Nall;
fprintf(‘STL: mean RMSE=%g; Wght.
Mean RMSE=%g\n’, mRMSE, wRMSE);
```

Compute the $\langle RMSE \rangle_T$ for the STL algorithm.
Compute the $\langle RMSE \rangle_{N_t}$ and display the line of report for the STL algorithm.

The averaged performances of the STL models shall look as:

```
STL Lasso. Mean RMSE=0.80; Wght. Mean RMSE=0.77
```

The two measures of averaged root mean squared error are close. Naturally they confirm the empirical observations that the MTL algorithm is equivalent to the sequential use of STL algorithm.

7. Exercise 3: Optimizing one parameter

Goal of the Exercise:

- Set up a cross validation procedure
- Study the influence of the parameter of the algorithm on the generalization of the model
- Identify over-fitting and under-fitting
- Increase the coupling between the learning tasks into the MTL algorithm

Algorithm:

The cross-validation procedure presented here consists in (*i*) shuffling the dataset, (*ii*) dividing it into N_f subsets and (*iii*) in turn using one of them as test set and the others as training set. The whole procedure is repeated N_k times. The performances statistics are computed on each fold and the averaged values on all folds are finally considered.

The LASSO algorithm uses a regularization parameter ρ_1 controlling the balance between the reduction of the fitting error and the parsimony of the solution. Several values of the parameter are investigated and for each one, the cross-validation performances are measured (as described above). These performances are plotted as a function of the parameter ρ_1. For each value of the parameter the number of zeros into the vector W solution is plotted. Since the LASSO algorithm generates sparse solutions, the number of zero is a good indicator of the parsimony of the model. In this context a non-parsimonious model is over-fitted while a too parsimonious one is clearly under-fitted.

Step by step instructions:

It is assumed that the previous exercises were done. If it is not the case, run the scripts `Exo0.m` and `Exo1.m`. Then, the regression tasks will be stored into the cell arrays `X` and `Y` and STL statistics will be computed for all tasks. Also, the method parameter `p1` and optimization parameters will be set.

```
edit('MTL_CV1P.m');
Nk=3; Nf=3;
t=cputime;
[rslt,mrslt]=MTL_CV1P(X,Y,
'Least_Lasso',p1,opts,Nf,Nk);
e=cputime-t;
fprintf('MTL     elapsed     time
%g\n',e);
```

Set up a 3-fold cross-validation procedure repeated 3 times. The detailed per fold statistics are store into the variable `rslt` ($\langle RMSE\rangle_T$, $\langle RMSE\rangle_{N_t}$, per task RMSE, per task R2 and per task estimates), and averaged results are located into the variable `mrslt` ($\langle RMSE\rangle_T$, $\langle RMSE\rangle_{N_t}$, per task RMSE). The command cputime return the time in seconds since MATLAB was started. It is used to estimate the elapsed CPU time used by the MTL algorithm.

```
STL=cell(1,nDopa);
t=cputime;
for t=1:nDopa
  [rslt,STL{1,t}]=
MTL_CV1P(X(1,t),Y(1,t),
'Least_Lasso',p1,opts,Nf,Nk);
end
e=cputime-t;
fprintf('Serial  STL  elapsed  time
%g\n',e);
```

The same cross-validation procedure is applied on STL models. Performances for each task are stored in a separate cell of the `STL` cell array. The elapsed CPU time is computed for comparison with MTL

Typically, the CPU times for generating the LASSO MTL model are much smaller than those for the same number of STL models.

```
v1=transpose(mrslt{3});
v2=[];
for t=1:nDopa
  v2=[v2; cell2mat(STL{t}(3))];
end
```

Per task RMSE is transferred to two vectors: `v1` for MTL and `v2` for STL. They are used with the label vector `lDopa` to generate a bar plot in order to compare the performances of

```
figure;
bar([v1,v2]);
set(gca,'XTickLabel',lDopa);
legend('MTL','STL',2);
title('Cross-validated RMSE of MTL
and STL Lasso models with p1=5');
```

both algorithm.

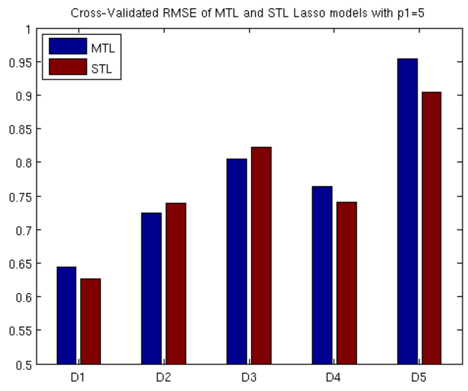


Figure 6. Comparison of 3 times 3-fold cross-validated RMSE of MTL models and the corresponding STL models on all five Dopamine subfamilies.

The cross validation confirms that the equivalence of the MTL and STL algorithm performances is a robust observation (
Figure **6**). The cross-validation procedure is certainly more demanding computationally but the equivalence between the STL and the MTL models is better established.

Once a benchmarking procedure is setup, it is time to estimate the influence of the algorithm's parameter ρ_1 on the performances of the models. The main performance criteria that will be followed it the cross-validated RMSE averaged per fold ($\langle RMSE \rangle_{CV}$) and the tasks averaged RMSE, the $\langle RMSE \rangle_T$.

```
ap1=linspace(1,100,25);
MTL_Lasso=cell(1,length(ap1));
for i=1:length(ap1)
  p1=ap1(i);
  [rslt,mrslt]=MTL_CV1P(X,Y,
'Least_Lasso',p1,opts,Nk,Nf);
  MTL_Lasso{i}=mrslt;
  fprintf('p1=%g; mean RMSE=%g; Wght.
mean  RMSE=%g  \n',  p1,  mrslt{1},
mrslt{2});
end
v3=[];
for t=1:length(ap1)
  v3= [v3,cell2mat(MTL_Lasso{t}(1))];
end
plot(ap1,v3);
title('Mean  RMSE  as  a  function  of
regularization parameter value');
xlabel('Parameter');
ylabel('Mean RMSE');
```

A linear sampling, `ap1`, of the interval [1:100] is used a values for the parameter `p1` of the algorithm. For each value, the 3 times 3 fold cross validation procedure is applied and the results are recorded into the cell array `MTL_Lasso`, each cell corresponding to on value of `p1`.
<u>This step is lengthy. Instead of running these lines, please load the pre-computed results: `Exo3_MTL_Lasso.mat`.</u>

Extract into a vector `v3`, the $\langle \mathrm{RMSE} \rangle_{\mathrm{T}}$ for each value of the parameter `p1`. Each component of `v3` corresponds to a sampled value of the parameter.
Use the vector `ap1` and the vector `v3` to plot the dependence of the overall models performances as a function of the parameter.

The plot of the performances of the MTL models as a function of the parameter shall look as on **Figure 7**.

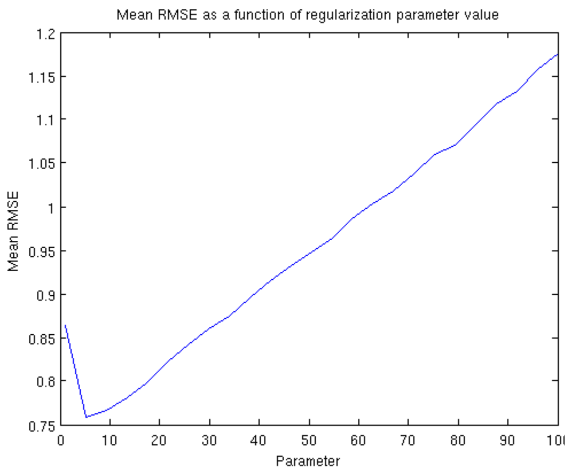


Figure 7. Cross-validated (3 times 3-fold) performances ($\langle RMSE \rangle_T$) of the dopamine MTL models, as a function of the parameter ρ_1 of the algorithm.

```
v4=[];
for t=1:50
  v4=[v4; cell2mat(MTL_Lasso{t}(3))];
end
plot(ap1,v4)
title('Per   dopamine   RMSE   as   a
function of regularization parameter
value');
xlabel('Parameter');
ylabel('RMSE');
legend(lDopa);
```

Extract into a matrix `v4`, the fold averaged $\langle \mathrm{RMSE} \rangle_{\mathrm{CV}}$ values for each value of the parameter `p1`. Use the vector `ap1` and the matrix `v4` to plot the dependence of the each model's performances as a function of the parameter.

The plots (**Figure 8**) shall evidence that the optimal value for the parameter ρ_1 for each dopamine subfamily model, are localized grossly in the same range.

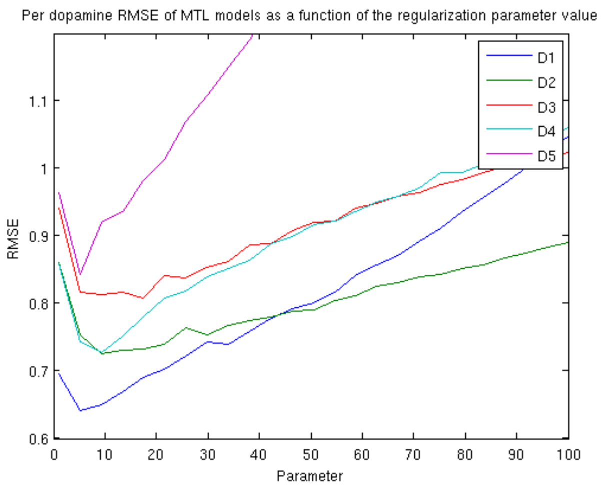


Figure 8. Cross-validated (3 times 3-fold) performances (RMSE) of the individual dopamine MTL models, as a function of the parameter ρ_1 of the algorithm.

```
sparsity=zeros(length(ap1));
for i=1:length(ap1)
```

For each value of the parameter `p1`, build a model using the whole dataset, compute

```
  p1=ap1(i);
  [W,funval]=Least_Lasso(X,Y,
p1,opts);
  sparsity(i)=nnz(W);
  fprintf('p1=%g        sparsity=%g
\n',p1,sparsity(i));
end;
```

the number of non-zero element (using the function `nnz`) in the weight matrix `W` and store it into the vector `sparsity`.

```
plot(ap1,sparsity)
title('Models sparsity as a function
of regularization parameter value');
xlabel('Parameter');
ylabel('Number of non-zero weights');
```

Plot the vector `sparsity` as a function of the vector of parameter value `ap1`.

The parsimony graph (**Figure 9**) illustrate how qualitatively evolve the solution with the values of the parameter p1 of the method.

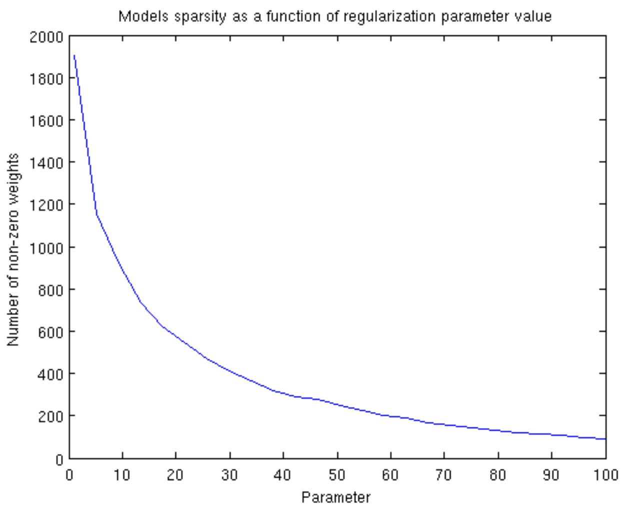


Figure 9. Number of non-zero elements in the solution matrix `W` as a function of the parameter ρ_1 value of the algorithm.

All RMSE curves have a familiar aspect: at one end of the parameter range, models are over-fitted and at the other, models are under-fitted. For these values, they generalize poorly. In between, the cross-validation RMSEs are minimal locating the optimal model considering the current dataset and algorithm.

A minimum of the mean RMSE is observed about ρ_1=5 and the optimal parameter value is about the same order across all dopamine tasks (**Figure 7**, **Figure 8**). However, a closer look shows that each task has its own optimum and therefore using one value for the parameter can be beneficial to some task and detrimental to the others. Therefore, the MTL algorithm is necessarily a compromise between all tasks. The performance measures of MTL models are somehow defining this compromise.

Finally, the sparsity analysis as a function of the parameter value (**Figure 9**) clearly rationalizes the interpretation of the RMSE curves. For small values of the parameter the weight matrix is non-sparse indicating over-fitted models. For large values of the parameter, the weight matrix is almost zero, thus indicating under-fitted models.

Finally, the regular shape of the RMSE curve with one shallow minimum means that a simple dichotomy algorithm is sufficient to optimize the algorithm's parameter.

Note: The different steps of this exercise can be repeated in the context of STL models for the sake of comparison. This is the object of the file `Exo3b.m`. This leads to very similar dependences of the models' performances with the parameter value (**Figure 10**). The corresponding pre-computed variable can be loaded using the file `Exo3b_sSTL.mat` into the folder `Precomputed`.

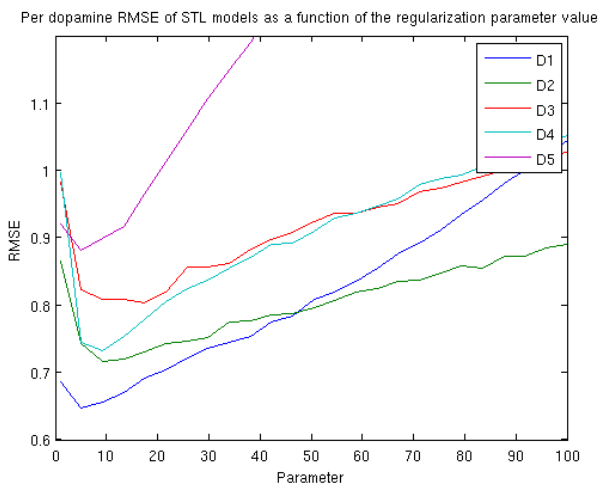


Figure 10. Cross-validated (3 times 3-fold) RMSE of the STL models for each dopamine subfamily, as a function of the parameter $\boldsymbol{\rho_1}$.

8. Exercise 4: The L21 algorithm

Goal of the exercise:

- Introduce another modification of the LASSO algorithm with strong coupling between tasks, the MTL L21 algorithm (Argyriou, Evgeniou, & Pontil, 2007)
- Illustrate a typical solution
- Compare this MTL algorithm to the MTL LASSO algorithm.

Algorithm:

Another objective function is introduced in order to improve the coupling between the tasks being learned. This function is a modification of the LASSO equation (1), based on the l_{21}-norm:

$$g_{\{Y,X\}}(W) = |Y - XW|_F^2 + \rho_{21}|W|_{21} \tag{5}$$

The l_{21}-norm is computed as follows:

$$|W|_{21} = \sum_{i=1}^{N_D} \sqrt{\sum_{j=1}^{T} {W_j^i}^2} \tag{6}$$

In the equation (6): rows are indexed by the letter i and refer to the N_D molecular descriptors; columns are indexed by the letter j and refer to the T tasks. The main characteristic of this objective function is that it leads to models using a joint set of variables across tasks. In other words, all tasks in this MTL formulation use a common subset of molecular descriptors.

Step by step instructions:

```
MTL_L21=cell(1,length(ap1));
for i=1:length(ap1)
  p1=ap1(i);
  [rslt,mrslt]=MTL_CV1P(X,Y,
'Least_L21',p1,opts,Nk,Nf);
  MTL_L21{i}=mrslt;
  fprintf('p1=%g;   mean   RMSE=%g;
Wght.    mean    RMSE=%g\n',    p1,
mrslt{1}, mrslt{2});
end
```

For each sampled value of the parameter p1, measure the cross-validated performances of the `Least_L21` algorithm and store them into the cell array `MTL_L12`.
<u>This step is lengthy. Instead of running these lines, please load the pre-computed results: `Exo4_MTL_L21.mat`.</u>

```
v5=[];
for t=1:length(ap1)
```

Store into the vector `v5` the task averaged $\langle RMSE\rangle_T$ values for each parameter value.

```
  v5=[v5,
cell2mat(MTL_L21{t}(1))];
end
v6=[];
for t=1:length(ap1)
  v6=[v6;
cell2mat(MTL_L21{t}(3))];
end
```

Store on each row of the matrix `v6`, the RMSE for each task corresponding to the same parameter value. Each column corresponds to the RMSE of a particular task.

```
plot(ap1,v5);
title('Mean RMSE as a function of
regularization parameter value');
xlabel('Parameter');
ylabel('Mean RMSE');
```

Plot the dependence of the mean RMSE to the parameter value.

```
plot(ap1,v6)
title('Per dopamine RMSE as a
function of regularization
parameter value');
xlabel('Parameter');
ylabel('RMSE');
legend(lDopa);
```

Plot for each task, the RMSE as a function of the parameter value.

These commands shall lead to the following plots (**Figure 11** and **Figure 12**).

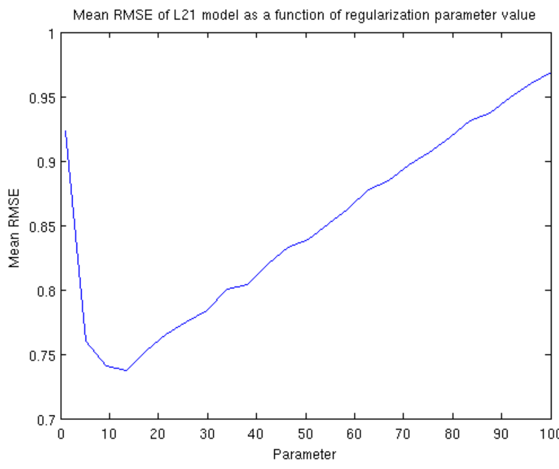


Figure 11. Cross-validated (3 times 3-fold) task averaged RMSE ($\langle \boldsymbol{RMSE} \rangle_{\boldsymbol{T}}$) as a function of the parameter $\boldsymbol{\rho_{21}}$ of the MTL $\boldsymbol{l_{21}}$-norm algorithm.

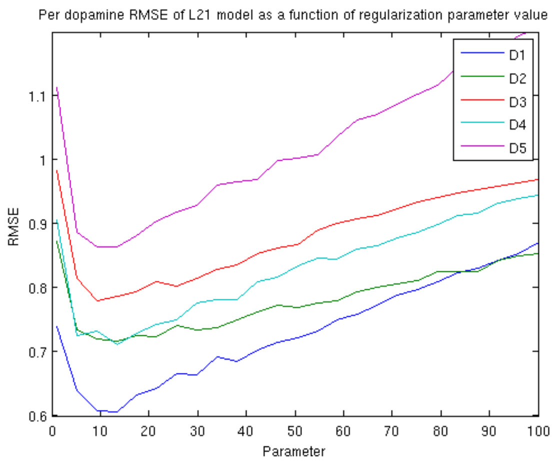


Figure 12. Cross-validated (3 times 3-fold) RMSE for individual dopamine subfamily task as a function of the parameter $\boldsymbol{\rho_{21}}$ of the MTL $\boldsymbol{l_{21}}$-norm algorithm.

```
sparsity_L21=zeros(length(ap1));
for i=1:lenth(ap1)
  p1=ap1(i);
  [W,funcval]=Least_L21(X,Y
,p1,opts);
  sparsity_L21(i)=nnz(sum(W,2));
end
figure;
plot(ap1,sparsity_L21);
title('Sparcity of L21 MTL
models');
```

For each parameter value `p1,` compute a L21 MTL model and count the number of non-zero rows into the weight matrix `W`.
Store the value into the vector `sparsity_L21`.
Then plot the sparsity as a function of the parameter of the algorithm.

```
xlabel('Parameter');
ylabel('Number of non-zero rows');
```

This creates the following plot (**Figure 13**).

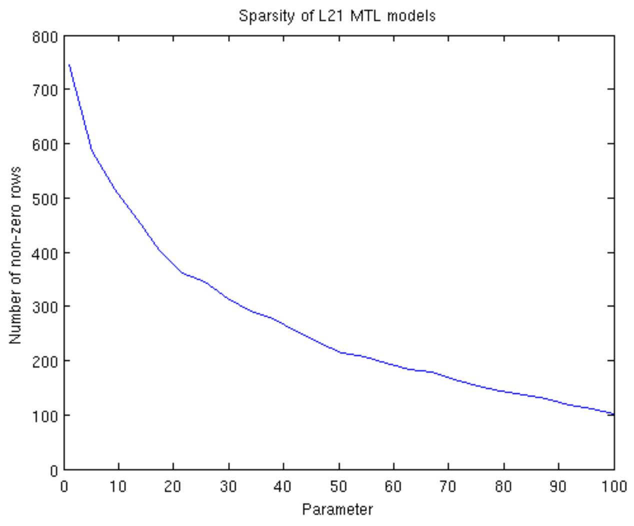


Figure 13. Number of non-zero rows into the vector $\boldsymbol{W}$ solution of the MTL $\boldsymbol{l_{21}}$-norm algorithm for different values of the parameter $\boldsymbol{\rho_{21}}$ of the algorithm.

The main characteristic of the matrix W solution of the l_{21}-norm problems is that entire rows are null. As demonstrated by the sparsity analysis, the larger value is the parameter, the more rows are zeros, and the model tends to be under-fitted (**Figure 13**). Reversely, if the parameter value is small, almost no row of the matrix W is zero, and the model is over-fitted.

Another observation: comparing to the STL LASSO models, the MTL LASSO formulation, lead to weak improvement in the models performances. The MTL l_{21}-norm formulation provides a more contrasted picture: more interesting improvement are possible (see for instance D5 or D3 in **Figure 10**, **Figure 8** and **Figure 12**). Some tasks tend to benefit more than the others of the MTL algorithm. Finally, the optimum value of the ρ_{21} parameter seems less dependent of the particular dopamine subfamily than the parameter ρ_1 of the MTL LASSO algorithm.

9. Exercise 5: Interactions between tasks

Goal of the exericise:

- Optimize a parameter using a dichotomy algorithm
- Measure optimal performances of L21 MTL algorithm while learning all dopamine tasks simultaneously
- Measure optimal performances of L21 MTL algorithm while learning D1, D5 tasks simultaneously
- Measure optimal performances of L21 MTL algorithm while learning D2, D3 and D4 tasks simultaneously

Algorithm:

Optimizing the algorithm parameter is necessary in order to compare the performances of different MTL (or STL) algorithm. A dichotomy procedure is proposed: it consists in searching a minimum of a function into a range, by recursively dividing the range search by two. In the current exercises, a range containing the optimal value for the parameter of an algorithm (leading to the lowest RMSE) is grossly known. The cross-validated RMSE is valued at the minim, 1/4th, half, 3/4th and maximum of the range. A new range is defined using the division immediately before and immediately after the one that has the lowest RMSE value. Thus, 5 evaluations are needed to initialize the algorithm and then 2 more evaluations at each step are added.

The dichotomy algorithm is very well suited in this case because the parameter to optimize has a unique shallow minimum yet the first derivative is too noisy to be estimated. Considering that the curves of RMSE as a function of the parameter are shallow, if the optimal value is known at about two orders of magnitude, only 5 iterations will be sufficient to get a sufficient estimation of the optimal parameter, thus requiring at maximum 10 evaluation of the RMSE.

In the current exercise, the mean RMSE will be optimized except for STL algorithm where the RMSE of each task will be optimized individually.

Step by step instructions:

```
pmin=1;
pmax=50;
Nf=3;
Nk=3;
maxiter=3;
```

Initialization of the optimization procedure, setting the minimal and maximal value for the parameter, the number of fold and the number of iterations

```
[p_Lasso,val_Lasso]=DichotomyOptim
ize( X, Y, 'Least_Lasso', opts,
pmin, pmax, Nf ,Nk, maxiter);
[p_L21,val_L21]=DichotomyOptimize(
X, Y, 'Least_L21', opts, pmin,
pmax, Nf, Nk, maxiter);
```

Search an optimal value for the MTL LASSO and MTL L21 algorithm on all tasks simultaneously.

```
p_STL=cell(2,nDopa);
for t=1:nDopa

[p_STL{1,t},p_STL{2,t}]=DichotomyO
ptimize( X(1,t), Y(1,t),
'Least_Lasso', opts, pmin, pmax,
Nf, Nk, maxiter);
end;
```

Sequential optimization of the parameter of the STL LASSO algorithm for each task.

```
X15=cell(1,2);
X15(1,1)=X(1,1); X15(1,2)=X(1,5);
X234=cell(1,3);
X234(1,1)=X(1,2);
X234(1,2)=X(1,3);
X234(1,3)=X(1,4);
%
Y15=cell(1,2);
Y15(1,1)=Y(1,1); Y15(1,2)=Y(1,5);
Y234=cell(1,3);
Y234(1,1)=Y(1,2);
Y234(1,2)=Y(1,3);
Y234(1,3)=Y(1,4);
```

Prepare new datasets restricted to dopamine D1 and D5 on the one hand, and to Dopamine D2, D3 and D4 on the other hand.

```
[p_Lasso_15,val_Lasso_15]=Dichotom
yOptimize( X15, Y15,
'Least_Lasso', opts, pmin, pmax,
Nf, Nk, maxiter);
[p_L21_15,val_L21_15]=DichotomyOpt
imize( X15, Y15, 'Least_L21',
opts, pmin, pmax, Nf, Nk,
maxiter);
%
[p_Lasso_234,val_Lasso_234]=Dichot
```

Optimize the parameter for MTL LASSO and MTL L21 algorithms on MTL datasets restricted to dopamine D1 and D5, then restricted to dopamine D2, D3 and D4

```
omyOptimize(       X234,       Y234,
'Least_Lasso',  opts,  pmin,  pmax,
Nf, Nk, maxiter);
[p_L21_234,val_Lasso_234]=Dichotom
yOptimize(       X234,        Y234,
'Least_L21', opts, pmin, pmax, Nf,
Nk, maxiter);
```

Compute cross-validated performances using the optimal parameter value previously found on each MTL case.

```
[rslt,m_Lasso]=MTL_CV1P(    X,    Y,
'Least_Lasso',  p_Lasso,  opts,  Nf,
Nk);
[rslt,m_L21]=MTL_CV1P(     X,     Y,
'Least_L21', p_L21, opts, Nf, Nk);
[rslt,m_Lasso_15]=MTL_CV1P(    X15,
Y15,   'Least_Lasso',   p_Lasso_15,
opts, Nf, Nk);
[rslt,m_L21_15]=MTL_CV1P(      X15,
Y15,  'Least_L21',  p_L21_15,  opts,
Nf, Nk);
[rslt,m_Lasso_234]=MTL_CV1P(  X234,
Y234,  'Least_Lasso',  p_Lasso_234,
opts, Nf, Nk);
[rslt,m_L21_234]=MTL_CV1P(    X234,
Y234,    'Least_L21',    p_L21_234,
opts, Nf, Nk);
```

Compute averaged RMSE for the STL models

```
m_STL=cell(1,nDopa);
for t=1:nDopa
  [rslt,m_STL{1,t}]=MTL_CV1P(
X(1,t),   Y(1,t),   'Least_Lasso',
p_STL{1,t}, opts, Nf, Nk);
end;
```

Reshape all statistical results into vectors that can be ploted.

```
aa=[];
for t=1:T
  aa=[aa,
cell2mat(m_STL{1,t}(1))];
end;
mRMSE=mean(aa);
wRMSE=0;
Nall=0;
for t=1:nDopa
  Nt=size(Yte{1,t},1);
  wRMSE=wRMSE+aa(t)*Nt;
  Nall=Nall+Nt;
end
wRMSE=wRMSE/Nall;
aa=[aa,mRMSE,wRMSE];
%
aa_Lasso=[m_Lasso{1,3},
m_Lasso{1,1}, m_Lasso{1,2}];
aa_L21=[m_L21{1,3},     m_L21{1,1},
m_L21{1,2}];
%
bb15=m_Lasso_15{1,3};
```

```
bb234=m_Lasso_234{1,3};
aa_Lasso_15_234=[bb15(1),
bb234(1),    bb234(2),    bb234(3),
bb15(2)];
mRMSE=mean(aa_Lasso_15_234);
wRMSE=0;
Nall=0;
for t=1:nDopa
  Nt=size(Yte{1,t},1);
  wRMSE=wRMSE+
aa_Lasso_15_234(t)*Nt;
  Nall=Nall+Nt;
end
wRMSE=wRMSE/Nall;
aa_Lasso_15_234=[aa_Lasso_15_234,
mRMSE,wRMSE];

bb15=m_L21_15{1,3};
bb234=m_L21_234{1,3};
aa_L21_15_234=[bb15(1),   bb234(1),
bb234(2), bb234(3), bb15(2)];
mRMSE=mean(aa_L21_15_234);
wRMSE=0;
Nall=0;
for t=1:nDopa
  Nt=size(Yte{1,t},1);
  wRMSE=wRMSE+
aa_L21_15_234(t)*Nt;
  Nall=Nall+Nt;
end
wRMSE=wRMSE/Nall;
aa_L21_15_234=[aa_L21_15_234,
mRMSE, wRMSE];
bar(transpose([    aa;    aa_Lasso;
aa_L21;            aa_Lasso_15_234;
aa_L21_15_234]));
lBars=[lDopa; 'mRMSE'; 'wRMSE'];
set(gca,'XTickLabel',lBars);
title('Models  RMSE  as  a  function
of the algorithm');
xlabel('Dopamine  and  global  RMSE
measure');
ylabel('RMSE');
legend('STL',  'MTL  Lasso',  'MTL
L21',  'MTL  Lasso  D1-D5/D2-D3-D4',
'MTL L21 D1-D5/D2-D3-D4');
ylim([0.6,1]);
```

Plot RMSE of models using different algorithm and MTL subsets and the corresponding averaged RMSE.

The statistics are summarized into the plot (**Figure 14**) generated by this exercise.

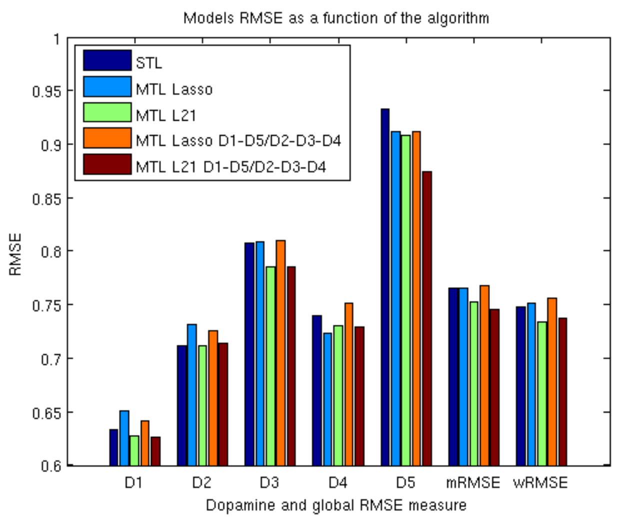


Figure 14. Summary of the benchmark of different MTL and STL setup. For each dopamine task, the RMSE is given. The mRMSE stands for the $\langle \boldsymbol{RMSE} \rangle_{\boldsymbol{T}}$ and the wRMSE stands for $\langle \boldsymbol{RMSE} \rangle_{\boldsymbol{N_T}}$. The orange and red bar are aggregating results for MTL restricted to D1, D5 on the one hand and to D2, D3 and D4 on the other hand. The aggregated results are used to compute the corresponding $\langle \boldsymbol{RMSE} \rangle_{\boldsymbol{T}}$ and . $\langle \boldsymbol{RMSE} \rangle_{\boldsymbol{N_T}}$

	Parameter	D1	D2	D3	D4	D5	$\langle RMSE \rangle_T$	$\langle RMSE \rangle_{N_T}$
	D1: 5.6							
	D2: 10.2							
STL	D3: 11.7	0.64	0.71	0.81	0.74	0.93	0.76	0.75
	D4: 10.2							
	D5: 4.1							
MTL LASSO	5.6	0.65	0.73	0.81	0.72	0.91	0.77	0.75
MTL L21	19.4	0.63	0.71	0.79	0.73	0.91	0.76	0.73
MTL LASSO D1, D5	4.1	0.64				0.91	0.76	0.76
MTL LASSO D2, D3, D4	11.7		0.73	0.81	0.75			
MTL L21 D1, D5	7.1	0.62				0.87	0.75	0.74
MTL L21 D2, D3, D4	11.7		0.71	0.78	0.73			

Table 4. Summary of the performances of STL and MTL methods through RMSE per dopamine task, $\langle \boldsymbol{RMSE} \rangle_{\boldsymbol{T}}$ and $\langle \boldsymbol{RMSE} \rangle_{\boldsymbol{N_T}}$. The optimal value of the parameter of each method is also given.

There are three observations to do on the results reported (**Figure 14** and **Table 4**). First, the Lasso MTL and STL are equivalent while the l_{21}-norm MTL is in general slightly advantageous. Second, for dopamine D5 the MTL algorithms are beneficial. Third, all things being equal, the pool of tasks being learned simultaneously have an impact on the final performances of the models. For instance, the L21 MTL model solely trained on the D1 and D5 receptor, is better for D5 than the corresponding single task model. Note that D5 is the most difficult task because the D5 dataset contains only 98 instances.

In this case MTL was reframed according to a biological concept: D1 and D5 on one side and D2, D3 and D4 on the other side. It is possible to try alternative grouping, but this one seems the best. In practice, why and how tasks shall be related remains subject to debate. It can be assumed that it is algorithm dependent and task dependent. For instance, in the case of those variants of the LASSO algorithm used in this tutorial, related task find solutions more efficiently, by favoring a common pool of molecular descriptors. In the case of neural networks, related tasks shall share a common state of the hidden layers. The advantage of relating tasks according to external assumptions is that it should enable model building even when the datasets are too small to allow STL methods to succeed.

10. Conclusions

This tutorial has presented a typical situation for profiling modeling: for a compound an array of properties has to be learned. However for each instance, only some properties are measured, the others are unknown. The Multi-Task Learning framework is perfectly suited for this situation.

The tutorial has focused on the LASSO algorithm as a Single Task Learning method and compared it to two MTL versions of the algorithm: the MTL LASSO and the MTL L21 algorithms. It introduced also some performance measures that are commonly used to globally assess the performances of an MTL model.

The observations are that in general, the MTL algorithm is faster than the equivalent sequence of STL. Beside, if the coupling between tasks is strong the MTL models can significantly differ from the sequential approach. If the tasks are related then the MTL method can help finding better models than an STL approach. But when the tasks are not related, the MTL approach can be detrimental. In the context of the LASSO algorithm, related tasks are, in fact, modeling tasks that can be based on a common subset of variables.

Bibliography

Argyriou, A., Evgeniou, T., & Pontil, M. (2007). Multi-task feature learning. (B. Schölkopf, J. Platt, & T. Hoffman, Eds.) *Advances in Neural Information Processing Systems , 19*, pp. 41-48.

Bolton, E. E., Wang, Y., Thiessen, P. A., & Bryant, S. H. (2008). PubChem: Integrated Platform of Small Molecules and Biological Activities. In W. Wang, N. Barker, C. Simmerling,

J. D. Madura, & W. Cornell, *The Annual Reports in Computational Chemistry* (Vol. 4, p. 27). Washigton DC: American Chemical Society.

Brown, J., Okuno, Y., Marcou, G., Varnek, A., & Horvath, D. (2014, April 27). Computational chemogenomics: Is it more than inductive transfer? *Journal of Computer-Aided Molecular Design* .

Caruana, R. (1997). Multitask Learning. *Machine Learning , 28*, pp. 41-75.

Gaulton, A., Bellis, L. J., Bento, P. A., Chambers, J., Davies, M., Hersey, A., et al. (2011, September 23). ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research , 40* (D1), pp. D1100-D1107.

Hu, Y., Gupta-Ostermann, D., & Bajorath, J. (2014, January 29). Exploring Compound Promiscuity Patterns and Multi-Target Activity Spaces. *Computational and Structural Biotechnology Journal , 9* (13), p. 11.

J. Zhou, J. C. (2014, 05 20). *MALSAR Manual.* Retrieved 05 20, 2014, from MALSAR: http://www.public.asu.edu/~jye02/Software/MALSAR/

MATLAB and Statistics Toolbox Release 2014a. (2014). Natick, Massachusetts, United States: The MathWorks, Inc.

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *J. R. Statist. Soc. B , 58*, pp. 267-288.

Tropsha, A. (2010). Best Practices for QSAR Model Development, Validation, and Exploitation. *Molecular Informatics , 29* (6-7), pp. 476-488.

Varnek, A., Gaudin, C., Marcou, G., Baskin, I., Panday, A. K., & Tetko, I. V. (2009, January 6). Inductive Transfer of Knowledge: Application of Multi-Task Learning and Feature Net Approaches to Model Tissue-Air Partition Coefficients. *Journal of Chemical Information and Modeling , 49* (1), pp. 133-144.

Zhou, J. Y., Chen, J., & Ye, J. (2012). MALSAR: Multi-tAsk Learning via StructurAl Regularization. (A. S. University, Ed.) Arizona, USA.