Full demonstration using simulated data

In the following, we will outline a complete demonstration of how to install the CCA/PLS toolkit, and generate some of the results that are presented in the accompanying tutorial paper (Mihalik et al., in review). Computation time on a single machine is about 1-3 hours.

For more details, please see the in-depth online documentation of the toolkit available at https://anaston.github.io/cca_pls_toolkit/.

Installation and prerequisites

First, if you have Git already installed, clone the repository from Github using the following command in a terminal window under MacOS and Linux or in a power shell window under Microsoft Windows.

```
git clone https://github.com/anaston/cca_pls_toolkit
```

In case you don't want to clone the repository via Git, you can also go to https://github.com/anaston/cca_pls_toolkit and download all scripts as a zip folder and unzip into a directory of your choice.

After the toolkit is downloaded, open MATLAB and go to the folder containing the toolkit (e.g., double-click on the toolbox folder in MATLAB). To initialize the toolkit, run the following lines in the MATLAB command window:

```
mkdir external
set_path;
```

Dependencies

For this demonstration, we need to add an additional MATLAB toolbox (PALM). For this, download PALM manually using this link (https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/PALM/UserGuide), copy the `PALM` folder into the `external` folder of the CCA/PLS toolkit, then finally add PALM to the MATLAB path using the following line in the MATLAB command window:

```
set_path('PALM');
```

PALM is a toolbox that allows statistical inference using permutation testing whilst taking into account the dependencies in your data (e.g., family structure or diagnosis of subjects).

Analysis setup

In the following, we will run a Sparse Partial Least Squares (SPLS) analysis on low-dimensional simulated data. In general, the CCA/PLS toolkit uses a nested MATLAB structure to define all steps involved in a CCA/PLS analysis, which

includes the exact algorithm, the type of deflation, the validation and statistical inference and so on. We call this configuration structure `cfg` and we will set its fields using the standard dot notation in MATLAB. For more information on all possible configurations, see the toolkit documentation. You can either follow this demo and copy and paste the relevant code sections into the MATLAB command window one at a time or you can simply run the demo_simul_paper.m script located in the `demo` folder of the toolkit.

Project definition

First, we specify the `demo` folder as our project directory in the `cfg` structure.

```
% Project folder
cfg.dir.project = fileparts(mfilename('fullpath'));
```

If you are working with your own data and from within a different folder, you need to change this variable to contain the path to your project. It is important that this folder you specify will also contain a 'data' folder where your data is located.

Data

The data used in this demo has already been created and saved to a folder under `demo/data` using the `generate_data.m` function with 1000 examples, 100 features in both data modalities (of which 10% include signal linking the two modalities) and noise level 2. The two modalities of data are stored in `X.mat` and `Y.mat` files. If you are using your own data, please make sure to create files that will match the simulated data structure, i.e., create `X.mat` and `Y.mat` files containing simple MATLAB arrays with rows for examples and columns for features.

Machine

For this demo, we will use an SPLS algorithm. All CCA/PLS models and their settings (e.g., amount or regularization) can be specified using the `.machine` field of the `cfg` structure. In this case, we will set the `machine.name` field to 'spls' to run SPLS. The metrics used to evaluate the CCA/PLS algorithms can be defined in the `.machine.metric` field of the `cfg` structure. Here, we specify in-sample correlation ('trcorrel'), out-of-sample correlation ('correl') measuring the generalizability of the model, similarity of the X and Y weights ('simwx', 'simwy') measuring the stability of the model across different training sets of data as well as the explained variance in the training set of X and Y ('trexvarx', 'trexvary'). To select the best hyperparameter (i.e., L1-norm regularization for SPLS), we will use generalizability (measured as average out-of-sample correlation on the validation sets) as optimization criterion. This is set by `.machine.param.crit = correl`. Finally, `.machine.simw` defines the type of similarity measure used to assess the stability of model weights across splits. We

set this to 'correlation-Pearson' which will calculate Pearson correlation between each pair of weights.

```
% Machine settings
cfg.machine.name = 'spls';
cfg.machine.metric = {'trcorrel' 'correl' 'simwx' 'simwy' ...
    'trexvarx' 'trexvary'};
cfg.machine.param.crit = 'correl';
cfg.machine.simw = 'correlation-Pearson';
```

Framework

The `.frwork` field defines the general framework used in the analysis. We support two main approaches in the CCA/PLS toolkit. In a `holdout` machine learning framework, the data is divided into training and test sets by randomly subsampling subjects (see Monteiro et al. 2016). In a `permutation` statistical framework, the data is not splitted, focusing on in-sample statistical evaluation (see e.g., Smith et al. 2015). In this demo, we use the holdout framework with 10 inner and 10 outer data splits. For additional details, see the reference above, the accompanying tutorial paper or the online documentation of the toolkit.

```
% Framework settings
cfg.frwork.name = 'holdout';
cfg.frwork.split.nout = 10;
cfg.frwork.split.nin = 10;
```

Deflation

Next, we set the deflation of SPLS. In this demo, We will use PLS-mode A deflation. For more details on deflation strategies, see the accompanying tutorial paper or the online documentation of the toolkit.

```
% Deflation settings
cfg.defl.name = 'pls-modeA';
```

Environment

Next, we set the computational environment for the toolkit. As our data is relatively low-dimensional (i.e., number of features is not too high) SPLS will run quickly on a standard computer localy. For time-consuming analyses, this can be changed to a computer cluster environment.

```
% Environment settings
cfg.env.comp = 'local';
```

Statistical inference

In the last step, we define how the significance testing is performed. To reproduce the results reported in the accompanying tutorial paper, set this to 1000 permutations. If you need to save computation time, the number of permutations can be reduced. Please be aware, however, that too few permutation runs will make the precision of the calculated p-value low (e.g., 100 permutations allow to have a $p = 0.01$ at most).

```
% Number of permutations
cfg.stat.nperm = 1000;
```

Run analysis

To run the analysis, we simply update our `cfg` structure to add all necessary default values that we did not explicitly define and then run the `main` function. After the analysis, we clean up all the intermediate files that were saved during analysis to clean up disk space. This analysis will run for about 1-3 hours on a standard computer.

```
% Update cfg with defaults
cfg = cfg_defaults(cfg);

% Run analysis
main(cfg);

% Clean up analysis files to save disc space
cleanup_files(cfg);
```

Results

After running the analysis, you will notice that a *framework* folder has been automatically created by the toolkit inside the *demo* folder, containing all of the results. Inside the *framework* folder, another folder called *spls_holdout10-0.20_subsamp10-0.20* has been created. This analysis folder should be unique to your analysis and by default it is named depending on the exact algorithm and analytic framework used. In this demo, it indicates that we have used an SPLS analysis with 10 holdout sets and 10 validation sets, each containing 20% of the data. As decribed in the accompanying tutorial paper, the associative effects identified by CCA/PLS are calculated iteratively, for instance, to be able to optimize the model's hyperparameters separately for each associative effect. These associative effects are called *levels* and the CCA/PLS toolkit will continue calculating additional associative effects whenever the current one reaches statistical significance. After running this demo, there are two folders under *res* in our analysis directory for level 1 and level 2. Since level 2 did not reach statistical significance, the toolkit stopped the computation at this associative effect. A quick overview of the results of level 1 can be found in the

results_table.txt file located in the level 1 folder. In that file the correlation, associated p-value and, as this was an SPLS analysis, the number of selected features for the two data modalities are displayed for all specified splits (see Table below).

split	correl	pval	nfeatx	nfeaty
1	0.4355	0.0010	12	9
2	0.3963	0.0010	12	12
3	0.3564	0.0010	33	58
4	0.3517	0.0010	29	4
5	0.4748	0.0010	11	10
6	0.4837	0.0010	9	10
7	0.3389	0.0010	11	15
8	0.3996	0.0010	12	57
9	0.3865	0.0010	10	13
10	0.4334	0.0010	10	11

All the additional results are stored automatically within corresponding `.mat` files. E.g., the results of the hyperparameter optimization are located in the *grid* folder separately for each level. Likewise, the results of the permutation testing are located in a folder called *perm* separately for each level.

Loading the results

To visualize the results of the SPLS analysis, a number of functions are provided in the toolkit to plot, e.g., the weights or latent variables of the individual associated effects. Before these functions can be called, they have to be added to the MATLAB path. To do so, run the following command in the MATLAB command window.

```
% Set path for plotting
set_path('plot');
```

Similar to the `cfg` structure that is used to define all analysis steps, a `res` structure can be created and used to load and visualize the results. First, we will load the results of the first level by specifying the directory of our analysis and the level we want to analyze. This is most easily done by quickly loading the `cfg_1.mat` file located in our results directory. The `res_defaults` function will then load the necessary result structure automatically.

```
% Load res
res.dir.frwork = cfg.dir.frwork;
res.frwork.level = 1;
res = res_defaults(res, 'load');
```

Plot projections

To plot the data projections (or latent variables) that has been learnt by the SPLS model, the `plot_proj` function can be used (see Figure 1). In this demo, we will add an x and y label to the `res` structure and will pass this as first argument to the `plot_proj` function. Next, we specify the data modalities as cell array and the level of associative effect. In this example, we plot the projections of `X` and `Y` for the first associative effect. We set the fourth input parameter to 'osplit' so that the training and test data of the outer split will be used for the plot which is paired with the fifth argument defining the specific outer split we want to use. We set this to the best data split (highest out-of-sample correlation in the holdout set). The next argument specifies the colour-coding of the data using the training and test data as groups. Then we specify the low-level function that will plot the results. In this case it is `2d_group` which will call the `plot_proj_2d_group` function. All the other arguments of the `plot_proj` function are optional. In this demo, we flip the sign of the projection. Note, that that sign of the model weights or latent variables is arbitrary and can be changed for convenience (e.g., to compare results across models). Finally, we set the properties of the figure, axes and legends as Name-Value pairs.

```
% Plot data projections
plot_proj(res, {'X' 'Y'}, res.frwork.level, 'osplit', ...
    res.frwork.split.best, 'training+test', '2d_group', ...
    'gen.figure.ext', '.svg', ...
    'gen.figure.Position', [0 0 500 400], ...
    'gen.axes.Position', [0.1798 0.1560 0.7252 0.7690], ...
    'gen.axes.XLim', [-5 4.9], 'gen.axes.YLim', [-4.2 5], ...
    'gen.axes.FontSize', 22, 'gen.legend.FontSize', 22, ...
    'gen.legend.Location', 'best', ...
    'proj.scatter.SizeData', 120, ...
    'proj.scatter.MarkerFaceColor', [0.3 0.3 0.9;0.9 0.3 0.3], ...
    'proj.scatter.MarkerEdgeColor', 'k', 'proj.lsline', 'on', ...
    'proj.xlabel', 'Modality 1 latent variable', ...
    'proj.ylabel', 'Modality 2 latent variable');
```

Both the training and the test set show high correlations between the two latent variables, indicating that the learnt associative effect generalizes well.

Plot weights

Plotting model weights heavily depends on the kind of data that has been used in the analysis. In case of our simulated data, we are interested if the model recovered the weights that were used for generating the data (these true model weights were automatically saved in our `data` folder as `wX.mat` and `wY.mat`). We will use a stem plot with the true and recovered weights in different colors (see Figure 2 and 3). The `res` structure will need to be passed as first argument. Next, we specify the data modalities and the type of the modality as strings. In

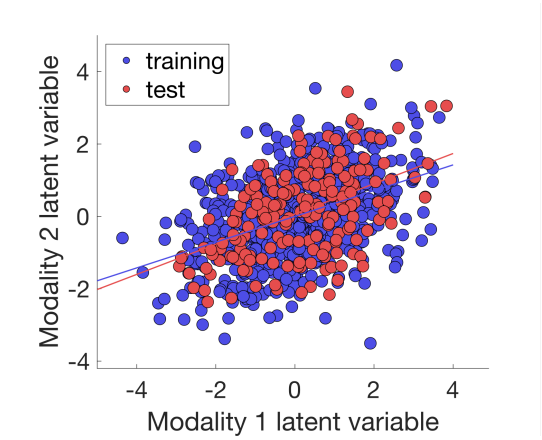

Figure 1: Latent variables of the first associated effects (blue: training set, red: test set).

this demo, we set these to `X` or `Y` and `simul`. The following argument defines the outer data split we want to use and will be set to the best split (as above for the data projections). Then we specify the low-level function that will plot the results. In this demo, it is set to `stem` to call the `plot_weight_stem` function and create a simple stem plot. Finally, we set the properties of the figure, axes and legends as Name-Value pairs.

```
% Plot X weights as stem plot
plot_weight(res, 'X', 'simul', res.frwork.split.best, 'stem', ...
    'gen.figure.ext', '.svg', ...
    'gen.figure.Position', [0 0 500 400], ...
    'gen.axes.Position', [0.1798 0.1560 0.7252 0.7690], ...
    'gen.axes.YLim', [-1.1 1.2], ...
    'gen.axes.YTick', [-1:0.5:1.2], ...
    'gen.axes.FontSize', 22, 'gen.legend.FontSize', 22, ...
    'gen.legend.Location', 'NorthEast', ...
    'simul.xlabel', 'Modality 1 variables', ...
    'simul.ylabel', 'Weight', 'simul.weight.norm', 'minmax');
```

The same plot can be generated for the `Y` data modality. All plots are automatically saved inside the `res/level1` folder.

```
% Plot Y weights as stem plot
plot_weight(res, 'Y', 'simul', res.frwork.split.best, 'stem', ...
    'gen.figure.ext', '.svg', ...
    'gen.figure.Position', [0 0 500 400], ...
    'gen.axes.Position', [0.1798 0.1560 0.7252 0.7690], ...
    'gen.axes.YLim', [-1.1 1.2], ...
    'gen.axes.YTick', [-1:0.5:1.2], ...
```

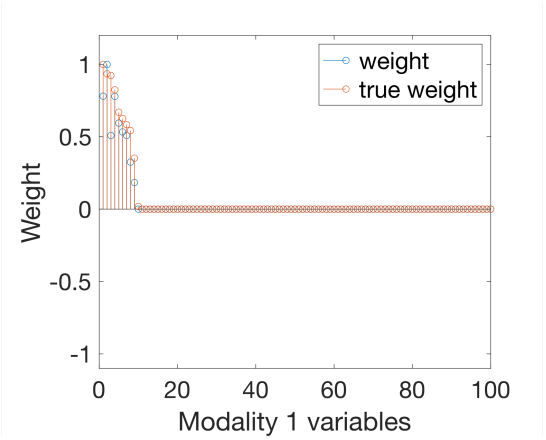

Figure 2: Weights of the first associative effect for modality 1 (blue: true model weights, red: weigths identified by SPLS).

```
    'gen.axes.FontSize', 22, 'gen.legend.FontSize', 22, ...
    'gen.legend.Location', 'NorthEast', ...
    'simul.xlabel', 'Modality 2 variables', ...
    'simul.ylabel', 'Weight', 'simul.weight.norm', 'minmax');
```

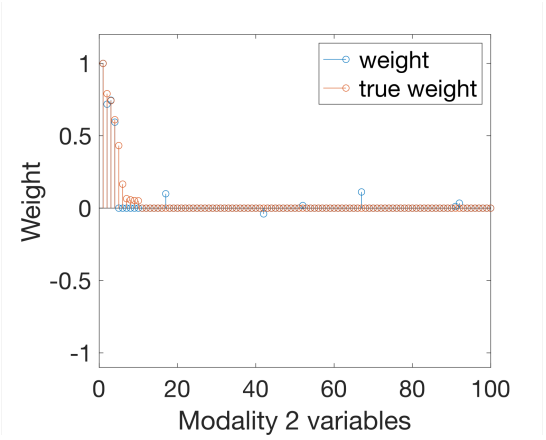

Figure 3: Weights of the first associative effect for modality 2 (blue: true model weights, red: weights identified by SPLS).

In this demo, we had only 1 significant associative effects. In case there are multiple significant associative effects, the process for plotting the results can be repeated for each level. Here, we plotted the results of the best data split, but other data splits can be also visualized in a similar manner. For more information on different algorithms, hyperparameter optimization, default parameters or

additional plotting functions, see the CCA/PLS toolkit documentation.