Risk Minimization by Cross Validation (RMCV)

Risk Minimization by Cross Validation (RMCV) is a score and an algorithm for learning Bayesian network (BN) classifiers [1].

Getting started with the RMCV code:

1. The RMCV code requires Kevin Murphy's Bayes Net Toolbox (BNT) [2]. The latter is freely available from https://github.com/bayesnet/bnt. You will have to make sure that it is correctly installed prior to using the provided code. In Windows, the following script will usually do the job:

```
current_dir=pwd;                    %saves current working directory
cd 'C:\MATLAB\work\bnt-master';     %this should point to the BNT directory
addpath(genpathKPM(pwd));           %add BNT paths
cd(current_dir);                    %return to working directory
```

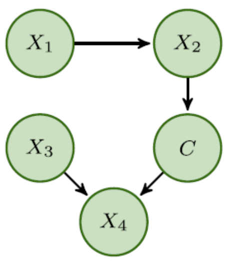
2. The RMCV score is described in Eq. (10) of [1]:

$$RMCV_K(D,\mathcal{G}) = \frac{1}{K}\sum_{k=1}^{K}\frac{K}{N}\sum_{i=1}^{N/K} L\left(c_{ki},\ \underset{c\in\{c_1,\ldots,c_{r_C}\}}{\operatorname{argmax}}\ P(C=c|D\backslash D_k^K, v'_{ki}, \mathcal{G})\right).$$

Feel free to check [1] for a detailed explanation of the score and terminology. A simple hill climbing search algorithm based on this score is also suggested in [1]:

```
Algorithm: RMCV
Input: An initial DAG, $\mathcal{G}$; A training set that is partitioned to $K$ mutually exclusive validation sets, $D = \{D_k^K\}_{k=1}^K$.
Output: BN model ($\mathcal{G}$,$\Theta$).
    compute $RMCV_K(D,\mathcal{G})$
    converged := false
    While converged = false
        For each $\mathcal{G}' \in$ Neighborhood($\mathcal{G}$)
            compute $RMCV_K(D,\mathcal{G}')$
        $\mathcal{G}^* := \arg\min_{\mathcal{G}'} RMCV_K(D,\mathcal{G}')$
        If $RMCV_K(D,\mathcal{G}^*) < RMCV_K(D,\mathcal{G})$
            Then $\mathcal{G} := \mathcal{G}^*$
            Else converged := true
    $\Theta$ := LearnParameters($D,\mathcal{G}$)
    Return ($\mathcal{G},\Theta$)
```

3. We will demonstrate our code using a synthetic dataset based on the BN shown in Fig. 1 of [1]:



(a) BN structure

$$\begin{aligned}
&P(X_1=0)=0.3, P(X_1=1)=0.7;\\
&P(X_3=0)=0.7, P(X_3=1)=0.3;\\
&P(X_2=0|X_1=0)=0.7,\ P(X_2=1|X_1=0)=0.3,\\
&P(X_2=0|X_1=1)=0.3,\ P(X_2=1|X_1=1)=0.7;\\
&P(C=0|X_2=0)=0.85,\ P(C=1|X_2=0)=0.15,\\
&P(C=0|X_2=1)=0.25,\ P(C=1|X_2=1)=0.75;\\
&P(X_4=0|X_3=0,C=0)=0.1,\ P(X_4=1|X_3=0,C=0)=0.9,\\
&P(X_4=0|X_3=0,C=1)=0.8,\ P(X_4=1|X_3=0,C=1)=0.2,\\
&P(X_4=0|X_3=1,C=0)=0.3,\ P(X_4=1|X_3=1,C=0)=0.7,\\
&P(X_4=0|X_3=1,C=1)=0.2,\ P(X_4=1|X_3=1,C=1)=0.8;
\end{aligned}$$

(b) Probabilities

First, we shall define a BN based on the graph and the given probabilities:

```
num_of_nodes=5;                 %total number of nodes
X1=1;                           %node names. Note: DAG must possess topological order.
X2=2;
C=3;
X3=4;
X4=5;
dag=zeros(num_of_nodes,num_of_nodes);   %five nodes 'X1','X2','C','X3','X4'
dag(X1,X2)=1;                           %an edge from X1 to X2
dag(X2,C)=1;                            %an edge from X2 to C
dag(X3,X4)=1;                           %an edge from X3 to X4
dag(C,X4)=1;                            %an edge from C to X4
discrete_nodes = 1:num_of_nodes;        %all nodes are discrete
node_sizes = 2*ones(1,num_of_nodes);    %all nodes have two possible values
%create BNT
bnet = mk_bnet(dag, node_sizes, 'names',{'X1','X2','C','X3','X4'},'discrete', discrete_nodes);
bnet.CPD{X1} = tabular_CPD(bnet, X1, [0.3 0.7]);
bnet.CPD{X3} = tabular_CPD(bnet, X3, [0.7 0.3]);
bnet.CPD{X2} = tabular_CPD(bnet, X2, [0.7 0.3 0.3 0.7]);
bnet.CPD{C} = tabular_CPD(bnet, C, [0.85 0.25 0.15 0.75]);
bnet.CPD{X4} = tabular_CPD(bnet, X4, [0.1 0.8 0.3 0.2 0.9 0.2 0.7 0.8]);
num_of_instances=1000;                   %set number of instances
my_train_samples=cell(num_of_nodes,num_of_instances);
my_test_samples=cell(num_of_nodes,num_of_instances);
%generate train and test datasets
for c1=1:num_of_instances
    my_train_samples(:,c1)=sample_bnet(bnet);
end
for c1=1:num_of_instances
    my_test_samples(:,c1)=sample_bnet(bnet);
end
train_data=cell2mat(my_train_samples.');
test_data=cell2mat(my_test_samples.');
```

At this point, at hand is a synthetic dataset ('my_train_samples') generated from the BN, from which we will learn an RMCV classifier. The provided code is a straight-forward implementation of the presented algorithm. The following procedures/functions are provided: 'rmcv_gs.m' is the main search procedure; 'rmho_score.m' is the Risk Minimization holdout score [1]. The search procedure calculates the RMCV score by calling RMHO several times with different training and validation folds. Note that a less straight-forward implementation of RMCV can be superior in terms of run-time compared to the provided code. Check Sections 5.4 and 7 in [1] for some ideas which can be easily implemented. The following simple script runs the RMCV algorithm:

```
%initial DAG
init_dag=dag;
init_dag(C,X4)=0;
%learn BN using RMCV
[final_dag,history_dags,history_scores]=rmcv_gs(init_dag,my_train_samples,my_test_samples,node_s
izes,C,4);
```

In this example, the initial DAG is close to the DAG of Fig. 1, but misses the edge between the class node *C* and X4. A learning process, based on enough instances, can usually recover this edge and yield a classifier with accuracy of roughly 83 percent. Note that the provided code not only returns the final DAG, but also the results from each iteration, including intermediate scores and classification accuracies.

4. We encourage you to try 'demo.m' and examine the various source code comments. Most are self-explanatory.

[1] R. Kelner and B. Lerner, "Learning Bayesian network classifiers by risk minimization," Int. J. Approx. Reas. **53**, 248-272 (2012).

[2] K. Murphy, "The Bayes net toolbox for Matlab," Comput. Sci. Stat. **33**, 331-350 (2001).