ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

FINAL PROJECT

1) PC ALGORITHM
2) DIRECTED INFORMATION GRAPH

NETWORK ANALYTICS MGT - 416

Emery Sébastien (258565)
11th June 2020

Problem 1

Determining the causality among a set of random variables $V = \{X_1, ..., X_p\}$ generally involves to first learn the causal graph of this set. The PC algorithm is used to learn such graph up to Markov equivalence classes. In this analysis, the population version of the PC algorihm is implemented and used on Data.mat dataset provided.

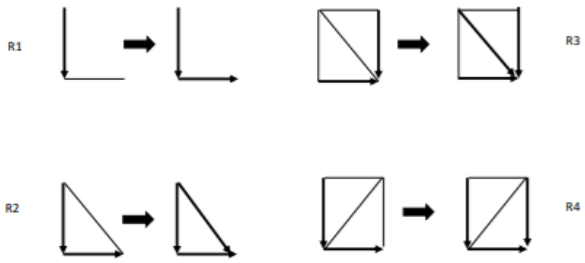
Table 1: Population version of the PC algorithm

Algorithm
Input : Vertex set V, observational data D, significance level α
Output : Estimated skeleton including v-structures
(1) Construct the complete undirected graph on the vertex set V
(2) Perform conditional independence tests at a given significance level α and delete edges based on the tests
(3) Orient v-structures

The conditional independence test in part (2) is done assuming that the joint distribution of random variables in V is a multivariate Gaussian. The tests in that case are done using the sample partial correlation $\hat{\rho}_{X_i,X_j|S}$ in comparison to a threshold $\Phi(1-\alpha/2)^{-1}/\sqrt{n-|S|-3}$ with $S \subseteq V \setminus \{X_i, X_j\}$, $\Phi \sim \mathcal{N}(0,\, 1)$ and significance level α. Tests are done gradually given an empty set to the entire S set. The test are performed as follows :

$$
\begin{aligned}
H_0 &: \hat{\rho}_{X_i,X_j|S} = 0 \\
H_1 &: \hat{\rho}_{X_i,X_j|S} \neq 0 \\
T &= \Phi(1-\alpha/2)^{-1}/\sqrt{n-|S|-3} \\
|\hat{\rho}_{X_i,X_j|S}| &\leqslant T \rightarrow H_0 \; is \; True
\end{aligned}
$$

This procedure allows to obtain the skeleton and orient the v-structures. Then using Meek's rules we can try to orient as many undirected edges as we can as follows :



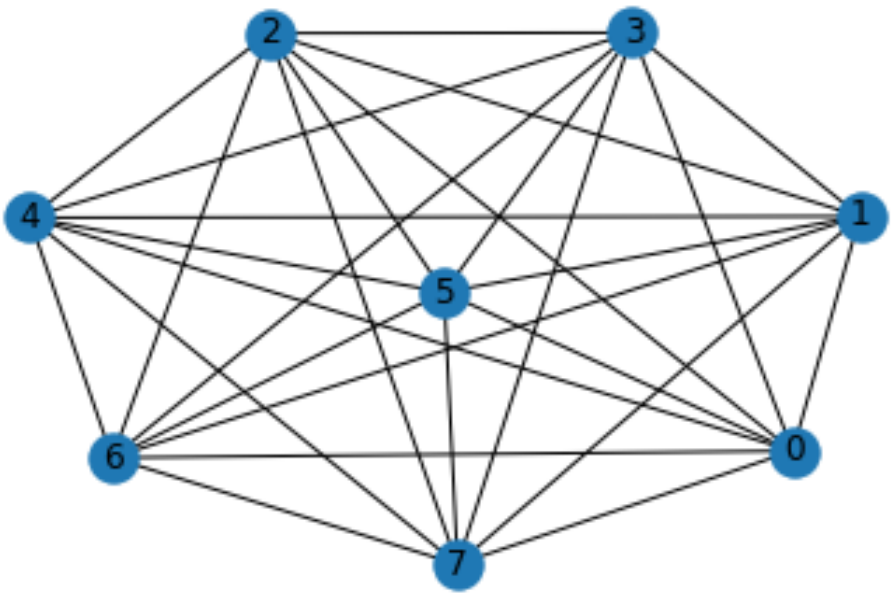
The resulting DAG can be learned completely or up to Markov equivalent class. The different class are statistically indistinguishable, this is the best we can do using this algorithm.

The analysis has been done using jupyter notebook and python. The functions used and the results are shown below. However, to have a better insight on the code please check the project_ex1.ipynb notebook.

The following code as been used to construct the complete undirected graph on the Vertex V of the Data.mat dataset with $\alpha = 0.05$:

```python
def complete_undirected_graph(D, save = False) :

    # get the number of RVs and sample
    P = data.shape[1]
    nb_sample = data.shape[0]

    # Adjacency matrix
    V_adjacency = np.ones((P,P))
    # no self connection
    np.fill_diagonal(V_adjacency, 0)

    # complete undirected graph
    G_undirected = nx.from_numpy_matrix(V_adjacency)
    nx.draw_networkx(G_undirected, with_labels=True)

    if save :
        plt.axis('off')
        plt.savefig("undirected_graph.png") # save as png
        plt.show() # display

    return G_undirected,V_adjacency,P,nb_sample
```

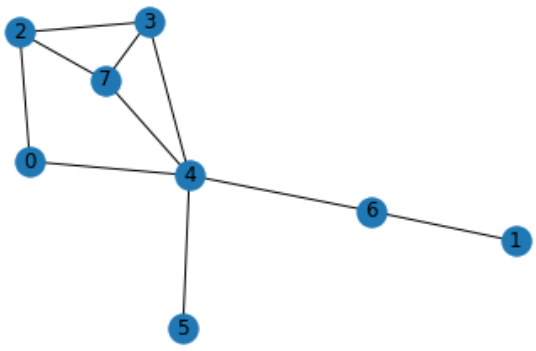
The folowing graph has been generated using networkx :



The skeleton has been estimated using conditional independence test with the following code :

```
def make_skeleton(D,alpha) :

    # plot the complete undirected graph
    plt.figure(figsize = (15,6))
    plt.subplot(121)
    G_undirected,V_adjacency,P,nb_sample = complete_undirected_graph(data)

    # data frame for correlation computation
    col = []
    for i in range(data.shape[1]):
        col.append(str(i))
    df = pd.DataFrame(D, columns=col)

    # combination of set s to control for partial correlation
    s_set = [[set() for i in range(P)] for j in range(P)]

    # loop over the number of variable to control
    for l in range(P) :
        # loop over all nodes
        for i in range(P):
            # loop over the remainning nodes for pairwise combination
            for j in range(i+1,P):
                adj_i = list(G_undirected.adj[i])
                # remove node j of the list to control
                if j in adj_i:
                    adj_i.remove(j)
                if len(adj_i) >= l:
                    for k in itertools.combinations(adj_i, l):
                        if nb_sample-len(k)-3 < 0:
                            break
                        # list for the input of partial_corr
                        s = []
                        for c in k:
                            s.append(str(c))
                        # partial correlation estimation
                        r = pg.partial_corr(data=df, x=str(i), y=str(j),covar=s,method='spearman').round(3)['r']['spearman']
                        # test value threshold
                        t = norm.isf(1-alpha/2) / np.sqrt(nb_sample-np.abs(len(s))-3)

                        # check if r is zero or not by comapring to t and store the s set
                        if np.abs(r) <= np.abs(t):
                            V_adjacency[i][j] = V_adjacency[j][i] = 0
                            s_set[i][j] |= set(k)
                            s_set[j][i] |= set(k)


    G_skeleton = nx.from_numpy_matrix(V_adjacency)
    plt.subplot(122)
    nx.draw_networkx(G_skeleton, with_labels=True)
    plt.show()

    return G_skeleton,V_adjacency,s_set,P
```

The folowing skeleton has been generated using networkx :

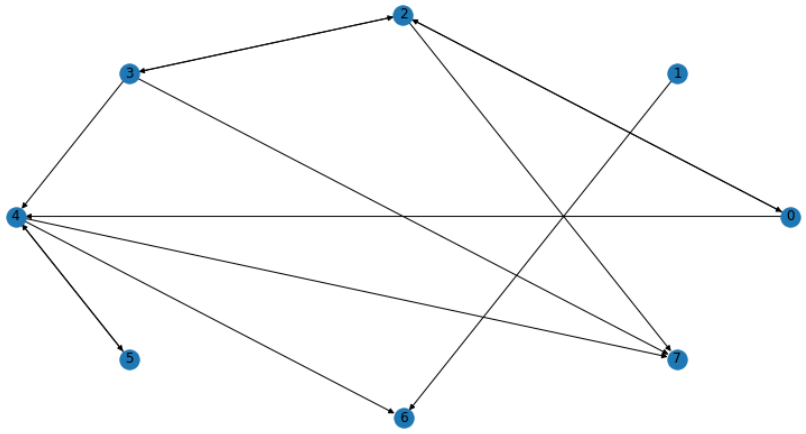


The following code has been used to orient the edges of the previous skeleton :

```
def Orient_edges(D,alpha) :

    # get the skeleton
    G_skeleton ,adj,s_set,P= make_skeleton(D,alpha)

    # transform skeleton graph in directed graph
    G_directed =G_skeleton.to_directed()
    nodes = G_skeleton.nodes()

    # Statistical orientation -> collider
    for (i, j) in itertools.combinations(nodes, 2):
        adj_i = set(G_directed.successors(i))
        if j in adj_i:
            continue
        adj_j = set(G_directed.successors(j))
        if i in adj_j:
            continue
        if s_set[i][j] is None:
            continue
        shared_k = adj_i & adj_j
        for k in shared_k:
            if k not in s_set[i][j]:
                if G_directed.has_edge(k, i):
                    G_directed.remove_edge(k, i)
                if G_directed.has_edge(k, j):
                    G_directed.remove_edge(k, j)

    # meek rules iterating for all combination(i,j)
    for (i, j) in itertools.combinations(nodes, 2):
        # Rule 1: Orient i-j into i->j whenever there is an arrow k->i such that k and j are nonadjacent.
        if G_directed.has_edge(i, j) and G_directed.has_edge(j, i):
            # Look all the predecessors of i.
            for k in G_directed.predecessors(i):
                # abort if there is an arrow i->k.
                if G_directed.has_edge(i, k):
                    continue
                # abort if k and j are adjacent.
                if G_directed.has_edge(i, j) or G_directed.has_edge(j, i):
                    continue
                # orient i-j into i->j
                G_directed.remove_edge(j, i)
```

```
        # Rule 2: Orient i-j into i->j whenever there is a chain i->k->j.
        if G_directed.has_edge(i, j) and G_directed.has_edge(j, i):
            # Find nodes k where k is i->k.
            succs_i = set()
            for k in G_directed.successors(i):
                if not G_directed.has_edge(k, i):
                    succs_i.add(k)
            # Find nodes j where j is k->j.
            preds_j = set()
            for k in G_directed.predecessors(j):
                if not G_directed.has_edge(j, k):
                    preds_j.add(k)
            # Check if there is any node k where i->k->j.
            if len(succs_i & preds_j) > 0:
                # orient i-j into i->j
                G_directed.remove_edge(j, i)

        # Rule 3: Orient i-j into i->j whenever there are two chains i-k->j and i-l->j such that k and l are nonadjacent.
        if G_directed.has_edge(i, j) and G_directed.has_edge(j, i):
            # Find nodes k where i-k.
            adj_i = set()
            for k in G_directed.successors(i):
                if G_directed.has_edge(k, i):
                    adj_i.add(k)
            # For all the pairs of nodes in adj_i,
            for (k, l) in itertools.combinations(adj_i, 2):
                # abort if k and l are adjacent.
                if G_directed.has_edge(i, j) or G_directed.has_edge(j, i):
                    continue
                # abort if not k->j.
                if G_directed.has_edge(j, k) or (not G_directed.has_edge(k, j)):
                    continue
                # abort if not l->j.
                if G_directed.has_edge(j, l) or (not G_directed.has_edge(l, j)):
                    continue
                # orient i-j into i->j.
                G_directed.remove_edge(j, i)

    plt.figure(figsize = (15,8))
    nx.draw_networkx(G_directed,pos =nx.circular_layout(G_directed) , with_labels=True, arrows=True)
```

The following graph was obtained :



The edges (0,2), (2,3) and (4,5) couldn't be learned with our version of the PC algorithm, they have the same conditional dependencies. Therefore, the number of different possible DAGS is all the different orientation combination of the three remaining edges making it 8 in this case. They are not shown here.

The time complexity of the algorithm is given by two component considering the conditional independence test as an elementary operation and d as the maximum degree of the graph. First, we have to compute the test for all combination of pairwise variable which is $O(\binom{d}{2}) \sim O(d^2)$, then we have to account to all possible conditioning which can be the empty set or the d-2 remaining variable which is given by $\sum_{k=0}^{d-2} \binom{d-2}{k} = 2^{d-2}$, therefore accounting for both effect give us the following complexity : $O(d^2 2^{d-2})$.

Problem 2

The stock prices' time series of twelve technology companies (Table 1) have been recorded in order to learn the causal structure among them. The prices were sampled every two minutes for seven days (T = 7).

Name	code
Apple Inc.	APPL (1)
Cisco Systems	CSCO (2)
Dell Inc.	DEL (3)
EMC Corporation	EMC (4)
Google Inc.	GOG (5)
Hewlett-Packard	HP (6)
Intel	INT (7)
Microsoft	MSFT (8)
Oracle	ORC (9)
International Business Machines	IBM (10)
Texas Instruments	TXN (11)
Xerox	XRX (12)

Table 2: List of companies in the analysis

The causality is measured in the Granger sense by computing the directed information ($I(x \to y||\underline{z})$) which represent the amount of information about Y causally provided by X given the set of variable z. This quantity is not symmetric in general ($I(x \to y||\underline{z}) \neq I(y \to x||\underline{z})$), both should be computed.

Assuming a Black-Scholes model of the market, the log-prices of these companies are following a jointly Gaussian process. As a consequence the directed information between two companies time series can be estimated as follows :

$$I(x \to y||\underline{z}) = \frac{1}{2}\sum_{t=1}^{T} \log \frac{|\sum_{y_1^t \underline{z}_1^{t-1}}||\sum_{x_1^{t-1} y_1^{t-1} \underline{z}_1^{t-1}}|}{|\sum_{y_1^{t-1} \underline{z}_1^{t-1}}||\sum_{x_1^{t-1} y_1^t \underline{z}_1^{t-1}}|} \tag{1}$$

where $\sum_{y_1^t \underline{z}_1^{t-1}}$ is the covariance matrix of (y(1),...,y(t),$\underline{z}$(1),...,$\underline{z}$(t-1).

The directed information graph (causal structure) is learned by estimating the directed information of all pairwise combination of two variables (companies) in both direction (as mentioned above) and stored in a square matrix (DI matrix).

The DI matrix can be thresholded to consider causal interaction only for DI values above a certain threshold by setting to zero values that are lower. This resulting matrix is used to plot the directed information graph.

The analysis has been done using jupyter notebook and python. The functions used and the results are shown below. However, to have a better insight on the code please check the project_ex2.ipynb notebook.

The DI matrix of the data contained in market.mat has been estimated using the following code :

```
def compute_DI_matrix(data) :

    # constant to recover from the shape of the data
    nb_company = data.shape[2]
    nb_sample = data.shape[1]
    T = data.shape[0]

    # initialize the DI matrix with zeros
    DI = np.zeros([nb_company,nb_company])

    # loop over all company combinations
    for X in range(nb_company) :

        for Y in range(nb_company) :
            tmp = 0.0
            # check if x and y are different
            if (X != Y) :

                # keep the indices of companies not in x or y
                comp = np.arange(nb_company)
                z = np.delete(comp,[X,Y])

                # loop over the time
                for t in range(1,T) :

                    # get the data up to time t
                    x_t_1 = data[:t,:,X]
                    y_t_1 = data[:t,:,Y]
                    y_t = data[:(t+1),:,Y]
                    z_t_1 = data[:t,:,z]
                    z_t_1 = np.swapaxes(z_t_1,1,2)
                    z_t_1 = np.reshape(z_t_1,(-1,nb_sample))

                    # get covariance matrices
                    cov_1 = np.cov(np.concatenate((y_t,z_t_1),axis=0))
                    cov_2 = np.cov(np.concatenate((x_t_1,y_t_1,z_t_1),axis=0))
                    cov_3 = np.cov(np.concatenate((y_t_1,z_t_1),axis=0))
                    cov_4 = np.cov(np.concatenate((x_t_1,y_t,z_t_1),axis=0))

                    # compute determinant
                    det_1 = np.linalg.det(cov_1)
                    det_2 = np.linalg.det(cov_2)
                    det_3 = np.linalg.det(cov_3)
                    det_4 = np.linalg.det(cov_4)

                    tmp += np.log((det_1/det_3)*(det_2/det_4))

                DI[X,Y] = 0.5*tmp

    return DI,np.around(DI, decimals=2)
```

The following DI matrix rounded to two decimals is obtained :

	APPL	CSCO	DEL	EMC	GOG	HP	INT	MSFT	ORC	IBM	TXN	XRX
APPL	0	0.28	0.25	0.29	0.47	0.5	0.64	0.68	0.55	0.38	0.38	0.18
CSCO	0.34	0	0.25	0.43	0.32	0.26	0.09	0.29	0.24	0.19	0.0.17	0.16
DEL	0.21	0.38	0	0.16	0.57	0.14	0.23	0.4	0.25	0.44	0.35	0.28
EMC	0.2	0.18	0.32	0	0.28	0.16	0.24	0.31	0.42	0.25	0.21	0.28
GOG	0.34	0.27	0.27	0.34	0	0.26	0.32	0.3	0.34	0.4	0.36	0.25
HP	0.35	0.42	0.2	0.34	0.3	0	0.2	0.64	0.22	0.42	0.25	0.33
INT	0.15	0.34	0.31	0.32	0.13	0.3	0	0.18	0.2	0.21	0.35	0.25
MSFT	0.48	0.55	0.34	0.36	0.54	0.21	0.37	0	0.2	0.28	0.38	0.41
ORC	0.44	0.34	0.25	0.1	0.32	0.13	0.32	0.2	0	0.27	0.27	0.26
IBM	0.29	0.41	0.47	0.48	0.3	0.27	0.22	0.42	0.32	0	0.18	0.21
TXN	0.23	0.26	0.15	0.16	0.24	0.24	0.14	0.12	0.09	0.17	0	0.18
XRX	0.11	0.15	0.23	0.24	0.11	0.17	0.17	0.16	0.24	0.25	0.29	0

Table 3: DI matrix of the market.mat dataset

Then I choose to apply a threshold on the DI matrix not rounded using the following code :

```
def threshold (thresh, DI) :

    # get the adjacency matrix of the resulting thresholding
    Adjacency = DI > thresh
    Adjacency = Adjacency.astype(float)

    # get indices where it is zero
    idx = np.argwhere(Adjacency == 0.0)
    # set to zero the element which are lower than the threshold
    DI_thresh = np.copy(DI)
    DI_thresh[idx[:,0],idx[:,1]] = 0.0

    return np.around(DI_thresh, decimals=2)
```

The threshold I used is set to 0.4 which is lower than the 0.6 used in HW3. This value allows to consider at least one causal interaction for every companies except TXN. Others values could be used as well as long as they are not to high or to low which result in no causal interaction at all or every causal interaction possible respectively, it should be reasonable. The resulting DI matrix is not shown as it would be the same than in table 2 with all value lower than 0.4 set to zero.

The following code using networkx library is used to generate the directed information graph from the thresholded DI matrix at 0.4 :

```
plt.figure(figsize=(18,18))
G = nx.from_numpy_matrix(np.matrix(DI_thresh), create_using=nx.DiGraph)
pos=nx.circular_layout(G) # positions for all nodes

# nodes
nx.draw_networkx_nodes(G,pos,nodelist=[0,1,2,3,4,5,6,7,8,9,10,11],node_color='orange',node_size=1200,alpha=0.5)

# edges
nx.draw_networkx_edges(G,pos,width=1.5,alpha=1.0)
edge_labels=dict([((u,v,),d['weight']) for u,v,d in G.edges(data=True)])
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels,label_pos = 0.38,font_color= 'r')

# some math labels
labels={}
labels[0]=r'APPL'
labels[1]=r'CSCO'
labels[2]=r'DEL'
labels[3]=r'EMC'
labels[4]=r'GOG'
labels[5]=r'HP'
labels[6]=r'INT'
labels[7]=r'MSFT'
labels[8]=r'ORC'
labels[9]=r'IBM'
labels[10]=r'TXN'
labels[11]=r'XRX'

nx.draw_networkx_labels(G,pos,labels,font_size=12)

plt.axis('off')
plt.savefig("labels_and_colors.png") # save as png
plt.show() # display
```

The directed information graph obtained is is sown below.