Multivariate Analysis and Statistical Learning PC Algorithm's implementation

Authors: Alex Foglia, Tommaso Puccetti

Università degli Studi di Firenze

21/12/2018

Theoretical references (1)

- Bayesian Networks can be rappresented as a **directed acyclic graph (DAG)**
- ”acyclic” means that there are no paths starting from a node v that ends with v itself, $\forall v \in G$

Theoretical references (2)

Let $G = (V, E)$ be a DAG relative to a finite set $X = \{X_v \forall v \in V\}$ of casual variables, then:

$$\forall u, v \in V \ non\ adjacent\ | v \in nd(u) \Rightarrow u \perp\!\!\!\perp v | nd(u) - v$$

Where $nd(u)$ is the set of **non-descendant** n of n, that are all those nodes u' for which there is no path from u to u'.

PC-Algorithm

Given a set of variables with a joint Gaussian probability distribution, it is possible to learn the DAG closer to the sample through the use of **PC-Algorithm**.
It is composed of two sub-functions that solve two different problems:

1. The construction of the skeleton (or **Moral Graph**)
2. The construction of the DAG from a given skeleton

Step one: read the dataset

- import **pandas** library
- call **pandas.read_csv()** function to read dataset
- define **alpha**
- call **get_skeleton** on dataset and alpha as arguments

Step two: initialization

- read names of the dataset variables accessing **dataset.columns** field
- retrieve the correlation matrix of the given dataset with **dataset.corr().values**
- initialize **N,n** as the number of sampling and the number of variables
- initialize **G** as the complete graph of dimension n
- initalize the **separation_set** as a list of list
- initialize **l = 0**, **stop = false**

Step three: define adj function

- define the **adj** function in order to get the adjacents of a node in a given graph

Step four: how many variables are actually dependent?

- set stop condition to true
- retrieve dependent variables: i,j are actually dependent if the adjacence matrix[i][j] is equal to 1
- call the set of dependent variables **act_dep**

Step five: variables needed for independence test

- for **x,y** in **act_dep**
- retrieve the **neighbors** of **x** calling the **adj()** function
- remove y from the **neighbors** set
- if **neighbors** set has dimension $\geq$ **l** then
  - if **neighbors** set has dimension $>$ **l** go ahead

Step six: conditional independence test

- foreach set **K** of neighbors of dimension **l**
- test independence of **x** and **y** given **K**
- if the p value is greater than **alpha**:
  - remove the edge x,y setting **G[x][y] = 0**
  - set **K** as the **separation_set[x][y]**

Step eight: from the skeleton to the CPDAG

- return **G** and **separation_set**
- call **to_cpdag(G, separation_set)**

Step eight: define the getIndependents() function

- define **getIndependents(adj_matrix,reqij, reqji)**
- this function retrieve all the variables i,j such that: **adj_matrix[i][j] == reqij** and **adj_matrix[j][i] == reqji**

Step nine: CPDAG initialization

- set the **cpdag** as the skeleton
- set **dip** as the set of variables i,j for which exists an edge from i to j

Step ten: rule ”zero”

- foreach pair x,y in **dip**:
- add to **allZ** all the variables **z** for which exists an egde from **z** to **j** and **z is not x**
- if:
  there is no edge between x and z
  there is a separation set between x and z
  there is a separation set between z and x
  y is not in separation set between x and z or **in separation set between z and x**, then:
- remove the edge from y to x and from z to y

Step eleven: apply rules

- using the same logic we apply the known rules 1,2 and 3
- return the resulting cpdag
- using **matplotlib** and **networkx** we are able to plot the resulting cpdag

Python vs R

- consider this two codes