Tongge Huang
09/17/2018

AERE 504X Project_1 Structure Learning

Tongge Huang

Description

The Bayesian network is widely used in data exploration. However, in some cases, we have bare knowledge about the relationships between variables in the dataset. To generate Bayesian network, the structure learning algorithms must be used to understand the relationships between each data node.

The purpose of this project is using Bayesian scoring method and some searching strategies to find out the structure which can explain the dataset in an appropriate manner.

This project is a competition to find Bayesian network structures that best fit some given data. The fitness of the structures will be measured by the Bayesian score (described in the course textbook DMU 2.4.1).

Data

There are four datasets used in this project to evaluate the structure learning algorithm.

- `titanic`: 8 variables 899 observations
- `whitewine`: 12 variables 4898 observations
- `schoolgrades`: 28 variables 2287 observations
- `test`: 5 variables 1000 observations

Method

The basic methods I used in this project are Bayesian Structure Scoring and K2 Searching.

Bayesian Structure Score

The basic idea of BS score is to find a diagram (denoted as `G`) which can maximizes P(G|D) where `D` is the given dataset. The equation is given below.

Tongge Huang
09/17/2018

$$\ln P(G \mid D) = \ln P(G) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} \ln \left(\frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0} + m_{ij0})} \right) + \sum_{k=1}^{r_i} \ln \left(\frac{\Gamma(\alpha_{ijk} + m_{ijk})}{\Gamma(\alpha_{ijk})} \right).$$

(2.83)

where,

- `ri` is the number of instantiations of `Xi`
- `qi` is the number of instantiations of the parents of `Xi`
- `mijk` to represent the number of times `Xi = k` given `πij` in the dataset `D`

K2 Algorithm

K2 is one of the most common search strategies. It start with a no directed edges diagram. And iterating over all the nodes by greedily adding the parents until the BS score won't increase by adding other parents.

The shortcoming of this searching method is not guarantee finding a globally optimal since the initial diagram was random generated.

Notes: The original K2 method assume the Dirichlet prior parameters `αijk = 1` for all `i`, `j`, and `k`.

Result

The corresponding scores and time used for generating the structure are listed below. The `Diagrams` are listed in the `appendix` section.

Data	Runing time	Scores
titanic	0.56 s	-3833.5
whitewine	1.97 s	-42128.0
schoolgrades	18.3 s	-47381.8
test	0.13 s	-2220.2

Reflection

By using the K2 algorithm, the Bayesian structure which is generate from structure learning can be used to represent the dataset. For instance, the `titanic`

structure shows both the `sex` and `passengerclass` are the parents of `survived`, which is reasonable.

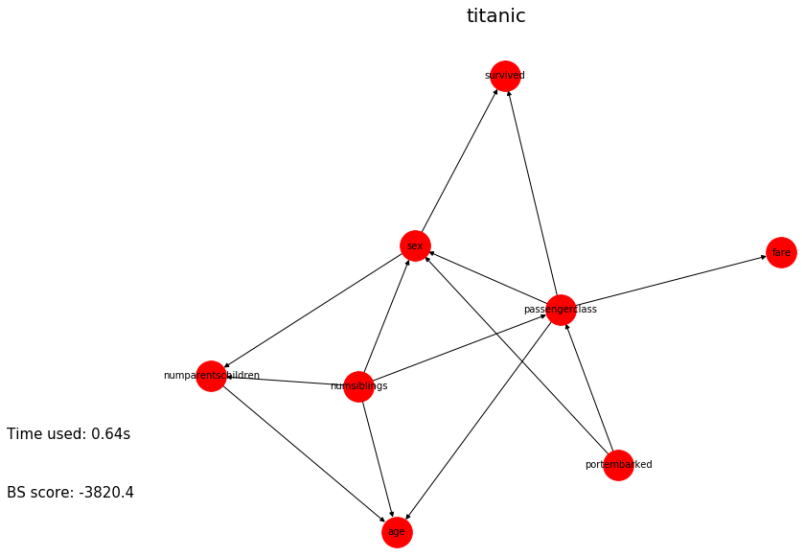
However, the shortcomings of my entire process are obviously. First, the K2 search will be trapped by `local optimal` rather than `globally optimal`. The result of the K2 search is highly depending on the initial random diagram and searching ordering. Also, the number of allowed parents for each node is another parameter based on the manual judgement.

To avoid local optimal problem, I think `Randomized Restart` and `Tabu Search` are the two possible methods to improve my results.

Reference

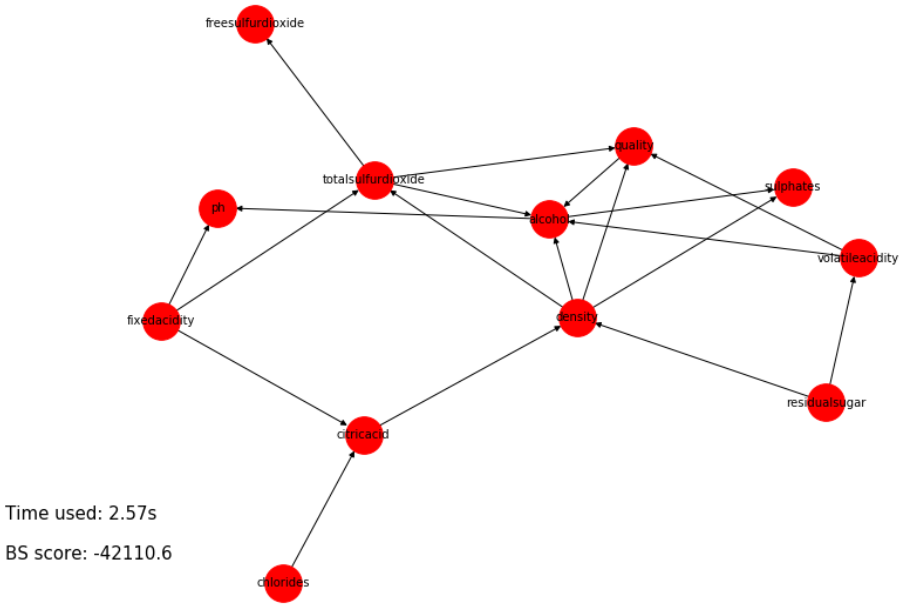
1. Decision Making Under Uncertainty Theory and Application, Mykel J. Kochenderfer
2. Class Code: https://github.com/Division-Q/aere504-notebook
3. Thomas Hossler: https://github.com/thomashossler/AA228-DecisionMakingUnderUncertainty/blob/master/structureLearning/Report.pdf
4. Ken Oung: https://github.com/kenoung/aa228/blob/master/project1/writeup.pdf
5. Ramon Iglesias: https://github.com/huevosabio/AA228/blob/master/project1/Project%201%20-%20Ramon%20Iglesias.pdf

Diagrams



Tongge Huang
09/17/2018

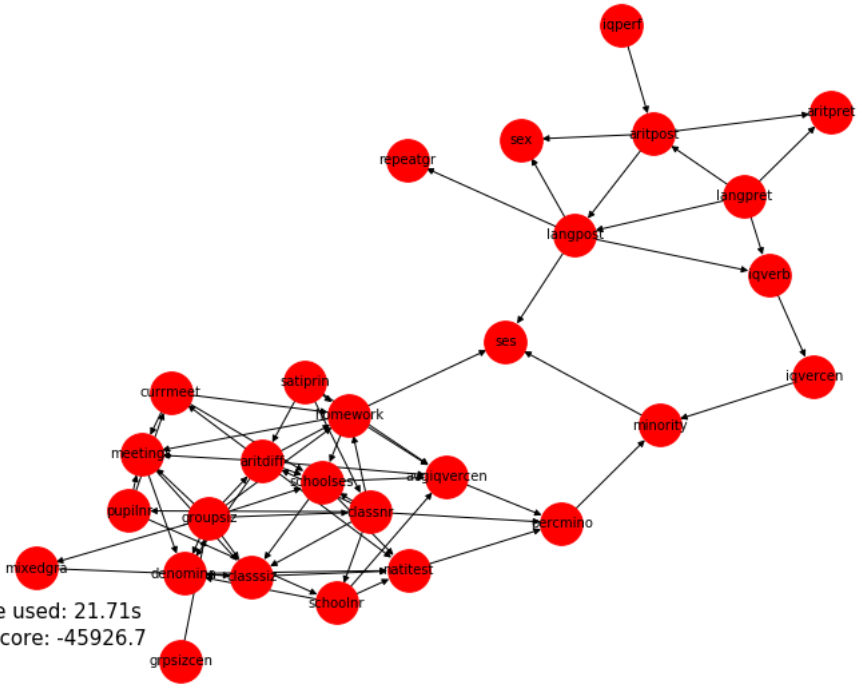
whitewine



Time used: 2.57s

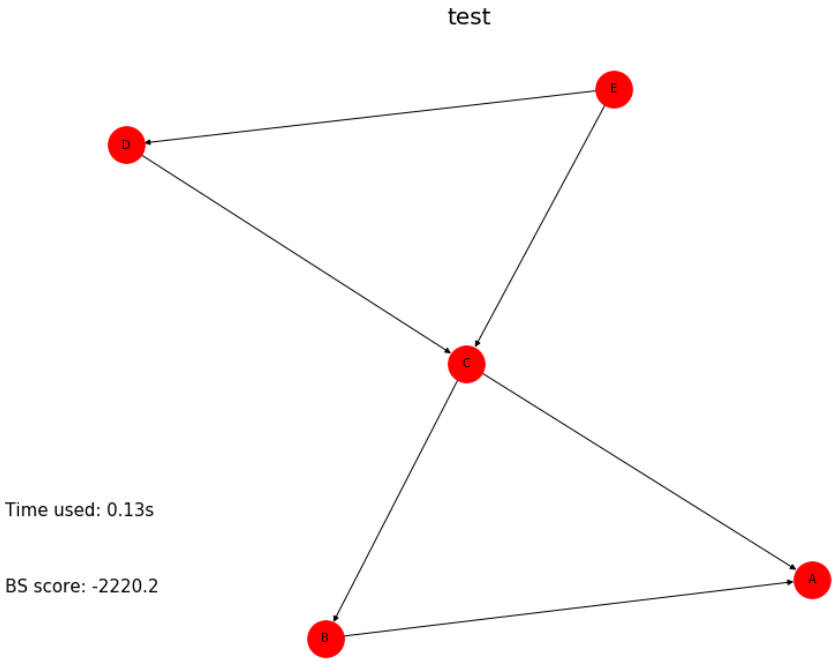
BS score: -42110.6

schoolgrades



Time used: 21.71s
BS score: -45926.7

Tongge Huang
09/17/2018



Appendix

```
import numpy as np
import pandas as pd
from random import shuffle

import graphviz
from networkx.drawing.nx_agraph import graphviz_layout

import itertools
import networkx as nx

from timeit import default_timer as timer
import matplotlib.pyplot as plt
from scipy.special import gammaln
```

Tongge Huang
09/17/2018

```
import warnings

warnings.filterwarnings("ignore")


def initial_graph(df):

    G = nx.DiGraph()

    nodes = list(df.columns)

    shuffle(nodes)

    G.add_nodes_from(nodes)

    return G


def bs_score(node,parents,data):

    score = 0.0

    if not parents:

        mijk = np.array([data[node].value_counts()])

    else:

        mijk = np.array(pd.pivot_table(data[[node]+parents],index=
parents,columns=node,aggfunc=len).fillna(0))


    for i,m in enumerate(mijk):

        alpha = len(np.unique(data[node]))

        score += ((gammaln(alpha)-gammaln(alpha+m.sum())).sum()+su
m([gammaln(1+m[j]) for j in range(alpha)])).sum()

    return (score)


# Make sure the graph is DAG.

def succ_remove(G,node,nodes):

    possible_parents = nodes[:]

    for i in list(G.succ[node]):
```

```
            possible_parents.remove(i)
        possible_parents.remove(node)
        return possible_parents


def K2_search(data):
    G = initial_graph(data)
    nodes = list(G.nodes)


    sum_score = 0.0
    for i, node in enumerate(nodes):
        # generate all possible parents to search
        possible_parents = succ_remove(G,node,nodes)
        # Calculate current bs score
        current_parents = []
        current_score = bs_score(node, current_parents, data)
        while possible_parents:
            scores = []
            for j,parent in enumerate(possible_parents):
                new_score = bs_score(node, current_parents + [pare
nt], data)
                scores.append(new_score)
            # pick the best local score
            best_score = np.max(scores)
            if best_score>current_score:
                new_parent = possible_parents.pop(np.argmax(score
s))
                # add edges to temperal graph
                Gtemp = G.copy()
```

```
                Gtemp.add_edge(new_parent, node)
                # check if the diagram is DAG
                if nx.is_directed_acyclic_graph(Gtemp):
                    G.add_edge(new_parent, node)
                    current_parents.append(new_parent)
                    current_score = best_score
                    # Parents threshold
                    if len(current_parents) >= 5:#int(0.8*len(node
s)):
                        break
                else:
                    Gtemp = G
                    break
            else:
                break
        sum_score += current_score
    return G,sum_score

def structure_learn(data):
    start = timer()
    G, sum_score = K2_search(data)
    end = timer()
    time_used = end - start
    return G, sum_score, time_used

def graph_generator(G,score,timeused,dataname):
    px,py=10,50
    fig = plt.figure(figsize=(10,8))
```

```
    nx.draw(G, pos=graphviz_layout(G),with_labels=True, node_size=
1000,font_size=10)

    plt.title('{}'.format(dataname), fontsize=20)

    plt.text(px, py, 'Time used: {0:.2f}s'.format(timeused),fontsi
ze=15)

    plt.text(px, py-20, 'BS score: {0:.1f}'.format(score), fontsiz
e=15)

    fig.savefig('{}.png'.format(dataname), bbox_inches='tight')

    plt.show()


titanic = pd.read_csv('titanic.csv', sep=',')

whitewine = pd.read_csv('whitewine.csv', sep=',')

schoolgrades = pd.read_csv('schoolgrades.csv', sep=',')

test = pd.read_csv('structurelearning_test.csv',sep=',')


G_s, sum_score_s, time_used_s = structure_learn(titanic)

G_m, sum_score_m, time_used_m = structure_learn(whitewine)

G_l, sum_score_l, time_used_l = structure_learn(schoolgrades)


graph_generator(G_s, sum_score_s, time_used_s,'titanic')

graph_generator(G_m, sum_score_m, time_used_m,'whitewine')

graph_generator(G_l, sum_score_l, time_used_l,'schoolgrades')

graph_generator(G_t, sum_score_t, time_used_t,'test')
```