ITR Project Version 2.0 : comprehensive search implementation

Jie Xue

March 21, 2018

Contents

1 Introduction

1.1 Individual treatment recommendation

According to Wiki, precision medicine is a medical model that proposes the customization of healthcare, with medical decisions, practices, and/or products being tailored to the individual patient. In this model, diagnostic testing is often employed for selecting appropriate and optimal therapies based on the context of a patient's genetic content or other molecular or cellular analysis. Tools employed in precision medicine can include molecular diagnostics, imaging, and analytics/software.

With new treatments and novel technology available, personalized medicine has become an important piece in the new era of medical product development. Traditional statistics methods for personalized medicine and subgroup identification primarily focus on single treatment or two arm randomized control trials. Motivated by the recent development of outcome weighted learning framework, we propose an alternative algorithm to search treatment assignments which has a connection with subgroup identification problems. Our method focuses on applications from clinical trials to generate easy to interpret results. This framework is able to handle two or more than two treatments from both randomized control trials and observational studies. We implement our algorithm in C++.

1.2 Mathematical background

- There are N subjects from a large population.
- A_i is the treatment assignment (actions), where $i = 1, \ldots, N$.
- Y_i is the response assuming that larger Y_i is better (rewards).
- X_i is a vector of covariates.
- (Y, A, X) is the generic random variable of $\{(Y_i, A_i, X_i)\}$.
- $\mathcal{P}$ is the distribution of (Y, A, X).
- E is the expectation with respect to $\mathcal{P}$.
- Population space χ, i.e. $X_i \in \chi$
- $\mathcal{D}(\cdot)$ is a treatment recommendation based on covariates, i.e. $\mathcal{D}(\cdot) : \chi \to A$.
- $\mathcal{P}^{\mathcal{D}}$ is the distribution of (Y, A, X) given that $A = \mathcal{D}(X)$.

Define

$$E^{\mathcal{D}}(Y) = \int Y d\mathcal{P}^{\mathcal{D}} = \int Y \frac{d\mathcal{P}^{\mathcal{D}}}{d\mathcal{P}} d\mathcal{P} = E\left[\frac{I\{A = \mathcal{D}(X)\}}{p(A|X)} Y\right]$$

where

$$\frac{d\mathcal{P}^{\mathcal{D}}}{d\mathcal{P}} = \frac{p(y|x,a)\, I\{a = \mathcal{D}(x)\}\, p(x)}{p(y|x,a)\, p(a|x)\, p(x)} = \frac{I\{a = \mathcal{D}(x)\}}{p(a|x)}$$

Our objective is to find $\mathcal{D}(\cdot)$ to maximize the following value function:

$$\mathcal{D}_o \in \arg\max_{\mathcal{D} \in R} E^{\mathcal{D}}(Y) = E\left[\frac{I\{A = \mathcal{D}(X)\}}{p(A|X)} Y\right]$$

where R is a space of possible treatment recommendations.

Example 1 depth search (from ITR.ABC):

Suppose we have two doctors and each of them has a treatment rule. Which doctor is a better one?

- Doctor Adam: give patients treatment 1 if $X \geq 2$, and treatment 2 otherwise, denoted as $\mathcal{D}_A(X)$.
- Doctor Barry: give patients treatment 1 if $X \geq 3$, and treatment 2 otherwise, denoted as $\mathcal{D}_B(X)$.

Table 1: Example calculation

ID	Y	A	X	$P(A\|X)$	$\mathcal{D}_A$	$\mathcal{D}_B$	$\mathcal{D}_A = A$	$\mathcal{D}_B = A$
1	1	1	1	0.5	2	2	0	0
2	2	1	2	0.5	1	2	1	0
3	3	1	3	0.5	1	1	1	1
4	4	1	4	0.5	1	1	1	1
5	5	1	5	0.5	2	2	1	1
6	3	2	1	0.5	1	2	1	1
7	3	2	2	0.5	1	1	0	1
8	3	2	3	0.5	1	1	0	0
9	3	2	4	0.5	1	1	0	0
10	3	2	5	0.5	1	1	0	0

Doctor Adam:

$$\begin{aligned} E^{\mathcal{D}_A}(Y) &= \frac{1}{10}\left(\frac{0}{0.5}\times 1+\frac{1}{0.5}\times 2+\frac{1}{0.5}\times 3+\frac{1}{0.5}\times 4+\frac{1}{0.5}\times 5+\frac{1}{0.5}\times 3+\frac{0}{0.5}\times 3+\frac{0}{0.5}\times 3+\frac{0}{0.5}\times 3+\frac{0}{0.5}\times 3\right) \\ &= 3.4 \end{aligned}$$

Doctor Barry:

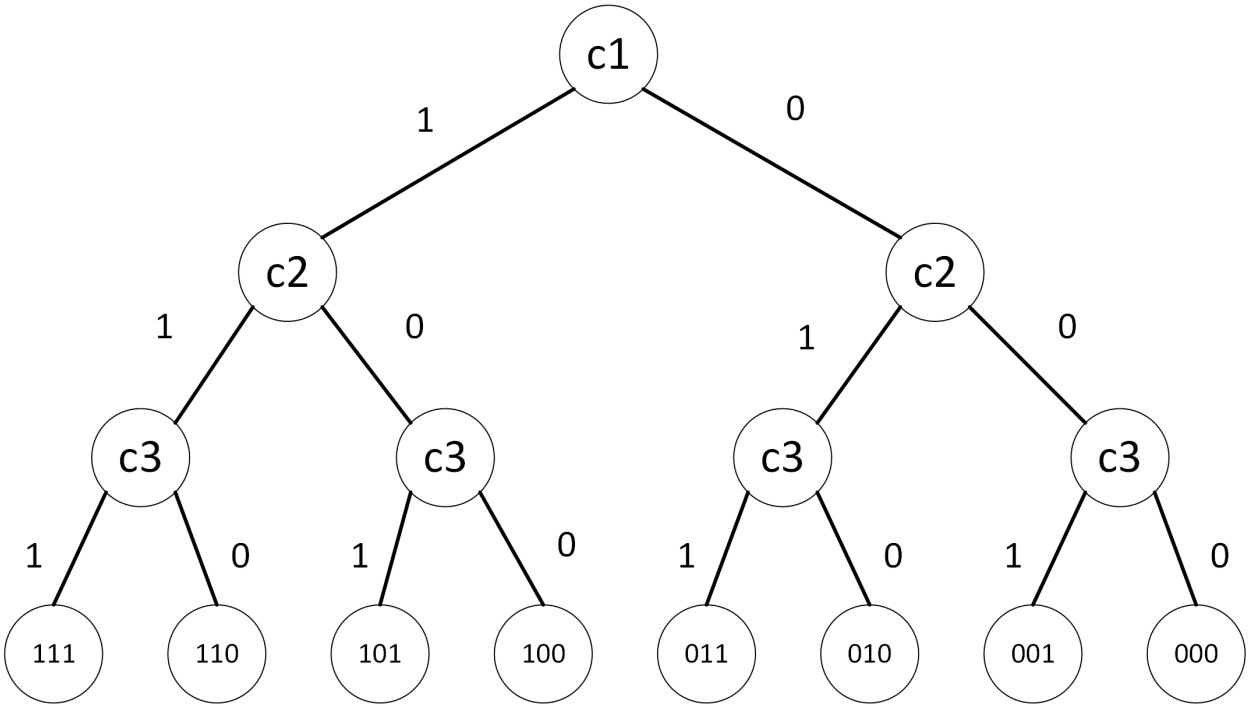
$$\begin{aligned} E^{\mathcal{D}_B}(Y) &= \frac{1}{10}\left(\frac{0}{0.5}\times 1+\frac{0}{0.5}\times 2+\frac{1}{0.5}\times 3+\frac{1}{0.5}\times 4+\frac{1}{0.5}\times 5+\frac{1}{0.5}\times 3+\frac{1}{0.5}\times 3+\frac{0}{0.5}\times 3+\frac{0}{0.5}\times 3+\frac{0}{0.5}\times 3\right) \\ &= 3.6 \end{aligned}$$

Conclusion: Doctor Barry's rule is better than Doctor Adam's.

1.3 Logic for comprehensive search

The comprehensive searching problem could be formulated as nested conditional operations with independent, exclusive mutual criteria.

Assume we have three criteria, c_1, c_2, and c_3. For each criteria, there would be a binary result for satisfied giving 1 or unsatisfied giving 0.



Therefore, there exists 8 possible results from 111 to 000 which fits a 8 length vector. In this project, the last criteria checks if the patient applies treatment 1. As a result, the expectation for results 1 (11) use the information of 110, 100, 010, and 000. If we know the summation of all response that take treatment 0. Result 1 could be simplified as the function of 111 and 110.

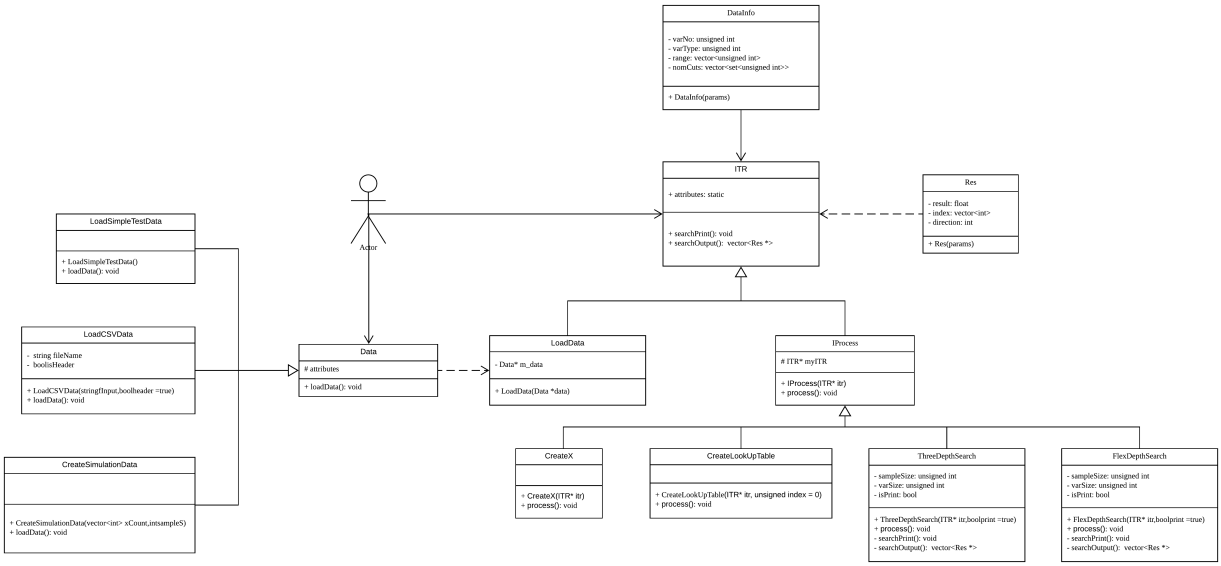
2 Architecture and Design

2.1 Design goals

In the project, we implement the ITR comprehensive search by using C++11. Apply appropriate design pattern with the consideration of abstraction, code reuse, stability, and scalability. The desired running time is less than 120s.

2.2 Class Diagram

Figure 1: UML class diagram



The ITR system is comprised of 4 main parts as following:

- Res: Store results
- DataInfo: generate searching information for each variable
- Data: interface for input data
  - LoadSimpleTestData: load ITR.ABC example data
  - LoadCSVData: load CSV data of 9 covariants, 100 samples
  - CreateSimulationData: create simulation data
- ITR: class for comprehensive search
  - LoadData: load data (base class for ITR)
  - IProcess: interface for operations
    - CreateX: merge types of data
    - CreateLookUpTable: create look up table

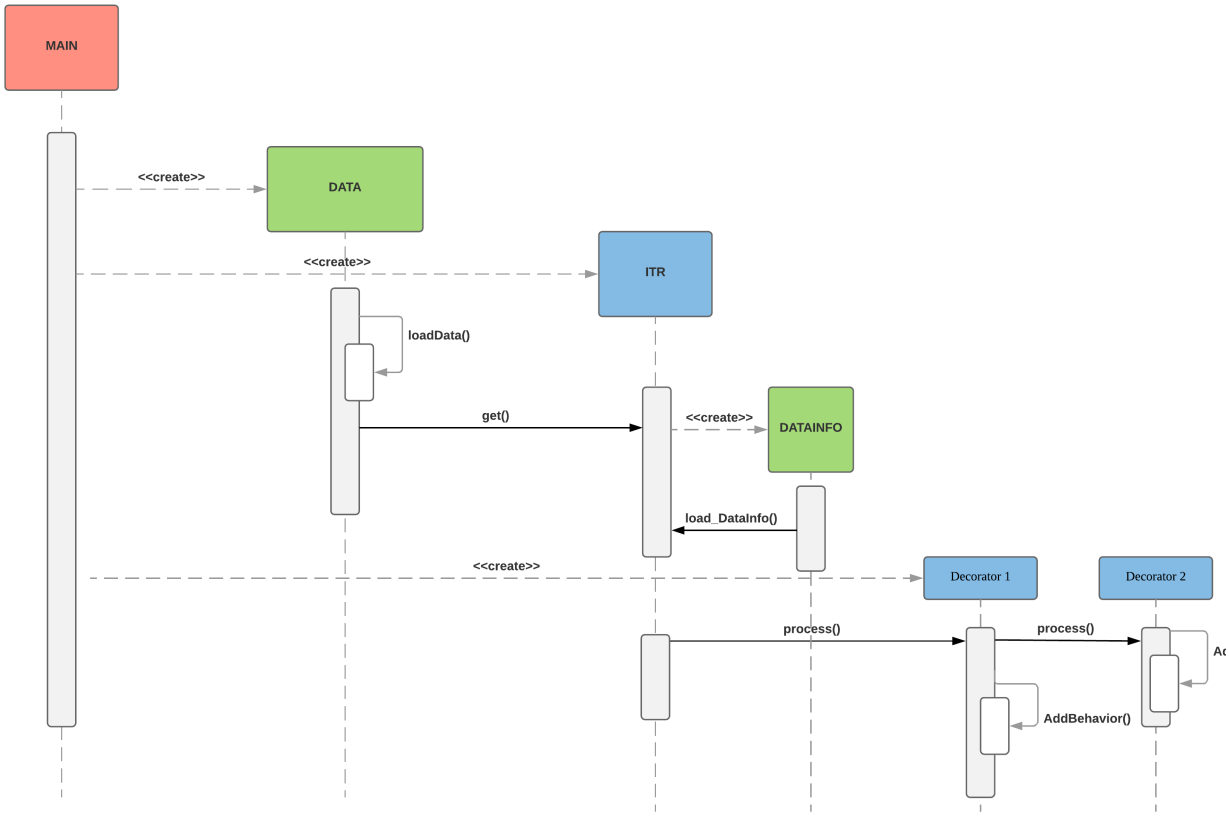
- OneDepthSearch: 1 depth search
- TwoDepthSearch: 2 depth search
- ThreeDepthSearch: 3 depth search
- FlexDepthSearch: flexible depth search

The decorate method design pattern is applied for Data.

The decorator pattern is implemented for ITR operation.

2.3 Sequence Diagram

Figure 2: UML sequence diagram



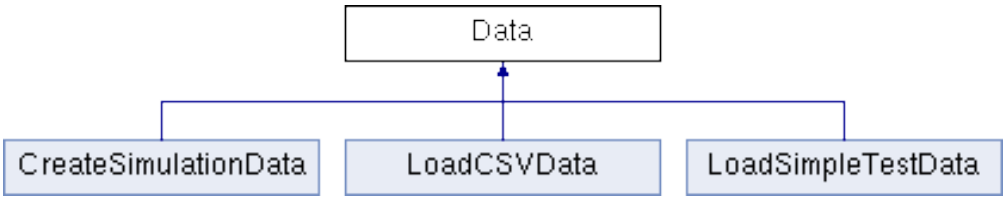
The data class is first created. Then data is passed to ITR through a pointer. DataInfo is called to generate criteria information for each variable. Decorators could be added depends on the algorithm.

2.4 Detailed Class Design

2.4.1 Data Class Reference

The purpose of the interface Data is to load or create data.

Inheritance diagram for Data:



Protected Attributes:

- unsigned int sampleSize
- vector<unsigned int> id
- vector<vector<double>> y
- vector<vector<unsigned int>> actions
- vector<vector<double>> x_Cont;
- vector<vector<unsigned int>> x_Ord
- vector<vector<unsigned int>> x_Nom
- vector<unsigned int> dataType

Public Member Functions

- **Data**(): constructor
- virtual void **loadData**(): virtual function
- const vector<unsigned int>& **getID**()
- const vector<vector¡double>>& **getY**()
- const vector<vector¡unsigned int>>& **getActions**()
- const vector<vector¡double>>& **getX_Cont**()
- const vector<vector<unsigned int>>& **getX_Ord**()
- const vector<vector<unsigned int>>& **getX_Nom**()
- const vector<unsigned int>& **getX_Type**()
- unsigned int **getSampleSize**()

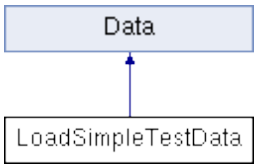
More information about this class can be found from the following files:

- Data.h
- Data.cpp

2.4.2 LoadSimpleTestData Class Reference

The purpose of LoadSimpleTestData class is load data given by ITR.ABC.

Inheritance diagram for LoadSimpleTestData:



Public Member Functions

- **LoadSimpleTestData**() : constructor
- void **loadData**() : overload

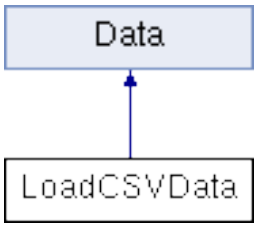
More information about this class can be found from the following files:

- LoadSimpleTestData.h
- LoadSimpleTestData.cpp

2.4.3 LoadCSVData Class Reference

The purpose of LoadCSVData class is load CSV data.

Inheritance diagram for LoadCSVData:



Private Attributes:

- string fileName
- bool isHeader

Public Member Functions:

- **LoadCSVData**(string fInput, bool header = true): constructor, default header = true
- void **loadData**() : overload

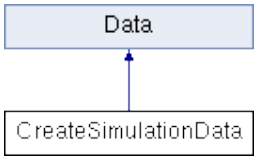
More information about this class can be found from the following files:

- LoadCSVData.h
- LoadCSVData.cpp

2.4.4 CreateSimulationData Class Reference

The purpose of CreateSimulationData class is create simulation data.

Inheritance diagram for CreateSimulationData:



Public Member Functions:

- **CreateSimulationData**(vector<int> xCount, int sampleS): constructor, sampleS for sample size, 0 for cont., 1 for ord., 2 for nom.
- void **loadData**() : overload

Private Member Functions:

- void **createID**();
- void **createActions**();
- void **createY**();
- void **createX**();
- vector<unsigned int> **createSeed**()
- vector<double> **createDouble**(unsigned int seed, double lowerBound = 0.0, double upperBound = 100.0)
- vector<unsigned int> **createInt**(unsigned int seed, unsigned int lowerBound = 0, unsigned int upperBound = 4)

More information about this class can be found from the following files:

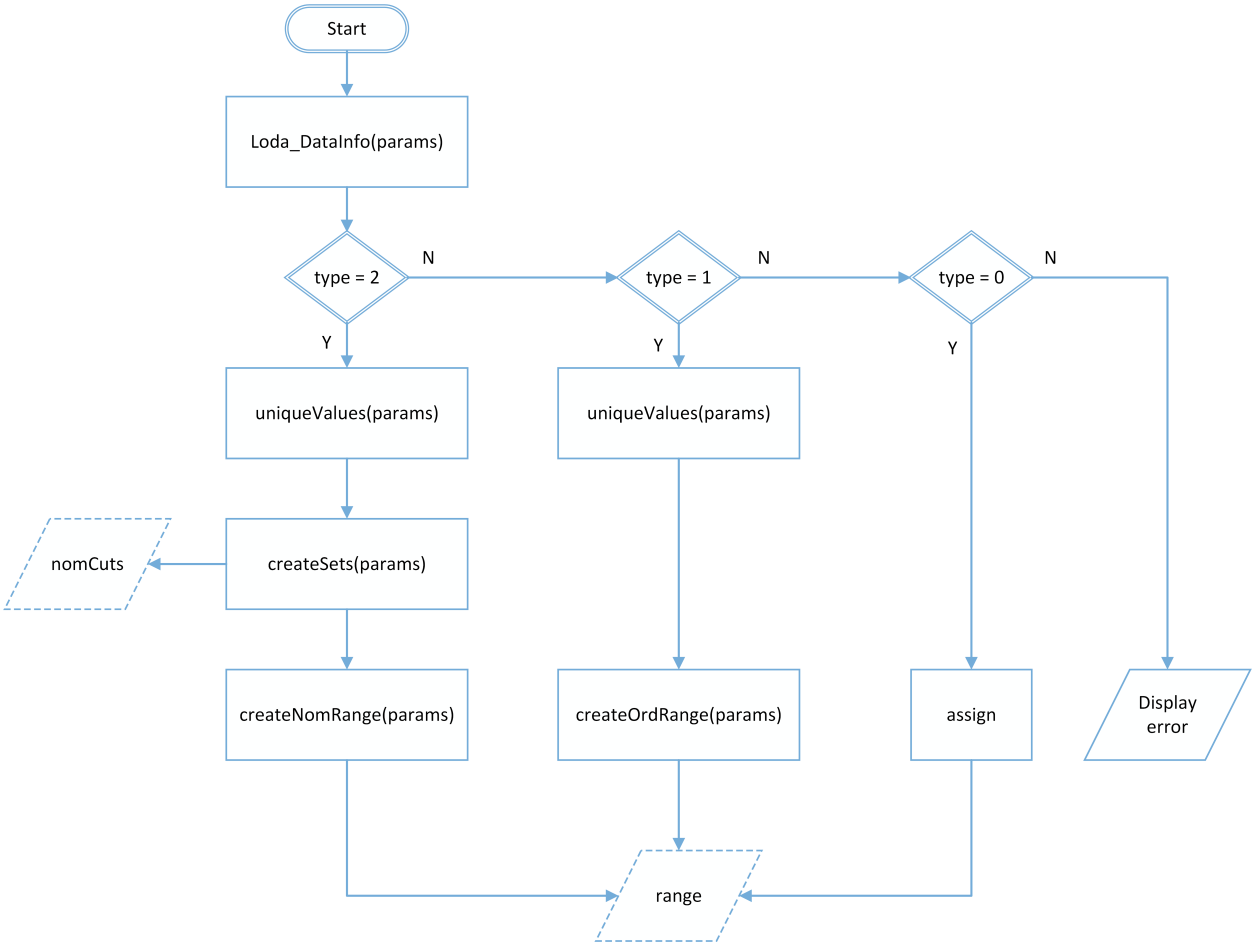
- CreateSimulationData.h
- CreateSimulationData.cpp

2.4.5 DataInfo Class Reference

The purpose of DataInfo class is generating the searching information for each covariant.

- Continuous variable:
  - Assign cut range $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ for available number $x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Ordinal variable:
  - Get unique value and find the corresponding cut range
  - Eg: Cut range $[1, 2, 3, 4, 5]$ for $x \in \{0, 1, 2, 3, 4\}$
- Nominal variable:
  - Get unique value and find the corresponding cuts and cut range
  - Eg: For $x \in \{0, 1, 2\}$, all possible subsets are $\{\}, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}$. However, we only choose the short ones. As a result, inside the "nomCuts" we have $\{\}, \{0\}, \{1\}, \{2\}$.

Flowchart of DataInfo class:



Private Attributes:

- int varNo : variable no.
- int varType : variable type, 0 for continuous, 1 for ordinal, 2 for nominal
- vector<int> range : vector for store cut range
- vector<set<int>> nomCuts : vector for store nominal cuts

Public Member Functions:

- void **load_DataInfo** (int no, int type, vector<int> &dataSetColumn) : method works as constructor
- int **getVarNo** () : return variable no.
- int **getVarType** () : return variable type
- int **getCutSize** () : return no. of cuts
- bool **nomContains** (int x, int index) : return if nomCuts[index] contain x
- int **getRange** (int i) : return range no. i
- void **printVarInfo** () : print all variable info. in each object

- void **printSet** (int i) : print set i

Private Member Functions

- vector<int> **uniqueValues** (vector<int> &vectorIn)
- vector<set<int>> **createSets** (vector<int> vectorIn)
- vector<int> **createNomRange** (int i) : create cut range for nominal variable
- vector<int> **createOrdRange** (vector<int> x) : create cut range for percentile continuous/ordinal variable

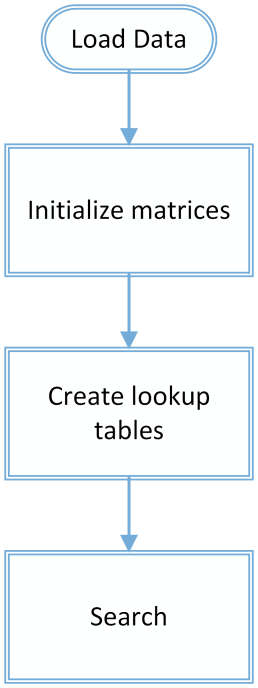
More information about this class can be found from the following files:

- DataInfo.h
- DataInfo.cpp

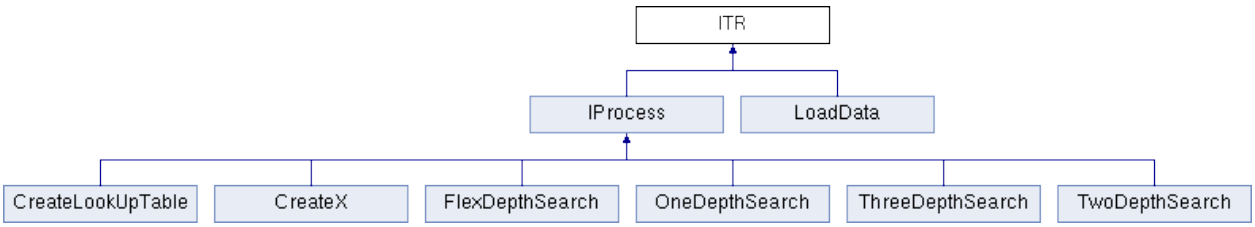
2.4.6 ITR Class Reference

The main purpose of the abstract class is store preprocessed data and implement search.

Flowchart for ITR class:



Inheritance diagram for ITR:



Protected Attributes

- double T0 : sum of treatment 0
- unsigned int** var_A : 2D matrix for A (sample_Size × action_Size)
- double** var_Y : 2D matrix for A (sample_Size × y_Size)
- bool*** table_X : 3D lookup table (var_Size × cut_Size[i] × sample_Size)
- vector<DataInfo *> info: 1D variable info. vector (var_Size)
- unsigned int* cut_Size: 1D cut size vector (var_Size)
- static vector<**Res** *> searchResults: vector for searching results

Private Attributes

- static Data* mydata
- static vector<unsigned int> var_Type
- static vector<vector<unsigned int>> var_X

Public Member Functions

- **ITR** (): constructor
- virtual void **process** ()
- static void **setData**(**Data*** data)
- static void **setVarType**(vector<unsigned int> X)
- static void **setX**(vector<vector<unsigned int>> X)
- static **Data*** **getData**();
- static vector<vector<unsigned int>>& **getX**();
- static double **getT0**();
- static vector<unsigned int> **getVarType**();
- static vector<**Res** *> **getSearchResults**();

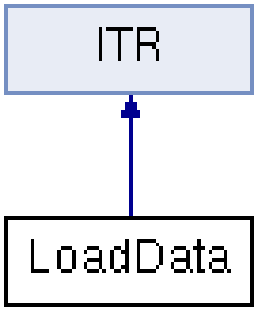
More information about this class can be found from the following files:

- ITR.h
- ITR.cpp

2.4.7 LoadData Class Reference

The purpose of LoadData class is for load data

Inheritance diagram for LoadData:



Private Attributes:

- **Data*** m_data

Public Member Functions

- **LoadData** (**Data** *data) : constructor
- void **process** () : overload

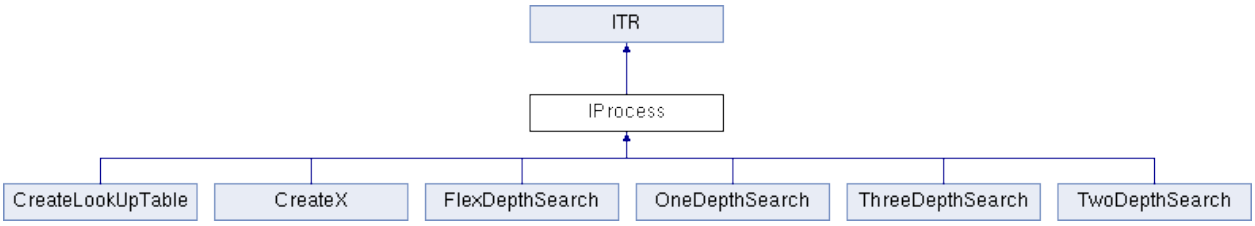
More information about this class can be found from the following files:

- LoadData.h
- LoadData.cpp

2.4.8 IProcess Class Reference

IProcess is an abstract class for processing data.

Inheritance diagram for IProcess:



Protected Attributes:

- **ITR*** myITR

Public Member Functions

- **IProcess** (**ITR** *itr) : constructor
- virtual void **process** () : virtual function

More information about this class can be found from the following files:

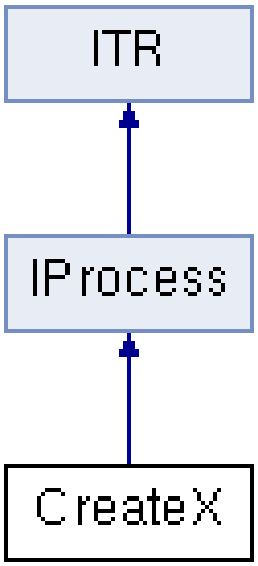
- IProcess.h
- IProcess.cpp

2.4.9 CreateX Class Reference

The purpose of CreateX class is for preprocessing data

- Convert double to unsigned int
- Combine variables

Inheritance diagram for CreateX:



Public Member Functions

- **CreateX** (**ITR** *itr) : constructor
- void **process** () : overload

Private Member Functions

- vector<vector<unsigned int>> **percentileVec**(vector<vector<double>> const &vectIn);
- vector<unsigned int> **percentileVec**(vector<double> const &vectIn);
- std::map<double, unsigned int> **percentileMap**(vector<double> const &vectorIn);
- double **percentile**(double len,double index);
- unsigned int **assignPercentile**(double p);
- vector<vector<unsigned int>> **combineData**();

More information about this class can be found from the following files:

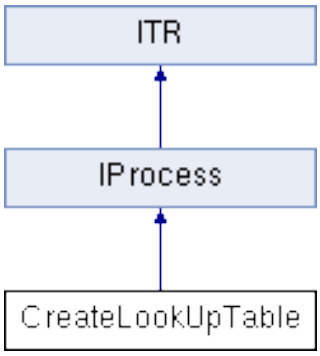
- CreateX.h

- CreateX.cpp

2.4.10 CreateLookUpTable Class Reference

The purpose of CreateLookUpTable class is for create lookup table.

Inheritance diagram for CreateLookUpTable:



Private Attributes:

- unsigned int actionIndex;
- unsigned int sampleSize;
- unsigned int varSize;
- unsigned int actionSize;
- unsigned int ySize;

Public Member Functions

- **CreateLookUpTable** (**ITR** *itr, unsigned int index=0) : constructor, index for select 0 action
- void **process** () : overload

Private Member Functions

- void init() : initial dynamic allocation
- void load_CutSize() : load cut size
- void init_TableX()
- void load_table_X(const vector<vector<unsigned int>> &data_X)
- void load_Action(const vector<vector<unsigned int>> &data_A)
- void load_Y(const vector<vector<double>> &data_Y)
- void cleanAll() : delete dynamic allocation
- double sumTreatmentCal() : sum Y given A = index

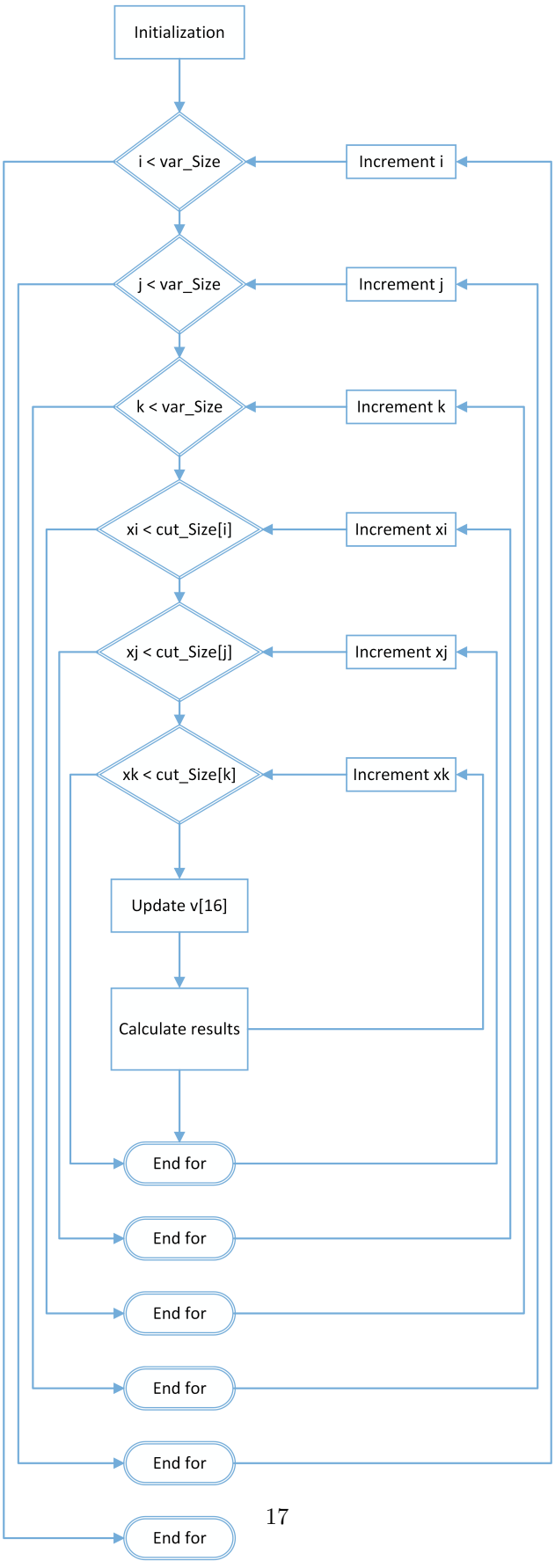
More information about this class can be found from the following files:

- CreateLookUpTable.h
- CreateLookUpTable.cpp

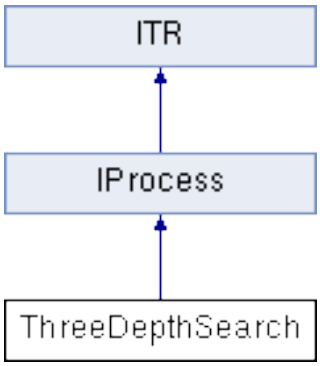
2.4.11 ThreeDepthSearch Class Reference

The purpose of ThreeDepthSearch class is for 3 depth comprehensive search.

Flowchart for ThreeDepthSearch class:


Initialization
i < var_Size
Increment i
j < var_Size
Increment j
k < var_Size
Increment k
xi < cut_Size[i]
Increment xi
xj < cut_Size[j]
Increment xj
xk < cut_Size[k]
Increment xk
Update v[16]
Calculate results
End for
End for
End for
End for
End for
End for

Inheritance diagram for ThreeDepthSearch:



Private Attributes:

- unsigned int depth;
- bool isPrint;
- unsigned int sampleSize;
- unsigned int varSize;

Public Member Functions

- **ThreeDepthSearch** (**ITR*** itr, bool print = true) : constructor
- void **process** () : override function

Private Member Functions

- void **searchPrint**() : print results
- vector<**Res** *> **searchOutput**() : output results

More information about this class can be found from the following files:

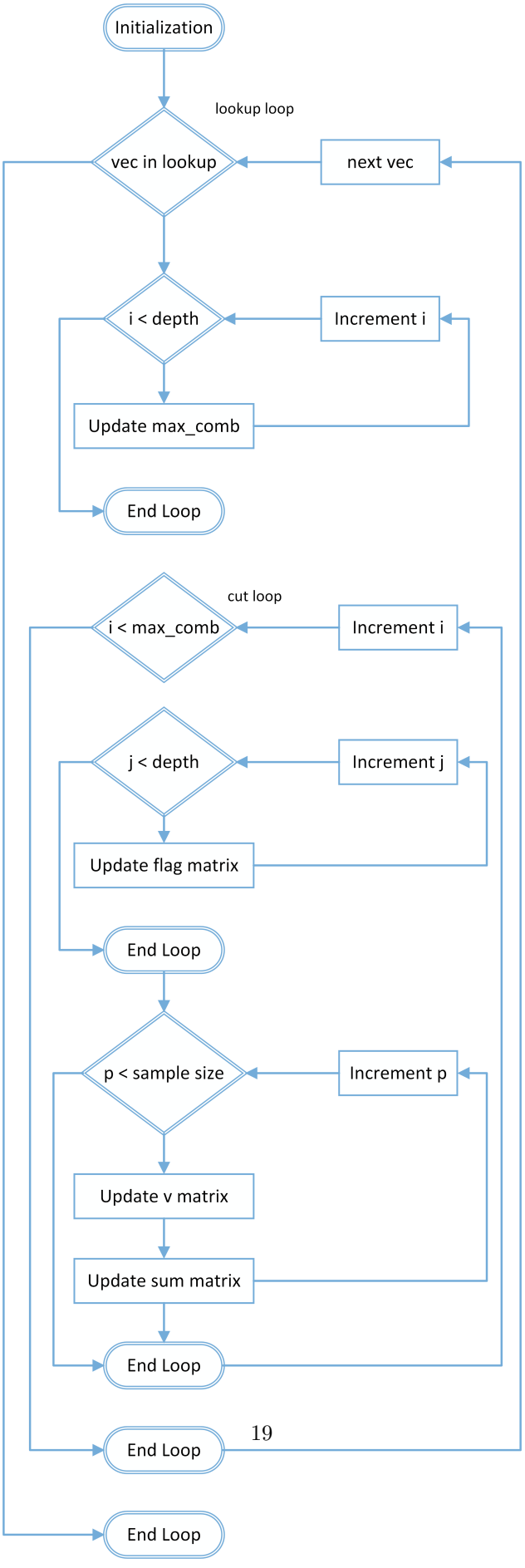
- ThreeDepthSearch.h
- ThreeDepthSearch.cpp

2.4.12 FlexDepthSearch Class Reference

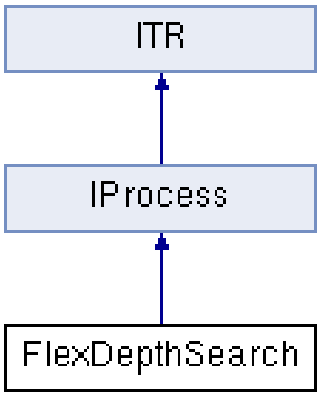
The purpose of FlexDepthSearch class is for flexible depth comprehensive search. It is mainly comprised of three loops:

- Outer loop: loop through all variable combinations
- Middle loop: loop through all possible cuts for given variable(s)
- Inner loop: loop through samples for searching

Flowchart for FlexDepthSearch class:


Initialization
lookup loop
vec in lookup
next vec
i < depth
Increment i
Update max_comb
End Loop
cut loop
i < max_comb
Increment i
j < depth
Increment j
Update flag matrix
End Loop
p < sample size
Increment p
Update v matrix
Update sum matrix
End Loop
End Loop
End Loop

Inheritance diagram for FlexDepthSearch:



Private Attributes:

- unsigned int depth;
- bool isPrint;
- unsigned int sampleSize;
- unsigned int varSize;
- vector<unsigned int> varType;
- vector<vector<int>> lookup;

Public Member Functions

- **FlexDepthSearch** (**ITR*** itr, unsigned int d, bool print = true) : constructor, d for search depth
- void **process** () : override function

Private Member Functions

- vector<vector<int>> **combine**(int n, int k) : create lookup for outer loop
- void **searchPrint**() : print results
- vector<**Res** *> **searchOutput**() : output results

More information about this class can be found from the following files:

- FlexDepthSearch.h
- FlexDepthSearch.cpp

2.4.13 Res Class Reference

Private Attributes:

- float result: results
- vector<int>* index: variable combination information
- int direction: cut direction information

Public Member Functions

- **Res** (double res, vector<unsigned int> ind, int dir): construction
- float **getRes** (): return result
- vector<unsigned int> **getIndex** (): return variable combination
  - Eg: 1 depth search: {1 2} means var 1 with cut 2
  - Eg: 2 depth search: {1 2 3 4} means var 1 with cut 2, var 3 with cut 4
- int **getDirection** (): return cut directions
  - Eg: 1 depth search: 1 means satisfy criteria, 0 means not satisfy criteria
  - Eg: 2 depth search: 0 means not satisfy criteria 1 and 2, 1 means satisfy criteria 2 but not satisfy criteria 1, 2 means satisfy 1 but not satisfy criteria 2, 3 means satisfy criteria 1 and 2

More information about this class can be found from the following files:

- Res.h
- Res.cpp

3 Test

In this simulation, we proposed 3000 samples, each with total 35 variables. The 35 variables are comprised of 25 continuous variables, 5 ordinal variables (range 0 to 4), and 5 nominal variables (range 0 to 4). A binary treatment is assigned uniformly for each patient. The response is assigned randomly to the range of 0 to 100.

3.1 System information

- System Type: System Type x64-based PC
- Processor: Processor Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz, 2401 Mhz, 2 Core(s), 4 Logical Processor(s)
- Installed Physical Memory (RAM): 8.00 GB

3.2 Run-time performance test

- Fixed step search (ThreeDepthSearch)
  - Print: around 60s
  - Output: around 50s
- Flexible step search (FlexDepthSearch) with depth = 3
  - Print: around 90s
  - Output: around 75s

3.3 Unit testing

For unit testing "googletest" is implemented by using Microsoft Visual Studio. Python is employed for result validation.

Three test datasets have been used for testing.

- Simple test data (1 ord)
- CSV test data (3 cont, 3 ord, 3 nom)
- Simulated test data (2 cont, 3 ord, 4 nom)

30 tests for 8 test cases haven been run for testing.

Store 100 samples, 9 variables(3 continuous variables, 3 continuous variables, 3 nominal variables)

1. DataInputTest
   1.1 TestSimpleInputData: validate input data for simple test data
   1.2 TestReadCSV_Size: validate data size for CSV test data
   1.3 TestReadCSV_Value: validate input data for CSV test data
   1.4 TestSimulationData_Size: validate data size for simulated data
2. LoadDataTest

2.1 SimpleTestData: validate "class LoadData" for simple test data

2.2 CSVData: validate "class LoadData" for CSV test data

2.3 SimulationData: validate "class LoadData" for simulated data

3. CreateXTest

3.1 SimpleTestData: validate "class CreateX" for simple test data

3.2 CSVTestData: validate "class CreateX" for CSV test data

3.3 SimulationData: validate "class CreateX" for simulated data

4. CreateLookUpTableTestTest

4.1 SimpleTestData_Part_I: validate DataInfo for "class CreateLookUpTable" with simple test data

4.2 SimpleTestData_Part_II: validate lookup table for "class CreateLookUpTable" with simple test data

4.3 CSVData_Part_I: validate DataInfo for "class CreateLookUpTable" with CSV test data

4.4 CSVData_Part_II: validate lookup table for "class CreateLookUpTable" with CSV test data

4.5 SimulatedData: validate DataInfo for "class CreateLookUpTable" with simulated data

5. OneDepthSearchTest

5.1 SimpleTestData_Part_I: validate "class OneDepthSearch" for print with simple test data

5.2 SimpleTestData_Part_II: validate "class OneDepthSearch" for output with simple test data

5.3 CSVData_Part_I: validate "class OneDepthSearch" for print with CSV test data

5.4 CSVData_Part_II: validate "class OneDepthSearch" for output with CSV test data

5.5 SimulationTestData: validate "class OneDepthSearch" for output with simulated data

6. TwoDepthSearchTest

6.1 CSVTestData: validate "class TwoDepthSearch" for output with CSV test data

6.2 SimulationTestData: validate "class TwoDepthSearch" for output with simulated data

7. ThreeDepthSearchTest

7.1 CSVTestData: validate "class ThreeDepthSearch" for output with CSV test data

7.2 SimulationTestData: validate "class ThreeDepthSearch" for output with simulated data

8. FlexDepthSearchTest

8.1 CSVTestData_1D: validate "class FlexDepthSearch" for depth 1 with CSV data

8.2 CSVTestData_2D: validate "class FlexDepthSearch" for depth 2 with CSV data

8.3 CSVTestData_3D: validate "class FlexDepthSearch" for depth 3 with CSV data

8.4 SimulatedTestData_1D: validate "class FlexDepthSearch" for depth 1 with simulated data

8.5 SimulatedTestData_2D: validate "class FlexDepthSearch" for depth 2 with simulated data

8.6 SimulatedTestData_3D: validate "class FlexDepthSearch" for depth 3 with simulated data

4 Future work

1. Add more recommendation algorithms
2. Add more data preprocessing methods, eg. data cleaning method