ITR 0.6.0 User Guide

Haoda Fu[1]
Jie Xue[2]
Bo Zhang[3]

[1]fu_haoda@lilly.com
[2]xue_jie@lilly.com
[3]zhang_bo3@lilly.com

Contents

Chapter 1

Introduction

Precision medicine is a medical model that proposes the customization of healthcare, with medicial 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.

Assume that there are N subjects from a large population. For sample i, A_i is the treatment assignment, Y_i is the response where larger value is better, and X_i is vector of covariates. Let (Y, A, X) be the generic random variable $\{(Y_i, A_i, X_i)\}$, $\mathcal{P}$ be the corresponding distribution, and E be the expectation with respect to $\mathcal{P}$. Let $\mathcal{D}(\cdot)$ be a treatment recommendation, i.e,

$$\mathcal{D}(\cdot) : \chi \to A, \quad X_i \in \chi,$$

and $\mathcal{P}^{\mathcal{D}}$ be the distribution of (Y, A, X) given $A = \mathcal{D}(X)$. Define

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

where

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

The quantity $E^{\mathcal{D}}(Y)$ can be used to compare different treatment recommendations. For instance, assume that we have two doctors, Adam and Barry, each having a treatment rule. Doctor Adam gives patients treatment 1 if $X \geq 2$ and treatment 0 otherwise, denoted by $\mathcal{D}_A$, and doctor Barry give patients treatment 1 if $X \geq 3$ and treatment 0 otherwise, denoted by $\mathcal{D}_B$. Over the following population, we have

$$E^{\mathcal{D}_A} = \frac{1}{10} \times \frac{1}{0.5} (1 \times 1 + 0 \times 2 + 0 \times 3 + 0 \times 4 + 0 \times 5 + 0 \times 3 + 1 \times 3 + 1 \times 3 + 1 \times 3 + 1 \times 3) = 2.6$$

Table 1.1: Comparison of treatment recommendations between Doctors Adam and Barry

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

and

$$E^{\mathcal{D}_B} = \frac{1}{10} \times \frac{1}{0.5} \left(1 \times 1 + 1 \times 2 + 0 \times 3 + 0 \times 4 + 0 \times 5 + 0 \times 3 + 0 \times 3 + 1 \times 3 + 1 \times 3 + 1 \times 3\right) = 2.4$$

Therefore, we conclude doctor Adam's treatment recommendation is better than doctor Barry's treatment recommendation. When there are more than two treatment rules, the objective is to find $\mathcal{D}_o$ such that

$$\mathcal{D}_o \in \arg\max_{\mathcal{D} \in R} E^{\mathcal{D}}(Y) = \arg\max_{\mathcal{D} \in R} E\left[\frac{I_{A=\mathcal{D}(X)}}{p(A|X)} Y\right] \tag{1.1}$$

where R is the space of possible treatment recommendations. Alternatively, the problem can be equivalently expressed as

$$\mathcal{D}_o \in \arg\min_{\mathcal{D} \in R} E\left[\frac{I_{A \neq \mathcal{D}(X)}}{p(A|X)} Y\right]. \tag{1.2}$$

Chapter 2

Installing ITR

ITR is implemented in C++ and exported as a module to R using Rcpp. This chapter describes the steps to install ITR both as a standalone C++ library and a R package.

2.1 As a C++ library

To build ITR as a standalone C++ library, user needs to have `cmake` version 3.12 or later installed. Assume that the code has been unpacked in `/path/to/ITR`, and will be built in `/path/to/build` and installed to `/path/to/install`, execute the following steps

```
> cd /path/to/build
> cmake /path/to/ITR -DCMAKE_INSTALL_PREFIX=/path/to/install
> make
> make install
```

This will install header files to `/path/to/install/include/itr` and static library `libitr.a` to `/path/to/install/lib`. The package has been tested with `gcc` version 8.1.0 or later, and LLVM 9.1.0 or later. User could specify which compiler to use by prefixing `CC=/path/to/c_compiler CXX=/path/to/cpp_compiler` to the `cmake` configuration line.

2.2 As a R module

To install ITR as a R module, user issues

```
> R CMD INSTALL /path/to/ITR
```

This will install the package for all users on the computer. If the user does not have root access or wishes to install the package locally at `/path/to/local/R/library`, it can be achieved by running

```
> R CMD INSTALL -l /path/to/local/R/library /path/to/ITR
```

If the library is installed locally, one needs to load the package by specifying the `lib.loc` option:

```
library('ITR', lib.loc='/path/to/local/R/library')
```

The package requires C++14, so one may encounter an error during the installation

```
Error in .shlib_internal(args) :
  C++14 standard requested but CXX14 is not defined
```

If that is the case, create a personal ∼/.R/Makevars file with the following lines

```
CXX14 = g++ # or the full path to your C++14 compliant compiler
CXXFLAGS = -O2 -g $(LTO)
CXXPICFLAGS = -fPIC
CXX14STD = -std=c++14
```

Chapter 3

Using ITR

This chapter describes how to use ITR as a C++ library and a R module. There are three steps of using ITR:

1. Register the data set to be analyzed with ITR.
2. Create an analyzer object.
3. Perform the analysis and query the results.

3.1 Register data set

Any data set wants to be analyzed must be registered by calling the function

```
size_t register_data(const std::string &input)
```

The input specifies the path to the data set in the csv format. The function returns an identifier of the data set which can be passed to the analyzer object discussed later.

There are a few requirements on the csv input file. The first line of the file must be a header. The first column is the subject identifier, followed by continuous variables (labeled as `cont*`), ordinal variables (labeled as `ord*`), nominal variables (labeled as `nom*`), actions (labeled as `A*`), responses (labeled as `Y*`), and condition probability $P(A = a_i|X_i)$. The ITR library is case insensitive to these labels. An example of the data set looks the following

```
ID,Cont1,Cont2,Cont3,Ord1,Ord2,Ord3,Nom1,Nom2,Nom3,A,Y,P(A=1|X)
1,20.0871,15.6859,135.029,4,2,0,4,3,3,0,31.2112,0.5
2,284.225,5.51997,74.278,0,3,3,2,3,3,0,20.8458,0.5
3,62.2001,75.9237,11.6757,1,3,3,4,1,3,0,94.7321,0.5
```

3.2 Create analyzer object

There are two types of analyer objects in ITR: comprehensive search (CS) and angle-based classifier (ABC). The constructor for CS takes two parameter:

```
CompSearch(unsigned depth, unsigned nthreads)
```

The first parameter specifies the depth of the search and the valid values are 1, 2, and 3. The second parameter specifies the number of threads to use. The value is internally truncated up to the number of available cores in the system. If the value is set to 1, the computation is performed sequentially.

The constructor for ABC takes four parameters:

```
AngleBasedClassifier(double c, double lambda, const std::string kernel,
                     unsigned threads)
```

The first parameter is a positive real number for the large-margin loss function. The second parameter is the weight of the penalty term in the objective function. The third parameter is a string object describing the kernel function and its parameters. The valid values are: (1) `rbf xxx` for the radial basis function kernel where the kernel parameter σ is set to `xxx`. (2) `poly xxx yyy` for the polynomial kernel where `xxx` is the shift and `yyy` is the degree of the polynomial. If `yyy` is set to 1, it is simply the linear kernel. The last parameter is the number of threads to use.

3.3 Perform analysis and query results

To analyze any registered data set, one first calls the `preprocess` function

```
void preprocess(size_t i)
```

passing in the identifier of the data set returned by the `registeer_data` function. Afterwards, the analysis can be performed by calling the `run` method of the analyzer object. For the CS object, the `run` method takes no parameter. For the ABC object, the `run` method takes three parameters to configure the underlying L-BFGS solver.

```
int AngleBasedClassifier::run(size_t maxItr, size_t m, double eps)
```

The first parameter specifies the maximum number of iterations allowed for the L-BFGS solver. The second parameter specifies the number of correction terms saved by the L-BFGS solver. The third parameter is a positive value controlling the accuracy of the solution. In terms of return values, value 0 indicates that the solution is found within the allowed number of iterations; value 1 indicates that no solution is found after the maximum allowed iterations; value -1 indicates error happens during the iteration.

Once the `run` method completes, one can query the results of the analysis. For CS, the available choices are

```
rVector CompSearch::score(size_t ntop)
```

This function reports the best scores.

```
uMatrix CompSearch::var(size_t ntop)
```

This function reports the variables associated with the best scores.

```
sVector cut(size_t i)
```

This function reports the cut value associated with the ith best score.

```
sVector dir(size_t i)
```

This function reports the cut direction associated with the ith best score.

For ABC, the available choice is

```
rVector AngleBasedClassifer::beta()
```

This function returns the solution to the nonlinear optimization problem.

3.4 C++ demo

The package comes with a demonstration program in the `demo` directory. To build it, using the configurations in Section 2.1, one performs the following steps

```
> cd /path/to/build/demo
> make demo
```

For other use, one needs to include the header file `ITR.h` in the source code that uses the library, and needs to supply `-I/path/to/install/include/itr` and `-L/path/to/install/lib -litr` to compile the code and link against the library. The output of the `demo` prorgam looks like the following

```
Score = 68.8498, X1 < 49.8351 (percentile 50), X2 >= 49.6823 (percentile 20), X7 not in 0 2
Score = 67.8728, X1 < 58.8109 (percentile 60), X6 not in 2 3 , X7 not in 2 3
Score = 67.4449, X1 < 49.8351 (percentile 50), X7 not in 0 2 , X8 not in 2
Score = 67.4299, X1 < 42.1108 (percentile 40), X5 < 3 (4 out of 5), X6 not in 2 3
Score = 67.4087, X1 < 49.8351 (percentile 50), X6 not in 0 2 , X8 not in 2
0.932404
-0.125072
-0.0889801
0.102649
-0.00148621
0.00923544
0.0340478
-0.145428
...
```

3.5 R demo

Here is an R session demonstrating the usage of the package.

```
> library('ITR')
Loading required package: Rcpp
> data_id <- register_data('sample100.csv')
> cs <- new(CompSearch, 3, 8)
> cs$preprocess(data_id)
> cs$run()
> cs$score(5)
[1] 68.84978 67.87278 67.44488 67.42991 67.40870
```

```
> cs$var(5)
     [,1] [,2] [,3]
[1,]    1    2    7
[2,]    1    6    7
[3,]    1    7    8
[4,]    1    5    6
[5,]    1    6    8
> cs$cut(2)
[[1]]
[1] "58.8109 (percentile 60)"

[[2]]
[1] "2 3 "

[[3]]
[1] "2 3 "

> cs$dir(2)
[[1]]
[1] " < "

[[2]]
[1] " not in "

[[3]]
[1] " not in "
> abc <- new(AngleBasedClassifier, 10.0, 1.0, "rbf 1e3", 1)
> abc$preprocess(data_id)
> abc$run(100, 10, 1e-1)
> abc$beta()
  [1]  9.368988e-01 -1.282671e-01 -1.012102e-01  9.934106e-02 -1.130552e-02
  [6]  5.073681e-03  2.694525e-02 -1.426019e-01  9.910970e-02 -1.102812e-01
 [11]  8.493024e-03  8.918804e-03 -2.457499e-02 -6.818580e-02 -1.817148e-02
 [16]  6.353913e-02  8.686820e-02  1.021895e-01  8.709502e-02 -2.069887e-02
 [21]  6.153464e-02 -6.866418e-02 -7.470917e-02  3.794699e-02  2.184204e-02
 [26]  8.245382e-02 -8.168155e-02 -1.002616e-02  3.766475e-02  5.580140e-02
 [31]  5.418532e-02 -9.802538e-02  1.065100e-01  1.443494e-01 -1.361683e-01
 [36]  1.331555e-01 -1.286964e-01  1.298254e-01  2.617285e-02 -3.220321e-02
 [41] -1.019195e-01 -9.550773e-02 -5.108990e-02 -1.372925e-01  4.591341e-02
 [46] -1.437009e-01 -3.149146e-02  7.162312e-02 -1.464040e-01  6.264910e-02
 [51] -9.430271e-02 -1.139210e-01  8.449190e-02 -1.335786e-01  6.590237e-02
 [56]  8.748744e-02 -6.655964e-02 -2.470679e-02  7.374848e-02  6.311262e-02
 [61]  4.016364e-02  6.806053e-02 -8.459916e-02 -1.378274e-02  1.249061e-02
 [66]  9.976777e-03 -3.141552e-02  1.155593e-01 -2.821169e-02  1.761921e-02
 [71] -1.388808e-02  1.677525e-01  8.383804e-02 -9.458604e-02  1.243435e-02
 [76]  7.769387e-02 -1.573087e-01 -8.478658e-02  7.738239e-02 -2.592789e-02
 [81] -5.793699e-02 -7.709801e-02  2.415769e-06 -4.766364e-02 -9.346931e-02
 [86] -1.048874e-01  1.023146e-01 -1.096255e-01  1.240425e-01  2.887337e-02
 [91]  1.165838e-01 -7.381607e-02  3.223406e-02 -6.719682e-02  1.093244e-01
 [96]  1.235799e-01 -1.006323e-02  1.145411e-01 -9.606964e-03  5.631287e-02
[101]  5.073803e-02
>
```

Chapter 4

Implementation

This chapter shares some notes of the implementation with future developers.

4.1 Comprehensive search

Given a covariate X_i and a particular cut value c, there are two treatment recommendations.

Recommendation 1 (R_1): Give treatment 1 if $X_i < c$ and treatment 0 otherwise.

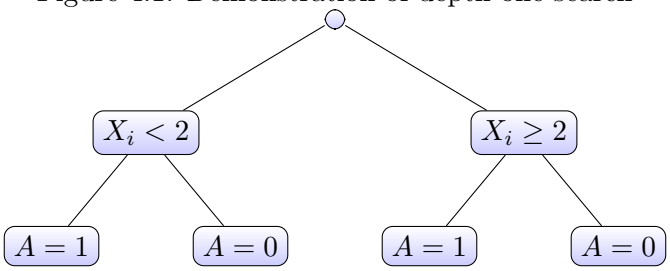
Recommendation 2 (R_2): Give treatment 1 if $X_i \geq c$ and treatment 0 otherwise.

In other words, $R_1 = I_{X_i<c}$ and $R_2 = I_{X_i \geq c}$. In the implementation, the bitmask $I_{X_i<c}$ for each cut is saved. One way to implement the search is

```
// Depth one
double v[2] = {0.0};
for (size_t i = 0; i < N; ++i) {
  v[0] += y[i] * (a[i] == m[i]);        // X < c
  v[1] += y[i] * (a[i] == 1 - m[i]);    // X >= c
}

// Depth two
double v[4] = {0.0};
for (size_t i = 0; i < N; ++i) {
  // X1 < c1 and X2 < c2
  v[0] += y[i] * (a[i] == m1[i]) * (a[i] == m2[i]);

  // X1 < c1 and X2 >= c2
  v[1] += y[i] * (a[i] == m1[i]) * (a[i] == 1 - m2[i]);

  // X1 >= c1 and X2 < c2
  v[2] += y[i] * (a[i] == 1 - m1[i]) * (a[i] == m2[i]);

  // X1 >= c1 and X2 >= c2
  v[3] += y[i] * (a[i] == 1 - m1[i]) * (a[i] == 1 - m2[i]);
}
```

As the depth of search increases, this implementation becomes more cumbersome. However, if the package is to be deployed on GPUs, the above implementation can be revised to utilize vectorization. Currently, the package is intended to run on multicore computers, and the computation is organized differently which we explain next.

Figure 4.1: Demonstration of depth one search



Consider the depth one case shown in Figure 4.1. Recall that the bitmask of $X_i < 2$ is stored as a vector. For each sample j being processed, there are four combinations depending on the value of A_j and the bitmask $X_{ij} < c$. This means, the corresponding Y_j can be sorted into four buckets. For instance, the far left bucket corresponds to $X_{ij} < c$ and $A_j = 1$ and can therefore be indexed as bucket 3 $(=(11)_2)$. The far right bucket can be indexed as bucket 0 $(= (00)_2)$ and B_k, $k = 0, 1, 2, 3$ denote the sets of Y_j in each bucket and S_{T_0} is a set of Y_j where j satisfies $A_j = 0$. Notice that the sets have the following relationship,

$$B_3 - B_2 + T_0 = \{A = 1 \cap X_i < 2\} - \{A = 0 \cap X_i < 2\} + \{A = 0\} = \{A = 1 \cap X_i < 2\} + \{A = 0 \cap X_i \geq 2\}$$

Let T_0 be

$$T_0 = \sum_{j:A_j=0} Y_j$$

If the treatment recommendation is $I(X_i < 2)$, the recommendation is treatment 0, the value of $\sum_j I\{A_j = I(X_{ij} < 2)\}Y_j$ can be calculated as the sum of Y_j where $j \in \{B_3 - B_2 + T_0\}$. Similarly, one can verify that if the treatment recommendation is $I(X_i \geq 2)$, the sum can be based on the Y_j where $j \in \{B_1 - B_0 + T_0\}$.

We may notice that the $\{B_3 - B_2 + T_0\} = \{B_3 + B_0\}$ and $\{B_1 - B_0 + T_0\} = \{B_2 + B_1\}$. Therefore, we do not need to calculate T_i in this example. However, when the depth is equal to 3, the above calculation will be more efficient.