User Guide for `TOILC`

Arthur Américo

Contents

1 Introduction

This is a user guide to TOILC - Trading Optimizer under Information Leakage Constraints. The optimizer consists of a Python 3 script, and the code is freely available at [**?**].

This optimizer was built as a solution to the problem of maximizing trading activity while bounding the amount of information leakage. The details are discussed in the paper *Defining and Controlling Information Leakage in US Equities Trading*, to appear in the 2024 *Privacy Enhancing Technologies Symposium (PETS)*, and available as a preprint in [1].

2 The Linear Programming Model

This script tackles our problem by solving two LPs, whose objective and constraints are obtained by channel operations as detailed in the main article. A schematic illustration is given in Figure 1.

The composition depicted in Figure 1 can be modelled as a channel from X to $\tilde{X}$ as

$$System = (I \parallel (Aux Alice))Public. \tag{1}$$

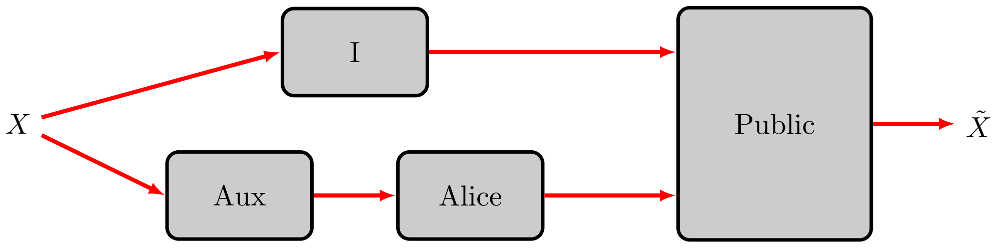

Figure 1: A schematic illustration of the model.

From p_X and (1), it is easy to obtain the value of $p_{\tilde{X}}$. Recall that $System(y|z) = p_{\tilde{X}|x}(y)$. Overloading the notation and letting p_X also stand for the probability vector$(p_X(x_1), \ldots, p_X(x_2))$, we can obtain the joint distribution matrix $P_{X,\tilde{X}}(x,y) = p_{X,\tilde{X}}(x,\tilde{x})$ by

$$P_{X,\tilde{X}} = D_{p_X} System, \tag{2}$$

where D_{p_X} is the diagonal matrix whose entries are the values of p_X, i.e.

$$D_{p_X} = \begin{pmatrix} p_X(x_1) & 0 & \cdots & 0 \\ 0 & p_X(x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_X(x_n) \end{pmatrix}$$

The row vector $p_{\tilde{X}}$ can be simply obtained by marginalizing the columns of $P_{X,\tilde{X}}$, which is equivalent to multiplying $P_{X,\tilde{X}}$ on the left side by the row vector $\vec{1} = (1, 1, \ldots, 1)$

$$p_{\tilde{X}} = \vec{1} P_{X,\tilde{X}}. \tag{3}$$

Notice that all the channel operations we did above and all the operations to obtain $p_{\tilde{x}}$ are linear operations.

2.1 Solving the First Linear Programming Problem

The first LP is used to obtain the maximum expected value of Alice's actions, given the variables of the variables of the problem. We then use this variable to minimiza a proxy to the variance of Alice's actions on a second LP.

Letting A be the r.v. of Alice's actions, we may derive the vector p_A, similarly to (2) and (3), by

$$p_a = \vec{1} D_{p_X} Aux Alice.$$

Thus, we maximize the expected value of Alice's actions given the preferential privacy constraints, by solving the following LP.

- **maximize:** $\mathbb{E}[A] = \sum_{a \in \mathcal{A}} a p_A(a)$
- subject to

- $\sum_{a \in \mathcal{A}} Alice(a|o) = 1$ $\quad \forall o \in \mathcal{O}$
- $Alice(a|o) \geq 0$ $\quad \forall a \in \mathcal{A}, o \in \mathcal{O}$
- $e^{-\epsilon} p_X(x) \leq p_{\tilde{X}}(x) \leq e^{\epsilon} p_X(x)$ $\quad \forall x \in \mathcal{X};\ x_l \leq x \leq x_h$
- $\sum_{x \geq x_h} p_{\tilde{X}}(x) \leq m\delta$

Where

$$x_l = \max\left\{ x \Big| \sum_{x' < x} p_X(x') \leq \delta_L \right\},\ x_h = \min\left\{ x \Big| \sum_{x' > x} p_X(x') \leq \delta \right\}$$

and the values of δ, $\delta_L \in [0,1]$ and $m \geq 1$ are given by the user.

The first two constraints guarantee that *Alic*e is indeed a channel, the third one is the differential privacy condition, and the last one is the bound on the cumulative distribution of the right tail, for which the differential privacy bounds are ignored

2.2 Solving the Second Linear Programming

Let E_{max} denote the solution to the LP in Section 2.1 — i.e., the maximum expected value of Alice's actions given the differential privacy constraints. The variance of A can be calculated as

$$Var(A) = \mathbb{E}[(A - \mathbb{E}[A])^2].$$

This quantity, unfortunately, is concave w.r.t. the entries of Alice. Thus, we use a proxy by substituting E_{max} for $\mathbb{E}[A]$ above, obtaining

$$\mathbb{E}[(A - E_{max})^2] = \sum_{a \in \mathcal{A}} p_A(a)(a - E_{max})^2,$$

which is linear w.r.t. Alice.

Finally, we minimize this quantity in a second LP. In order to do so, we add another constraint, guaranteeing that the value of $\mathbb{E}[A]$ is at least tE_{max}, for some $t \in [0,1]$. Notice that in the case t = 1, this constraint becomes $\mathbb{E}[A] = E_{max}$, and the LP below minimizes the actual variance.

- **maximize:** $\sum_{a \in \mathcal{A}} p_A(a)(a - E_{max})^2$
- subject to
  - $\sum_{a \in \mathcal{A}} Alice(a|o) = 1$ $\quad \forall o \in \mathcal{O}$
  - $Alice(a|o) \geq 0$ $\quad \forall a \in \mathcal{A}, o \in \mathcal{O}$
  - $e^{-\epsilon} p_X(x) \leq p_{\tilde{X}}(x) \leq e^{\epsilon} p_X(x)$ $\quad \forall x \in \mathcal{X};\ x_l \leq x \leq x_h$
  - $\sum_{x \geq x_h} p_{\tilde{X}}(x) \leq m\delta$
  - $\mathbb{E}[A] \geq tE_{max}$

3 The Script Behaviour and User Options

The problem is uniquely determined by p_X, Aux and $Public$. All these have default values in the code, but can be changed by options in the command line. A summary of all options that can be given to the script is given in Section 3.5

After the Script runs, it will print the values of $\mathbb{E}[X]$, $\mathbb{E}[\tilde{X}]$, $\mathbb{E}[A]$ and $Var(A)$, plot the solution on the screen, and save this plot in a aptly named pdf file. It is possible to change the printing behaviour of the script using the commands in Table 2. If the script is being used as a module instead of being run in the command line, it will return these values as attributes of an object (see Section 4).

3.1 The distribution p_X

The script's default behaviour is to let the range of X to be the integers from 0 to 50. It then performs 10^7 samples from a gaussian distribution centred on the middle value of the range and with standard deviation equal to 5. To change these values, the user can use the following options:

- `-range` n : changes the range of X to $(0, 1, \dots, n-1)$
- `-std` n: changes the standard deviation of the gaussian distribution to n
- `-samples` n : changes the number of samples from the gaussian to 10^n

Thus, if you want to run the script with X having 100 possible values, and a p_X obtained by 10^5 samples of a gaussian with standard deviation of 7, you can type in your command line

```
>python toilc.py -inputSize 100 -std 7 -samples 5
```

If the user has a distribution in a text file, he can load it by using `-distX filename.txt`. The file must consist of a single row or a single column of numbers, and it will be normalized into a probability distribution.

3.2 The channel Aux

The channel aux is controlled by the option `-noise`, which receives an argument from 0 to 1. If the user types into the command line

```
> python toilc.py -noise 0
```

then Aux is a perfect channel, meaning Alice has complete information about the realization of X. That is, Aux will be the channel

Aux	0	1	$\cdots$	n
0	1	0	$\cdots$	0
1	0	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	0	0	$\cdots$	1

this is also the channel used by default, when no `-noise` argument is given.

On the other hand, If the user types into the command line

```
> python toilc.py -noise 1
```

then the channel is completely noisy, and Alice receives no extra information about X. This corresponds to the channel

Aux	o
0	1
1	1
$\vdots$	$\vdots$
n	1

For values between 0 and 1, we add some discretised noise inspired on the geometric distribution. If the user inputs

```
> python toilc.py -noise p
```

for some $p \in (0,1)$, the channel is given by

$$Aux(j|i) = \alpha_i(1-p)(p)^{|i-j|},$$

where α_i is a normalizing factor, so that each row of Aux sums to one. Notice that, when p goes to 0 or 1, this channel behaves like the perfect and null channel, respectively.

As an example, if the range of X is $\{0,\ldots,3\}$, the channel obtained by setting $p = 0.5$ is

Aux	0	1	2	3
0	8/15	4/15	2/15	1/15
1	2/9	4/9	2/9	1/9
2	1/9	2/9	4/9	2/9
3	1/15	2/15	4/15	8/15

The user can also upload a file containing the aux channel by using the option `-aux filename.txt`. The file must consist only of the values of the matrix, with the rows corresponding to the inputs and the columns to the outputs, as in the channels above.

3.3 Number of Alice's actions, e^ϵ, δ, δ_L and m

The default number of Alice's actions is 20, but the user can change this number to any positive integer n by using the option `-act n`.

Similarly, the default value of e^ϵ is 1.3, but the user can change this number to any float $p \geq 1$ by using the option `-eeps p`

The default value of δ is 0.01 and of m is 1.1. The user can change these numbers to any floats $p \in [0,1]$ and $q \geq 1$ by using the options `-delta p` and `-tailMult q`, respectively. Finally, the default value of δ_L is 0 — that is, by default the program ignores the left tail, and it can be changed to any float $p \in [0,1]$ with the option `-delta p`.

3.4 The Threshold t and Ignoring the Second LP

The default threshold value t on the second LP is 1 — that is, by default, the solver will try minimizing the actual variance without diminishing the value of $\mathbb{E}[A]$. The user may change this behavior by using the command `-threshold t`, for some $t \in [0,1]$.

If the user is not interested in reducing the variance of Alice's actions, he can suppress the execution of the second LP by using the command `-noVariance`.

3.5 Other Commands and Summary

Besides the commands discussed above, there are many others that can be used to modify the behaviour of the program and what it prints. The tables below summarize these commands, including the ones introduced above. The user can also see these commands with a short description by typing into the command line

```
python toilc.py -h
```

4 Using the solver as a Module

If, instead of calling the solver in the command line, you would like to use it as a module, you can simply import it and call the `solve()` function. It receives as parameters the same commands as the ones in Tables 1 and 2 (without the leading '-') and the same types of inputs, with the difference of also accepting numpy arrays, lists or file objects for `distX`, `auxInput` and `publicInput`. Moreover, the function solve() also returns an object with the attributes listed in Table 3 below. For example, the following program runs the solver with noise 0.8 and $e^\epsilon \in \{1.1, 1.3, 1.5\}$, and prints the value of $\mathbb{E}[A]$

```
import toilc as tc

for eepsilon in [1.1, 1.3, 1.5]:
    result=tc.solve(noise=0.8, eeps=eepsilon)
    print(result.expectedAlice)
```

As another example, the following code calls the solver setting p_X to a vector generated by `numpy.random.randint` (notice that the solver normalizes the input distribution)

```
import toilc as tc
import numpy as np
```

```
p=np.random.randint(100, size=100)
tc.solve(distX=p)
```

Table 1: Commands that modify the linear program.

Command	Argument type	Default	Effect
`-inputSize n`	int	51	Makes the range of X the integers from 0 to $n-1$
`-std n`	int	10	Makes the standard deviation of X equal to n
`-samples p`	float	7	The distribution is obtained by 10^p samples
`-act n`	int	20	Makes Alice's set of actions the integers from 0 to $n-1$
`-noise p`	float	0.0	See Section 3.2
`-eeps p`	float	1.3	Sets $e^\epsilon = p$
`-threshold p`	float	1.0	Sets $t = p$
`-delta p`	float	0.01	Sets $\delta = p$
`-deltaLeft p`	float	0.0	Sets $\delta_L = p$
`-tailMult p`	float	1.1	Sets $m = p$
`-distX file.txt`	string (file path)		Imports p_X from `file.txt` and normalizes it
`-aux file.txt`	string (file path)		Imports *Aux* from `file.txt`
`-public file.txt`	string (file path)		Imports *Public* from `file.txt`
`-maxExpected`	No argument		Changes the LP objective to maximize $\mathbb{E}[\tilde{X}]$
`-noVariance`	No argument		Outputs the solution for the first LP

Table 2: Printing commands.

Command	Argument type	Effect
`-pretty`	No argument	Truncates all numbers at 3 decimal plates
`-printAux`	No argument	Prints *Aux* in the shell
`-printAlice`	No argument	Prints *Alice* in the shell
`-printAuxFile file.txt`	string (file path)	Prints *Aux* in `file.txt`
`-printAliceFile file.txt`	string (file path)	Prints *Alice* in `file.txt`
`-outputX file.txt`	string (file path)	Prints p_X and $p_{\tilde{X}}$ in `file.txt`
`-noPlot`	string (file path)	Does not plot the solution
`-pdfOutput file.pdf`	string (file path)	Saves the plot in file.pdf
`-omitSolution`	No argument	Does not print the LP's solution and other problem values

Table 3: Attributes returned by the function `solve`.

Attribute	Value
`expectedX`	$\mathbb{E}[X]$
`expectedXtilde`	$\mathbb{E}[\tilde{X}]$
`expectedAlice`	$\mathbb{E}[A]$ for the second LP
`minimumAlice`	$\mathbb{E}[A]$ for the first LP
`varianceAlice`	$Var(A)$
`cutoff`	$x_h + 1$
`cutoffLow`	x_l

References

[1] A. Americo, A. Bishop, P. Cesaretti, G. Grogan, A. McKoy, R. Moss, L. Oakley, M. Ribeiro, and M. Shokri, "Defining and controlling information leakage in us equities trading." Cryptology ePrint Archive, Paper 2023/971, 2023. https://eprint.iacr.org/2023/971.