Fall 2020
EE 236A
Christina Fragouli
Mine Dogan and Kaan Ozkara

EE236A: Linear Programming

Project II - Designing an LP Decoder for Group Testing

Due: Thursday, December 17, 2020

In this project, you will design and implement a Group Testing decoding algorithm. Like the previous one, this is also an open-ended project, which means there are no unique or "right" answers. The deadline for submitting the project is on **Thursday December 17**.
Group testing has a long history, starting from US military applications in the 1940s [4]. It has recently attracted a lot of attention in the context of Covid, as a strategy that can allow to significantly reduce the number of tests used [8, 3, 6, 9, 7]. The hope is that, group testing will not only reduce the number of tests needed, but also help achieve increased reliability. In this project, we will explore a new approach: we will take into account community structure to improve the performance of the linear program decoding group testing employs.

1 Background and Notation

The basic idea in group testing is that, instead of individually testing a biological sample say for a virus, to take several samples, pool them together, and perform a single test on all of them: if the test comes back negative, we know that none of the people tested has the virus; if the test result comes back positive, we know that at least one of the people tested is infected. We can then procceed to perform additional tests to identify the infected members. Group testing allows to significantly reduce the number of tests needed by allowing to "rule out" with a single test more than one individuals [5, 1].
More formally, the traditional group testing setup assumes a population of n members out of which some are infected. In the *probabilistic model*, each item is infected independently of all others with probability p [5, 1].
A group test τ takes as input samples from n_τ individuals, pools them together and outputs a single value: positive if any one of the samples is infected, and negative if none is infected. More precisely, let $x_i = 1$ when individual i is infected and 0 otherwise. Then the traditional group testing output

y_τ takes a binary value calculated as:

$$y_\tau = \bigvee_{i \in \delta_\tau} x_i, \tag{1}$$

where $\bigvee$ stands for the `OR` operator (disjunction) and δ_τ is the group of people participating in the test. The performance of a group testing algorithm is measured by the number of group tests $T = T(n)$ used, as well as the error probability achieved.

Group testing can be *adaptive*, where future tests are designed taking into account the outcome of past tests; and *nonadaptive*, where all tests are designed in advance; in this project, we are interested in nonadaptive testing, as this type of group testing is the most practical, since it allows to perform all the tests in parallel. Nonadaptive testing constructs, in advance, a *test matrix* $\mathbf{G} \in \{0,1\}^{T \times n}$ where each row corresponds to one test, each column to one member, and the non-zero elements determine the set δ_τ.

1.1 Community and infection models

In this project, we additionally assume a known community structure: the population can be decomposed in F disjoint sets of families, where family j has M_j members and $n = \sum_{j=1}^{F} M_j$. In the symmetric case, $M_j = M$ for all j and $n = FM$.

We will also assume a probabilistic infection model. Namely, a family is infected with probability q i.i.d. across the families. A member of an infected family is infected, independently from the other members (and other families), with probability p_j (in this project p_j is assumed to be same across all families). The rest of the families have no infected members.

1.2 Noisy testing

In the first part of the project we assume access to noiseless tests that have perfect accuracy. However, we will also consider noisy testing, where the test output is flipped with a small probability z. In particular, a test output that should be positive, flips and appears as negative with probability z (and similarly for negative test outputs). Noisy testing captures inaccuracies in testing - the vast majority of Covid tests today are “noisy” (do not have 100% accuracy).

Let $\hat{x}_i$ denote the estimate of the state of x_i after group testing. Hamming error captures the number of positions where $\hat{x}_i$ and x_i differ. Sometimes we want to distinguish between *False Negatives (FN)* and *False Positives (FP)* errors, where

$$\Pr(\text{FN}) = \Pr(\hat{x}_i = 0 | x_i = 1), \quad \Pr(\text{FP}) = \Pr(\hat{x}_i = 1 | x_i = 0). \tag{2}$$

In this project, we want you to consider two possible cases for noise, generated by the z-channel and the binary summetric channel. In the first case, you will assume that test results of some of the infected people may come back negative (false negative) with probability z. However, you will assume that the probability of false positive test results (test result of a healthy person comes back positive) is zero in this case. An illustration of this case can be seen in Figure 1.

In the second case, you will assume that test result of an infected individual can come back negative

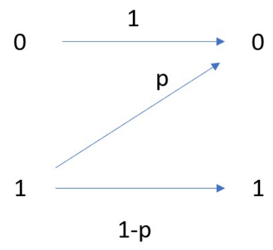

Figure 1: Illustration of z channel.

(false negative) or test result of a healthy person can come back positive (false positive) with probability z each. Hence, you need to consider both false negative and false positive results in this case. An illustration of it can be seen in Figure 2.

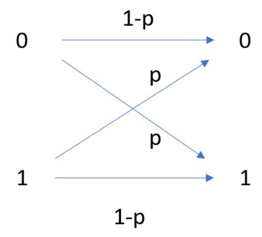

Figure 2: Illustration of binary symmetric channel.

1.3 Test Matrix Design

In this project we will use randomly constructed test matrices with constant column weight (this is the state of the art design in terms of performance); recall that each column corresponds to an individual.We will use column weight equal to $L = \lceil \frac{\ln 2T}{K} \rceil$, where $\ln 2 \approx 0.693$, T is the number of rows of the test matrix (number of tests) and K is the expected number of infected individuals. For example, if each individual is infected according to a Bernouli distribution with probability p then $K = np$. We will provide a function that generates the test matrix for you.

1.4 Decoding Methods

There exists various decoding algorithms in group testing. One is finding the smallest satisfying set **(SSS)**: in this algorithm the goal is to find the smallest set of infected individuals such that the constraints due to test outputs are satisfied. The intuition is that the infections rate is a small ratio, hence, infected individuals should be rare. Another is combinatorial orthogonal matching pursuit **(COMP)**. Here the items are assumed to be defective unless it is proven otherwise again due to test results. Yet another algorithm is the Definite Defectives **(DD)** one; this algorithm assumes that the items are non-defective unless proven otherwise due to test results.

In this project, we will use a Linear Programming Approximation to the **SSS** algorithm. An example decoder for group testing is given in (3) [2].

$$\begin{aligned}
&\text{minimize}_{\mathbf{z}} \sum_{i=1}^{n} z_i \\
&\text{subject to} \qquad \sum_{i=1}^{n} x_{ti} z_i \geqslant 1 \quad \text{when } y_t = 1 \\
&\textstyle\sum_{i=1}^{n} x_{ti} z_i = 0 \quad \text{when } y_t = 0 \\
&z_i \in \{0,1\}
\end{aligned} \tag{3}$$

where $\mathbf{z}$ is an indicator vector for defective items, x_{ti} indicates if item i is in test t and y_t is the outcome of test t. Starting from the above ILP, you can consider the associated relaxed LP, and try your own approaches from constructing an integral solution from the LP optimal solution.
Note that the integer program in (3) is just an example of a decoder in group testing and you are not restricted to use it. You may select/design a different ILP/LP for decoding.

2 Project Goal

Overview The goal of the project is to design, implement and test an LP for group testing problem in various settings. In the group testing literature a widespread assumption is that the defects in items (sickness in people in our case) are generated i.i.d with $Bernoulli(p)$ which means the probability that an item is defected is p is same for all the items independent of each other. This simple case will constitute a base for your group testing algorithm. Therefore, in the first case, you will assume that individuals are infected with i.i.d and try to implement a decoding algorithm (you can use the one given in (3)). After this base case, we will consider some more sophisticated cases.
The first is based on the following observation. There can be certain social structures in the population which create correlations we can exploit to improve our decoding algorithm. For example, if a person is sick there is a high chance that he/she will transmit the infection to the people in his/her family or if a students takes a course, he/she can infect his/her classmates.

We will also consider the case of noisy test results. When individuals are tested, errors can happen. For example, with 0.1 probability, the test result of an infected person can come back negative (false negative). An example of a decoder that can work with noisy test results is given in (4) [2].

$$\begin{aligned}
\text{minimize}_{\mathbf{z},\boldsymbol{\xi}} \quad & \textstyle\sum_{i=1}^{n} z_i + \zeta \sum_{t=1}^{T} \xi_j \\
\text{subject to} \quad & z_i \geqslant 0 \\
& \xi_t \geqslant 0 \\
& \xi_t \leqslant 1 \\
& \textstyle\sum_{i=1}^{n} x_{ti} z_i \geqslant \xi_t \quad \text{when } y_t = 0 \\
& \textstyle\sum_{i=1}^{n} x_{ti} z_i + \xi_t \geqslant 1 \quad \text{when } y_t = 1
\end{aligned} \tag{4}$$

where $\boldsymbol{\xi}$ is the vector of slack variables and parameter ζ controls the trade-off between declaring a small number of items to be defective (sparsity) and the degree to which the test outcomes are in agreement with the decoded z_i (most slack variables being zero) [2]. However, the example in (4) is for the base case (i.i.d infection). You should modify your decoding algorithm such that it can take into account community structure.

3 Outcomes

Your main contribution will be in designing an LP decoding algorithm that takes into account the community structure. You may design more than one if you would like and compare them. The output of the project should include the code, a description of your approach, and the requested plots described in the following.
A main goal is to compare benefits we get when we take into account, vs not taking into account, the community structure.

3.1 No community structure

This is the baseline case, to test your algorithm when no community structure is taken into account. We would like to plot, as a function of the number of tests used T, with $T \in [100 : 100 : 700]$ the FP, FN and Hamming error rate, for the following cases:

1. Infection probability $p = 0.1$, and $n = 1000$
2. Infection probability $p = 0.2$, and $n = 1000$

Please implement your decoder inside the function decribed in section 5.

3.2 Community structure

Assume now that infections are generated according to a community structure. In particular we are interested in the non-overlapping case, i.e. the classes/families have no joint members. We are going to consider two different use cases:

Case 1 a neighborhood consisting of $F = 200$ families each one having $M = 5$, and

Case 2 a university department consisting of $F = 20$ classes of $M = 50$ students each.

For each use case, we examine 2 different regimes about how the number of infected members scales with the entire population:

1. A case where $q = 0.1$ and $p = 0.8$
2. A case where $q = 0.2$ and $p = 0.6$.

Here q denotes the probability of a family having an infected member and p denotes the probability of getting infected in an infected family. So we have in total four different cases. To obtain results that are statistically significant, we want to average over at least 100 randomly generated instances. Our main goal is to compare the performance we can achieve in terms of FP, FN and Hamming error, if we ignore the community structure during decoding, and if we take the community structure into account, as a function of the number of tests T used. In each run, reconstruct the infection status of all members/students using: (i) your vanilla LP decoding algorithm that does not leverage the community structure and (ii) the LP decoding algorithm you propose that takes into account the community structure. Plot, as a number of T, with $T \in [100, 300, 500, 1000]$ the FP, FN and Hamming error rate.

3.3 Noisy Decoding

Consider now the same cases as in Section 3.2, but assume that tests are noisy. Describe your decoding algorithm in this case, and provide the same plots as before for both z channel and binary symmetric channel cases described in section 1.2, when the flipover probability is 0.1, and when the flipover probability is 0.2.

4 Data set

In this project, we will provide a code for you so that you can generate a random population to test your decoding algorithm. The data set generated by this code represents a population in which some of the individuals are infected. The code will create two arrays. The first array is a vector representing the individuals in the population. If the value at a specific position is 1, this means that the corresponding individual is infected. Otherwise, it means that the individual is healthy. Note that the infection probability is p. For the no community structure case, the individuals are infected with probability p i.i.d and the code generates the population under this assumption. For the community structure case, family members are correlated, therefore, if a family is infected, this increases the probability of its family members being infected. The code again generates the population under this assumption. The code also generates the test matrix. Each row of the test matrix represents a test and each column represents an individual in this population. For example, if the element at $(i,j)^{th}$ position is 1 in the test matrix, this means that the j^{th} individual is tested in i^{th} test. Finally, the code will also generate a vector that holds the test results.

5 Implementation

We provide a python file named `MyDecoder.py`. There, you will find some useful functions you can use to generate a population in different settings; it will also serve you as a template. You can implement your auxiliary functions, but please ask us first if you are going to edit the already given population generation functions. In particular you have the following functions in the python file:

5.1 Functions Provided

These functions are provided by us, please don't edit them.

- *generator*: This function generates a population for the base case, where all individuals can be infected i.i.d. The inputs are n:number of people, *prob_inf*: probability of infection, T: number of tests. The outputs are X:Test matrix; *ppl*:infection vector, if an entry is 1 then the corresponding individual is infected; y: vector of test results, if an entry is 1 then the result of the corresponding test is positive.

- *generator_nonoverlapping*: This function generates a population in case of a non-overlapping family structure. The inputs are n: total number of individuals, m: total number of families, q: probability that a family contains infected individuals, p: probability that an individual becomes infected in an 'infected' family, T: number of tests. Outputs are X: test matrix, ppl: infection vector, A: family structure matrix, y test results vector.

- *add_noise_zchannel*: Given a test vector this function adds a z-channel type noise to test results, in other words flips only happen if the original test result is positive. Inputs are y:noiseless test results and *p_noise*: probability of flipping in test results. Output is the noisy test result vector.

- *add_noise_bsc*: Given a test vector this function adds a binary channel type noise to test results, in other words flips happen independent of the original test result. Inputs are y:noiseless test results and *p_noise*: probability of flipping in test results. Output is the noisy test result vector.

5.2 Functions to Implement

These are the functions that you will implement. Make sure your implementation allow any size of population, number of families, probabilities.

- *lp*: Here you need to implement your base linear program, it should output your predictions of the infected individuals.

- *lp_nonoverlapping*: Here you need to implement a linear program considering a non-overlapping family structure, it should output your predictions of the infected individuals.

- *lp_noisy_z*: Here you need to implement a linear program considering that the test results might be flipped in a Z-channel manner, it should output your predictions of the infected individuals.

- *lp_noisy_bsc*: Here you need to implement a linear program considering that the test results might be flipped in a binary symmetric channel manner, it should output your predictions of the infected individuals.

- *lp_noisy_z_nonoverlapping*: Here you need to implement a linear program considering both a non-overlapping family structure and Z-channel noisy test results, it should output your predictions of the infected individuals.

- *lp_noisy_bsc_nonoverlapping*: Here you need to implement a linear program considering both a non-overlapping family structure and binary symmetric noisy test results, it should output your predictions of the infected individuals.

You need to implement your main code in your `MyDecoder.py` file under the `if _name_ == '_main_':` statement.

6 Grading

- This project is worth 15 points.
- Use the template file of the project to fill in the implementation of your decoder. Provide the code that you implement in the most self-sufficient manner, so that you would expect it to run on virtually any machine with Python.
- For the purposes of grading, we will check your decoders locally after submission. We may use a different dataset when grading, so make sure your decoder can work with any number of families, probabilities of infection, and noise.
- You need to submit a folder named `'group_{groupnumber}'` (`group_5` for the group with name group 5) on CCLE. Inside this folder there should be a file named `MyDecoder_{groupnumber}.py` (`MyDecoder_5.py` for group 5). Also, you should include your report in this folder.

7 Competition

We will select the decoding algorithm that performs best on different scenarios and we will give 5 bonus points to each member of the winning team. We will select the winning team by running your decoders for different infection probabilities, family and population sizes. Therefore, please don't change the parameters of generator functions that we provide and try to write your main functions in the most self-sufficient manner so that we can easily run your decoders for different settings.

References

[1] Matthew Aldridge, Oliver Johnson, and Jonathan Scarlett. Group testing: an information theory perspective. *CoRR*, abs/1902.06002, 2019.

[2] Matthew Aldridge, Oliver Johnson, and Jonathan Scarlett. Group testing: an information theory perspective. *CoRR*, abs/1902.06002, 2019.

[3] Maria Broadfoot. Coronavirus test shortages trigger a new strategy: Group screening. See `https://www.scientificamerican.com/article/coronavirus-test-shortages-trigger-a-new-strategy-group-screening2/`, May 2020.

[4] Robert Dorfman. The detection of defective members of large population. *The Annals of Mathematical Statistics*, 14:436–440, December 1943.

[5] D-Z Du and F.K. Hwang. *Combinatorial Group Testing and Its Applications.* Series on Applied Mathematics, 1993.

[6] Jordan Ellenberg. Five people. one test. this is how you get there. *NYtimes*, May 2020.

[7] Sabyasachi Ghosh et al. Tapestry: A single-round smart pooling technique for covid-19 testing. *medRxiv*, 2020.

[8] Christian Gollier and Olivier Gossner. Group testing against covid-19. See `https://www.tse-fr.eu/articles/group-testing-against-covid-19`, April 2020.

[9] Claudio M. Verdun et al. Group testing for sars-cov-2 allows for up to 10-fold efficiency increase across realistic scenarios and testing strategies. *medRxiv*, 2020.