Single Cell Transcriptomics for Boolean Network Inference and Validation: Binarisation and Generation of Synthetic scRNA-seq Datasets

M1 Internship Report

MSc Bioinformatics and Computational Biology
M1 programme: Bioinformatics and Bio-statistics
Two-month Summer Internship

Gustavo MAGAÑA LOPEZ

Internship supervisors:

Loïc PAULEVÉ Laboratoire Bordelais de Recherche en Informatique	Head Scientific Supervisor
Christine FROIDEVEAUX Laboratoire de Recherche en Informatique	Scientific Supervisor
Stéphanie CHEVALIER Université Paris-Saclay, CNRS, Laboratoire interdisciplinaire des sciences du numérique	Scientific Supervisor
Andrei ZINOVYEV Computational Systems Biology of Cancer at Institut Curie	Scientific Supervisor

Contents

Preamble

Abstract

Binarisation of transcriptomic data is the first step needed in order to construct and evaluate Boolean Networks representing the dynamical system of gene regulation; with applications ranging from cellular transdifferentiation to clinically useful models of cancer patients. During my internship I wrote and published a `Python` package providing an interface to binarisation technique implemented in `R` via the `Python` package `rpy2`. I employed a parallel back-end in order to minimise the execution time of the binarisation pipeline's slowest part: The distribution classification. I wrote a naive algorithm to sample from the inferred distributions in order to assess the performance of the distribution classification algorithm. I proposed a model for a particular distribution which was not specified in the original article and tested it. The results indicate that a better model is needed, which opens the path for future projects in the area.

Dedicatory

I would like to dedicate this document to my parents Gustavo Magaña González and Yolanda López Camarena. Thank you for encouraging me and feeding my scientific curiosity since my childhood. None of my achievements would be possible without your love and support.

Aknowledgements

I want to thank the Instituto de Financiamiento e Información para la Educación (*Educafin*) for their financial support through the *Talentos de Exportación* scholarship programme. This programme has enabled me to pursuit my MSc in Bioinformatics and Computational Biology at Université Paris-Saclay.

I would like to thank Stéphanie Chevalier for proposing me this internship, letting me come closer to an area of bioinformatics that was previously unknown to me. I want also to thank all of my advisors: Your guidance and feedback throughout this project has allowed me to grow scientifically as never before.

1 Introduction

My main scientific advisor was Loïc Paulevé who is the principal investigator at the BNeDiction project which aims at providing a generic pipeline for the automatic construction of ensembles of dynamical models to deliver robust predictions from biological data, with applications to single-cell RNA-seq data. These dynamical models are implemented as Boolean Networks in which each node can be either "active" (1/True) or "inactive" (0/False). My work is therefore a connection between BNeDiction and the Laboratoire Interdisciplinaire des Sciences du Numérique LISN with whom I signed my internship.

Biological data seldom presents itself in such a discrete manner. Transcriptomic sequencing data provides a count matrix, in which some individuals have higher counts than others for certain genes. However there is no unique way to determine if a given gene should be considered as "active"/"highly expressed" or not.

Available binarisation techniques have been conceived for working with *bulk RNA-seq* data. **Single cell RNA-Sequencing** (*scRNA-seq*) data has different properties and biases, such as the characteristic zero-inflation of count matrices absent in bulk data.

Regarding binarisation techniques, recent publications try to solve different issues. Reference-based methods such as RefBool try allowing for the classification of a third state that can be interpreted as an intermediate expression of genes. In addition to this, another advantage offered by this method is that each measurement is associated to a $p-$ and $q-$values indicating the significance of each classification [Jung et al., 2017]. On the other hand, their implementation is written in `MATLAB`, a proprietary programming language for numerical analysis. Not only does one have to pay for a license in order to use the platform, `MathWorks` takes pride in selling ad-ons such as the `Statistical Toolbox` needed to run their pipeline. Other available techniques such as PROFILE [Beal et al., 2019b] propose a simple and efficient binarisation technique. However their code was dependent of the dataset, lacking generalisation.

Thus, my work focused on addressing an accessibility problem to binarisation pipelines.

Preamble: Optional Research Course

At the beginning of this semester I opted out of a `Python` course and asked to participate in a research project instead. My internship was a continuation of the work I did during the second semester of this academic year.

Objectives

- Integrate *scRNA-seq* binarisation methods to the BoNesis package for boolean network inference. These binarisation methods will be part of the first step of the pipeline: data preprocessing.
- Draft a method for synthetic scRNA-seq data generation from boolean results (the output of boolean networks) in order to provide a ground truth for the boolean network inference experiments.

2 Materials and Methods

Datasets

Since the beginning of my *TER*, I worked mainly with two datasets which are the following:

- **METABRIC**: The Molecular Taxonomy of Breast Cancer International Consortium database is a Canada-UK Project which contains targeted sequencing data of 1,980 primary breast cancer samples. METABRIC's `data_expression.txt` contains bulk RNA-seq data. It can be downloaded from this repository.

- **NESTOROWA**: A single-cell resolution map of mouse hematopoietic stem and progenitor cell differentiation [Nestorowa et al., 2016]. The data can be found on the Gene Expression Omnibus, using the series id GSE81682. I did not use the whole dataset, as the boolean networks that were interesting for Stéphanie Chevalier's work were those containing genes that deviated significantly from the mean expression profile. In order to select said subset, I used STREAM, an interactive pipeline capable of disentangling and visualizing complex branching trajectories from both single-cell transcriptomic and epigenomic data [Chen et al., 2019].

Development Tools

My main text editor is Visual Studio Code (*vscode*) [Microsoft, 2021] which I used to develop all Python source code files. I chose this editor because of the great variety of plug-ins available, integration with poetry and git. For the development of R code, I used RStudio [Allair, 2009]. In spite of vscode's virtues, RStudio has been specifically designed to be a data analysis environment which was a crucial part of my work. For these reasons I used both Integrated Development Environments (*IDEs*).

I used git and GitHub in order to keep an organised and annotated development history. A list of the repositories I created, worked with, etc. can be found in the appendix. This report was written and compiled using the **bookdown** [Xie, 2021] and **knitr** [Xie, 2015] `R` packages.

Virtual Environments (dependency management)

A fundamental problem in science is reproducibility. In computer science and bioinformatics publications the results' reproducibility heavily relies on having the exact same version of each and single one of software's dependencies.

Manual tracking, deployment, and verification of dependencies is tedious and time-consuming, not to mention error-prone. In order to address this **virtual environments** (*venvs*) and dependency trackers have been created. *Venvs* allow having many different installations of a programming language and its related dependencies. These are completely isolated from the operating system's version (if applicable). Furthermore, dependency trackers can create a text file specifying how to recreate the desired installation. This is a straightforward and reliable method to address this issue, which is why I chose it as my solution to the reproducibility issue.

There are other more recent solutions to this particular problem, using notably container technologies such as Docker and kubernetes. These approaches take the quest of reproducibility to a next level, creating "images" that contain a snapshot not only of the software packages but also of the operating system and its state. I did not personally use them as my virtual environments-based workflow was sufficient to develop the main software packages.

Python

I decided to use poetry [Eustace, 2018] as my dependency management system for `Python`. *Poetry* includes several advantages such as the automatic generation of a virtual environment for each new project, automatic resolution of package version conflicts and the possibility of directly publishing to PyPI. Furthermore, *poetry* uses the `pyproject.toml` specification which complies with one of the latest approved PEPs (see the appendix). The TOML language is designed to be simple and human-readable. Contrary to using a `setup.py` file which requires manual manipulation of the specification each time there is a change, the `pyproject.toml` file is automatically updated by *poetry*'s command line tool whenever a new release is created or a dependency is added.

R

To track my `R` dependencies I used renv [Ushey, 2021]. *Renv* is basically an `R` analogue of *venvs*. *Renv* works by creating a cache directory in which all installed packages will be stored. When initialised at the project's root directory, *renv* will automatically scan all the `*.R` and `*.Rmd` files, searching for their dependencies. It will then create a directory called `renv` to store the project specific library and other settings. It will append some lines to the project's `.Rprofile` so that each time that `R` is launched within directory, the project's private library will be loaded instead of the system's. *Renv* constantly analyses the project's dependencies (which packages are being imported in at least one script or notebook) and it takes the intersection of this set of needed packages and the set of those available in the project's library, writing a `renv.lock` file. The `renv.lock` file uses JSON to declare the `R` version and needed packages. Committing all changes made to this file into `git` will ensure the reproducibility of the `R` environment used to develop the software and perform all the experiments.

Polyglot package: integrating `R` with `Python`

During my *TER* I tried to write a pure `Python` implementation of the *PROFILE binarisation pipeline PROFILE-BINR*. However this turned out to be infeasible, at least within a reasonable time. First of all, there was not a `Python` implementation (on major statistics-related projects) of the *Dip Test of Unimodality* [Hartigan, 2016] which is fundamental to perform the distributions' classification. Of course I was not going to rely on some abandoned repository of someone claiming to have implemented it. I found a couple pure-Python versions of the `dip.test` on GitHub that I could have used, but the authors clearly stated that no maintenance was planned for their projects. Furthermore they where astonishingly slow, compared to the `R` version. This test was provided in `R` via the package `dip.test` [Maechler, 2021]. To further aggravate the issue, even though they are supposed to provide an interface to the same mathematical model, `Python::sklearn.mixture.GaussianMixture` and `R::mclust::Mclust` do not yield the same results when applied to real datasets (tests done with simulated gaussian mixtures yielded no significant differences). Of course I could have written and published a Python interface for the underlying `C` and `FORTRAN` code used by the R implementations but this was a more technical issue, rather than a bioinformatician's role in a project.

There are a couple of options for providing a `Python` interface to `R` but I decided to go with rpy2 as it is the biggest, most actively maintained. The structure is quite complex and objects are not automatically converted as they are in `R`'s interface to `Python` reticulate. In spite of this unpleasant surprise, `rpy2` offers a consistent interface to an embedded `R` session, which was the only real requirement for my work.

3 Results

Binarisation

The first result comes directly as a continuation of my *TER* project. I first developed a generic set of functions derived from the original notebooks. This was necessary because properties of the entry dataframe $X \in \mathcal{DF}(\mathbb{R})_{n \times m}$ (see mathematical notation) such as a column's name (`PATIENT_ID`) were hardcoded into certain functions of the original implementation. After this, I changed `compute_criteria()`'s processing scheme from *sequential* to *parallel*. The resulting implementation is **up to** $8x$ **times faster** on the reported datasets. See Figure 3.1 for a schematic representation of the benchmarks.

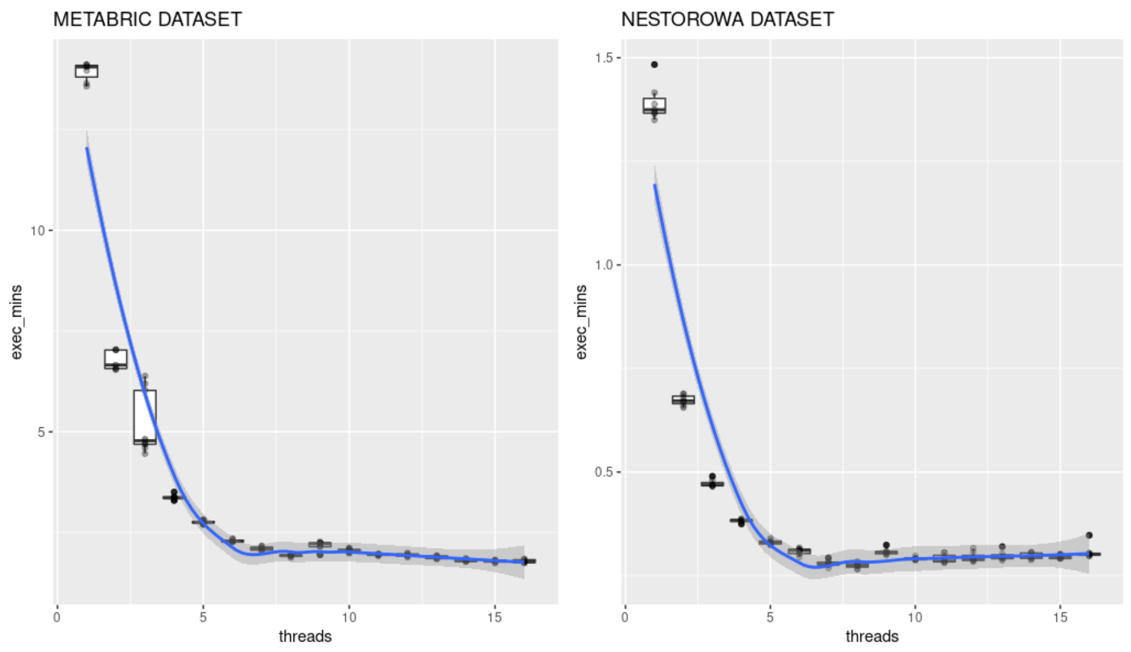

Figure 3.1: Execution time vs number of threads used to process the data.

Each boxplot represents 7 independent runs with the number of threads (optional parameter for the `R` function and its corresponding `Python` wrapper) set to the value appearing on the x axis. The first group ($n = 1$) represents the *sequential* scheme, all the others different number of threads $n \in [2, 16]$ for the *parallel* implementation.

Once the `R` functions had been tested, I created a `Python` class that would serve as the desired interface to access the binarisation technique from a Python environment. The class was designed to have a similar interface (set of methods) to the one employed by scikit-learn (*sklearn*) [Pedregosa et al., 2011]. This was a deliberate decision we took as a team, because *sklearn* is one of the most robust and popular machine learning `Python` packages. After thorough testing and discussion, the code was deployed to *PyPI* and the `colomoto` Anaconda (*conda*) channel [ana, 2020], from where it can be installed. See the repo's `README.md` for more details on dependencies and installation.

The PROFILE binarisation algorithm is presented as pseudocode in the *appendix*. On the same section there is a minimal example of how to use it.

Synthetic Data Simulation

The second goal of my internship was to develop a method to generate synthetic expression data from boolean entries such as the output of a boolean network. We decided to build the simulation method based on *PROFILE*'s gene distribution classification. Before designing an algorithm, I had to assess if *PROFILE*'s classification of the distributions was good enough to recreate the dataset's expression profile. Stated otherwise: When sampling from the inferred distributions and their parameters, we expect that the simulated dataset has a similar expression profile to that of the original. Furthermore, if we relaunch the binarisation pipeline with the new data, we would expect to get similar criteria and the same classification for all simulated observations across the genes.

This can be naively verified by comparing the distribution of mean expression values and their standard deviations across multiple datasets, as done by SPARSim [Baruzzo et al., 2020]. However we considered that this criterion was far from optimal: Even if the distributions of means and variances of the simulated genes seem to be the the same, the correlation structure of the resulting matrix $Y \in \mathcal{M}(R)_{n \times m}$ may differ from that of the input $X \in \mathcal{M}(R)_{n \times m}$. Therefore this criterion is necessary but not sufficient to evaluate a simulation method's performance.

We decided to go with SPsimSeq's [Assefa et al., 2020] evaluation strategy for their own work: **the mean** *vs* **variance curves**. This allows us to see not only that means and variances are on the same range (when comparing simulation to real) but also that variances for a given mean expression value do not significantly diverge from one another. In addition to this, I decided to visually represent the classifications made by *PROFILE* colouring the points accordingly. This can be seen on Figure 3.2.

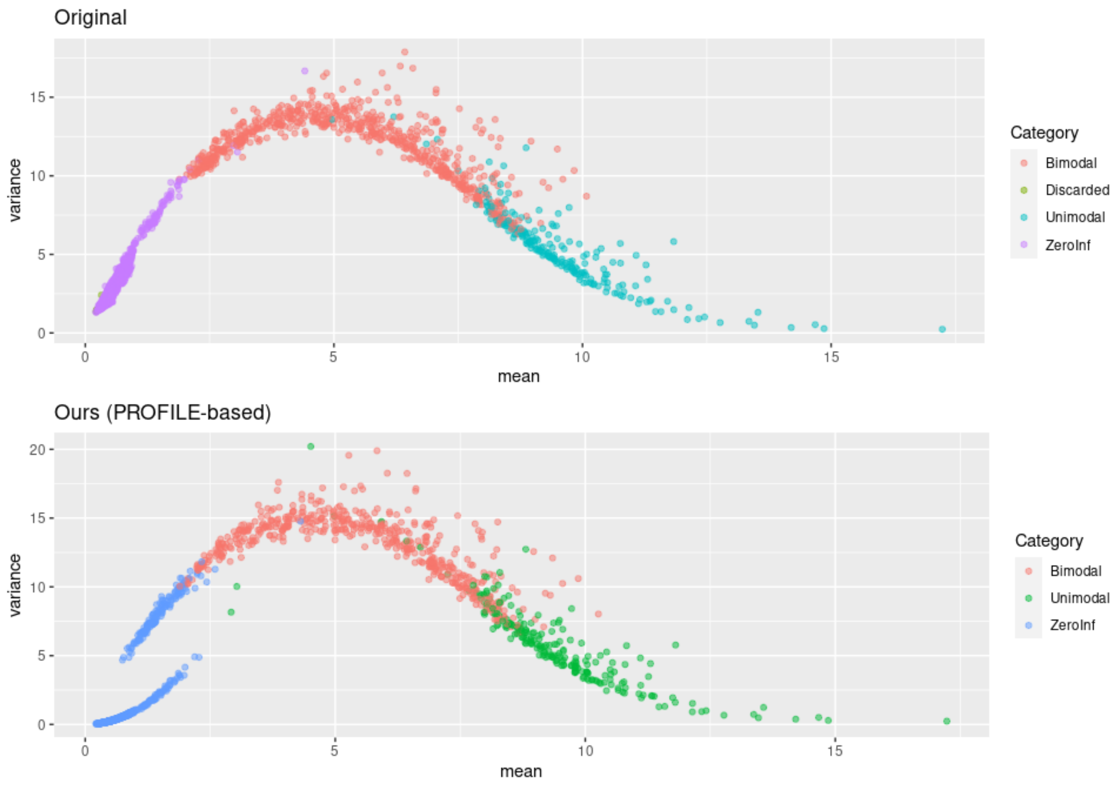

Figure 3.2: mean vs variance curve: (original vs naively sampled)

Overall, both curves seem to have the same form. We can also see that re-running the distribution classification yields similar results. There is however, a subset of the *ZeroInf* genes that seems to deviate significantly from the expected path. The code I used to sample from the estimated parameters can be found here.

4 Discussion

Bugs, Issues and unexpected behaviour

Memory duplication

The first problem that I encountered when doing a parallel implementation of `compute_criteria()` was that each one of the `R` workers spawned when using `doParallel` [Corporation and Weston, 2020b] (a library used to parallelise `foreach` loops [Revolution Analytics and Weston]) would consume as much memory as the main `R` process. Given that the METABRIC dataset's expression matrix consumes about 1*Go* of memory if 16 workers are spawned, 16 *Go* of RAM will be consumed. This was a considerable limitation because expression matrices can be considerably bigger and most laptops' memory does not exceed 16 *Go*

I started looking for alternatives and found an in-depth article discussing ways to do parallel programming and deal with big datasets using `R` [Kane et al., 2013]. I changed the parallel back-end from `doParallel` to `doSNOW` [Corporation and Weston, 2020a], which uses a different communication scheme: It is not forced to copy all the data from the main session to each worker. The combination of this parallel back-end and the `bigmemory` [Kane et al., 2019] package has the following property: only a descriptor is passed to the parallel worker instead of the object in memory thereby minimising the memory needed to achieve it.

One might notice that on Figure 3.1, there is an asymptotic behaviour. Parallelisation brings considerable time gains at first, and then it seems to stagnate arriving to a limit. The reason for this is: "…when more than four cores are employed the communication overhead required by snow overwhelms the performance gains." [Kane et al., 2013]. This back-end was chosen albeit this downside because it incurrs in less memory duplication and is truly cross-platform (other back-ends only work on the `UNIX-like` systems but on `Windows`).

Criteria and Expression indexes mismatch

When testing the `Python` interface, Loïc and I found out that some genes on the binarised matrix would contain entries equal to `-2147483648`. This seemed to be an overflow error, or maybe a type error behind the scenes. We tried debugging it by many means but none of them seemed to provide us with any useful information. When doing parallel processing, there is no guarantee that the data will be in the same order as it entered the function (the worker treating the first chunk of our data may finish last). We had a "misalignment" of the dataframe indexes that caused (sometimes) the wrong binarisation rule to be applied, creating the problem.

The solution to this was indexing the criteria dataframe with the expression dataframe's index, in order to ensure that the correct binarisation rule will be applied.

The burden of communication between `R` and `Python` via `rpy2`

When using `rpy2`, data must be passed back and forth between `Python` and the embedded `R` process. `rpy2` provides a pythonic interface to do so using context managers. However, for reasons that I ignore, passing data in the $Py \longrightarrow R$ direction is much slower than the inverse $R \longrightarrow Py$. When passing the expression dataframe into `R` this time is roughly equal to the time required to compute the criteria using 8 threads on my machine. This is one of profile_binr's greatest limitations.

Discrepancy of simulated *Zero-Inflated* genes distribution profiles

Figure 3.2 shows a significant deviation of the **mean** *vs* **variance** curve for genes classified as having a **Zero-Inflated** (*zeroinf*) distribution. In the original article [Beal et al., 2019a], authors provide a framework for binarising *zeroinf* genes, but do not propose an explicit model for sampling these distributions.

Given that *PROFILE-BIN* uses log-normalised expression matrices instead of raw counts, I knew that I had to find a continuous distribution to sample from instead of a discrete one. Looking at the histograms of *zeroinf* genes, I thought that using an exponential distribution was a reasonable approach. I estimated the sole parameter of the exponential distribution λ using its maximum likelihood estimator

$$\hat{\lambda} = \frac{n}{\sum\limits_{i=1}^{n} x_i} \tag{4.1}$$

When discussing the first results, Andrei and Loïc suggested that I explore other estimated parameters (i.e. criteria in the *PROFILE* context) to see if there were some correlations that could explain the observed discrepancies between real and simulated *zeroinf* genes.

First, I decided to analyse the binarisation threshold defined as $IQR(x)+q75(x)\ \ IQR(x) := q75(x)-q25(x)$ and noticed that it plummeted to zero after a *DropOutRate* of approximately 0.75. The excess of null entries on some genes biases the estimation of the quartiles, which makes the algorithm consider any non-null entry as an extreme value that should be binarized to 1. While this could be correct in some cases, *scRNA-seq* is known to have more zero entries than *bulk RNA-seq*. The fundamental problem here is that we have no way of distinguishing biological zeroes from technical ones. It is currently impossible to determine which percentage of the zero entries actually represent the absence of a particular transcript in the cell and non a sequencing error.

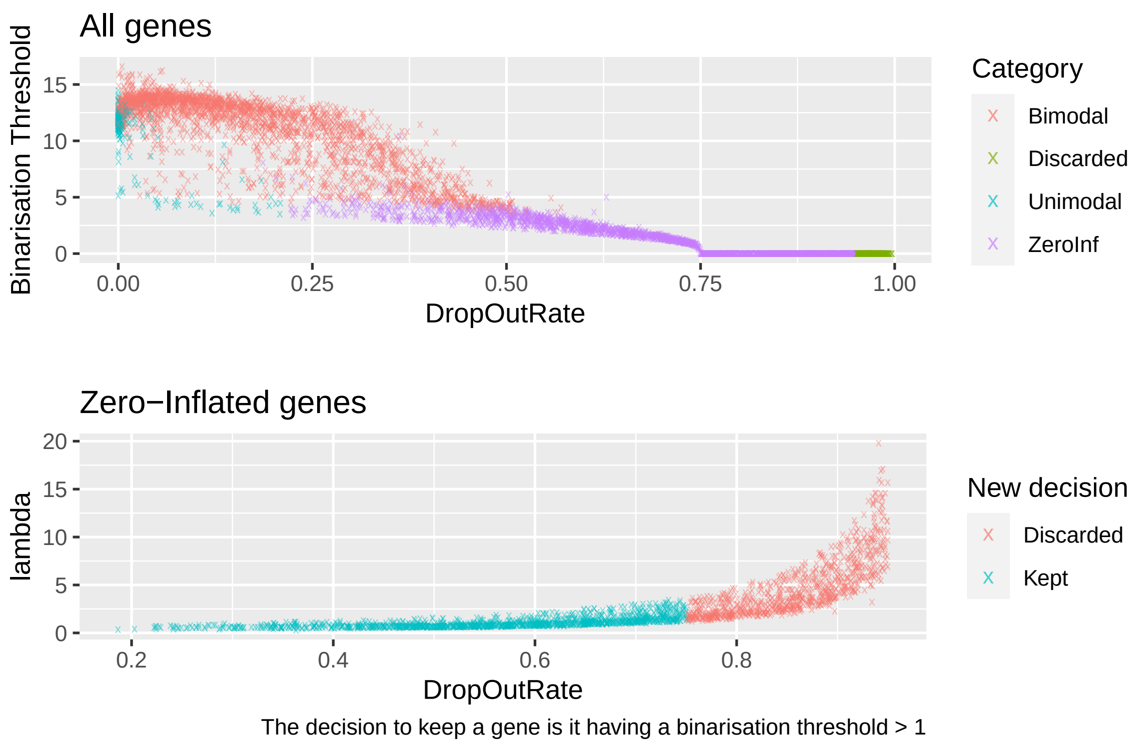

Figure 4.1: Two central parameters for ZeroInf genes, plotted against the dropout rate

After our discussion, we hypothesised that maybe the genes that had this excess *DropOutRate* (*dor*) were responsible for the observed deviation. Figure 4.2 shows that this is not the case. The excess of zeroes only accounts for a small fraction of the observed discrepancy. This leads me to conclude that the mistake is fundamentally the choice of the exponential distribution, rather than an accidental property of the data. We can (and probably should) discard *zeroinf* genes with a *DropOutRate* such that its binarisation threshold is equal to zero, but a new approach is needed in order to build a robust simulation framework based on *PROFILE*.

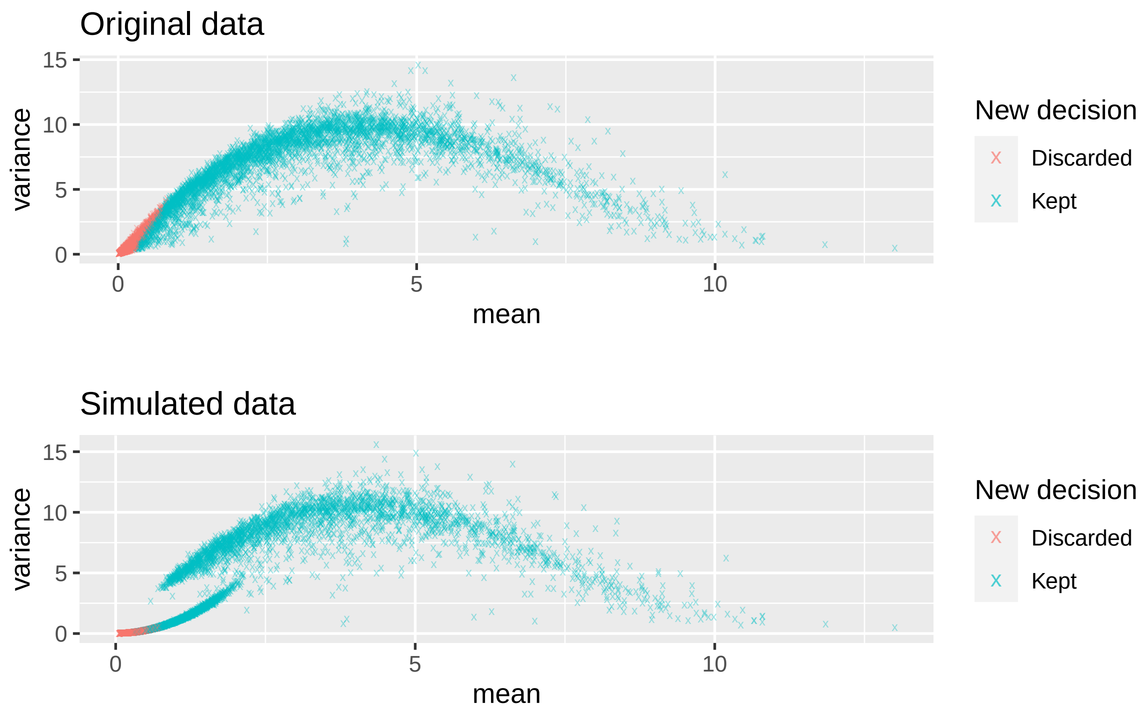

Figure 4.2: Visualisation of the stronger criteria for discarded genes

At our latest meeting, Andrei proposed a new approach that did not require adding new distributions to the classification: Once that a gene has been classified as *zeroinf*, we can remove all null entries and recompute the parameters. This way we could capture the sometimes unimodal, sometimes bimodal character that we observed on some *zeroinf* genes. This is something that I will have to implement in the future as the time of the internship has now come to an end.

5 Conclusion and perspectives

The `Python` package I wrote has now been tested and published via the two greatest distribution channels for `Python` software (*conda* and *PyPI*). This means that not only the members of the research team with whom I worked have access to it, but greater research community working in this domain. The interface is direct, simple and easy to use as shown in the appendix. The binarisation algorithm has a good performance in terms of speed (see 3.1) and memory consumption making it a useful tool for the team I worked in.

Albeit the discussed discrepancies regarding the samping of *zeroinf* genes, there is evidence that *PROFILE* model is good enough to be the basis for a new algorithm to produce synthetic *scRNA-seq* data.

6 Appendix

Notation and Disambiguation

Term Disambiguation

- *NB* is short for latin *Nota bene*. This translates to "note well". I use this when I want to draw the reader's attention to a particular issue.
- *PROFILE* is the name that the authors of **Personalization of Logical Models With Multi-Omics Data Allows Clinical Stratification of Patients** decided to their framework and methodologies.
- *PROFILE-BIN* is the binarisation technique developed in [Beal et al., 2019b].
- *profile_binr* is the Python package that I developed to make use of *PROFILE-BIN*, containing Python wrappers for the R code containing a generalised version of functions defined in original article.
- *scRNA-seq*
- *TER* means *Travaux d'Études et de Recherche*. It represents an optional research-focused project that replaces credits that would have been otherwise earned through a standard lecture class. It can be consulted in the programme's site.
- *PEP* stands for Python Enhancement Proposal. It is a design document which is really useful to the Python community as it sets a style guide and defines numerous standards related to the language. More info on the official site.

Mathematical Notation

- nan := a generic representation of an undefined value
  - This definition applies to different contexts such as the result of an undefined operation i.e. $x/0 \quad \forall x \in \mathbb{R}$, a value that could not be resolved to active or inactive whilst binarising an expression matrix, etcetera.
- $\mathbb{N}$:= the set of natural numbers
- $\mathbb{R}$:= the set of real numbers
- $\mathbb{R}^+ := \{x \in \mathbb{R} \mid x \geq 0\}$
- $\mathbb{B} := \{0, 1\}$
- $\mathbb{B}^\star := \mathbb{B} \cup \{\text{nan}\}$
- $\mathbb{T}$:= Any form of text, the string type
- $\mathcal{M}(\lambda)_{n \times m}$:= a matrix with n rows and m columns, containing entries of type λ.
  - When indexing a matrix A, the notation $A(i, j)$ clearly indicates the entry a_{ij}. To access a whole row (across all columns), we can write $A(i, :)$, conversely $A(:, j)$ denotes j-th column (across all rows).
- $\mathcal{DF}(\lambda)_{n \times m}$:= a dataframe with n rows and m columns, containing entries of type λ.
  - The fundamental difference between a matrix $\mathcal{M}$ and a dataframe $\mathcal{DF}$ is that the dataframe's rows and columns are named, meaning that a dataframe W can be indexed as follows: $W(\text{cell HSPC\_025}, \text{gene Coro2b})$. However, dataframes still allow number-based indexing as if they were matrices.

Statistical Models

Gaussian Mixtures

A mixture model can be conceived as a composite model defined by K components, each of which explains a subset of the observations. The superposition of these models $\phi_i(x|\theta_i) \;\forall\; i \in [1, K]$ weighted by the proportion of observations captured by each one of them $\pi_i \;\forall\; i \in [1, K]$ accounts for the nuances in the overall distribution. This can be written down as follows:

$$f(x|\theta) = \sum_{j=1}^{K} \pi_j \phi(x|\theta_j) \quad \text{with} \quad \sum_{j=1}^{K} \pi_j = 1 \;\; ; \quad \theta_i \in \mathbb{R}^p, \;\; \pi_i \in \mathbb{R}^+, \;\; x \in \mathbb{R} \tag{6.1}$$

A **Gaussian Mixture Model** (*GMM*) is the special case in which all components follow gaussian distributions $\phi_i(x|\theta_i) \mid \theta_i = (\mu_i, \sigma_i, \pi_i)$ with parameters representing the mean μ_i, standard deviation σ_i and probability π_i of each component. In their original work the authors of PROFILE [Beal et al., 2019a], [Beal et al., 2019b] decided that a two-component $GMM(K = 2)$ was ideal to capture the expression distribution of certain genes (along with the **Zero-Inflated** (*ZeroInf*) and **Unimodal**). This is logical given that in some patients (as was the focus of their publication) genes might be highly expressed whereas in others not. Furthermore, a bimodal model is adequate for the binarisation (observations from one distribution are set to zero, those belonging to the second component to one) they were aiming to achieve.

$$x_i \sim \phi(x|\phi_1) \longrightarrow 0 \quad ; \quad x_j \sim \phi(x|\phi_2) \longrightarrow 1 \quad ; \quad i \neq j \tag{6.2}$$

Albeit being developed for the processing of bulk data, this model is still adequate for the binarisation of *scRNA-seq* data. Given a single-cell dataset, one would expect to see that some genes are highly expressed in some cells and inactive in others. This character is optimally modelled by the proposed $GMM(K = 1, \sigma_1 = \sigma_2)$

Bimodality Index (BI)

The **Bimodality Index** (*BI*) evaluates the ability to fit two distinct Gaussian components with equal variance. Once the best 2-Gaussian fit is determined, along with the respective means μ_1 and μ_2 and common variance σ, the standardized distance δ between the two populations is given by

$$\delta = \frac{|\mu_1 - \mu_2|}{\sigma} \tag{6.3}$$

and the *BI* is defined as

$$BI = \delta\sqrt{\pi_1(1 - \pi_1)} \tag{6.4}$$

where π_1 is the proportion of observation in the first component. In *PROFILE*, the *BI* is computed using the `R` package `mclust` [Fraley et al., 2020].

Algorithms and Functions

Profile binarisation algorithm

```
PROFILE_BINARISATION()
lexicon
    $n \in \mathbb{N}$ (number of cells)
    $m \in \mathbb{N}$ (number of genes)
    $c \in \mathbb{N}$ (number of criteria)
    $E \in \mathcal{DF}(\mathbb{R})_{n \times m}$ (log-normalised expression)
    $C \in \mathcal{DF}(\mathbb{R} \cup \mathbb{T})_{m \times c}$ (criteria to determine binarisation rule)
    $B \in \mathcal{DF}(\mathbb{B})_{n \times m}$ (binarised expression)

begin
    for each $gene \in column\_names(E)$ :
        C(gene, :) $\leftarrow$ $compute\_criteria(E(:, gene))$
    for each $gene \in row\_names(C)$ :
        category $\leftarrow$ C(gene, Category)
        switch Category :
            case Discarded :
                B(:, gene) $\leftarrow$ nan
            case ZeroInf :
                B(:, gene) $\leftarrow$ $BIM\_class(E(:, gene))$
            default :
                (default applies to both Zero-Inflated and Unimodal genes)
                B(:, gene) $\leftarrow$ $OS\_class(E(:, gene))$
end
```

For further details regarding the functions `BIM_class()` and `OS_class()` please refer to the original article [Beal et al., 2019a] or to the repository.

Minimal Code Examples

ProfileBin example

```
from profile_binr import ProfileBin
import pandas as pd

# your data is assumed to contain observations as rows and genes as columns
# this is not always true when dealing with transcriptomics data !
data = pd.read_csv("path/to/your/data.csv")
data.head()

# create the binarisation instance using the dataframe with the index containing
# the cell identifier and the columns being the gene names
probin = ProfileBin(data)

# compute the criteria used to binarise the data, using 8 threads
probin.fit(8)

# Look at the computed criteria
probin.criteria

# get binarised data (alternatively .binarise()):
my_bin = probin.binarize()
```

List of Code Repositories

All the entries of the following list are links that will allow you to access the repositories that I mentioned, analysed, used, etc. during the *TER* and the internship.

- Reference:
  - Binatisation
    - PROFILE
    - RefBool
  - Synthesis
    - STREAM
    - SPARSim
    - SPsimSeq
- The Project's site:
  - BNediction
- My repositories:
  - profile_binr
  - additional resources

List of Python's Major Statistical Projects

- scikit-learn
- SciPy Stats
- statsmodels

Hardware Specifications and Programming Language Versions

Laptop and Operating System

I use a System76 Oryx Pro, running Pop!_OS which is an Ubuntu-based GNU/Linux distribution.

```
$ neofetch
             /////////////                gml@reykjavik
         /////////////////////            -------------
      ///////*767////////////////         OS: Pop!_OS 20.04 LTS x86_64
    //////7676767676*//////////////       Host: Oryx Pro oryp6
   /////76767//7676767//////////////      Kernel: 5.11.0-7614-generic
  /////767676///*76767///////////////     Uptime: 5 hours, 32 mins
 ///////767676///76767.///7676*///////    Packages: 3910 (dpkg), 75 (flatpak), 4 (snap)
/////////767676//76767///767676////////   Shell: fish 3.1.2
//////////76767676767////76767/////////   Resolution: 1920x1080, 1920x1080
///////////76767676/////7676///////////   DE: GNOME
////////////,7676,//////767////////////   WM: Mutter
/////////////*7676//////76/////////////   WM Theme: Pop
///////////////7676////////////////////   Theme: Pop-dark [GTK2/3]
 ///////////////7676///767////////////    Icons: Pop [GTK2/3]
  //////////////////////'////////////     CPU: Intel i7-10875H (16) @ 5.100GHz
   //////.7676767676767676767,//////      GPU: Intel UHD Graphics
    /////767676767676767676767/////       GPU: NVIDIA GeForce RTX 2060 Mobile
      ///////////////////////////         Memory: 10777MiB / 31979MiB
         /////////////////////
             /////////////
```

Python

```
$ python
Python 3.8.6 (default, May 27 2021, 13:28:02)
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

R

```
$ R
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
>
```

Bibliography

Anaconda software distribution, 2020. URL https://docs.anaconda.com/.

Joseph J. Allair. RStudio | Open source & professional software for data science teams - RStudio, 2009. URL https://www.rstudio.com/.

Alemu Takele Assefa, Jo Vandesompele, and Olivier Thas. SPsimSeq: Semi-parametric simulation of bulk and single-cell RNA-sequencing data. *Bioinformatics*, 36(10):3276–3278, 2020. ISSN 14602059. doi: 10.1093/bioinformatics/btaa105.

Giacomo Baruzzo, Ilaria Patuzzi, and Barbara Di Camillo. SPARSim single cell: A count data simulator for scRNA-seq data. *Bioinformatics*, 36(5):1468–1475, 2020. ISSN 14602059. doi: 10.1093/bioinformatics/btz752.

Jonas Beal, Arnau Montagud, Pauline Traynard, Emmanuel Barillot, and Laurence Calzone. Personalization of logical models with multi-omics data allows clinical stratification of patients. *Frontiers in Physiology*, 10(JAN), 2019a. ISSN 1664042X. doi: 10.3389/fphys.2018.01965.

Jonas Beal, Arnau Montagud, Pauline Traynard, Emmanuel Barillot, and Laurence Calzone. Personalization of logical models with multi-omics data allows clinical stratification of patients. *Frontiers in Physiology*, 10(JAN):1–23, 2019b. ISSN 1664042X. doi: 10.3389/fphys.2018.01965.

Huidong Chen, Luca Albergante, Jonathan Y. Hsu, Caleb A. Lareau, Giosue Lo Bosco, Jihong Guan, Shuigeng Zhou, Alexander N. Gorban, Daniel E. Bauer, Martin J. Aryee, David M. Langenau, Andrei Zinovyev, Jason D. Buenrostro, Guo Cheng Yuan, and Luca Pinello. Single-cell trajectories reconstruction, exploration and mapping of omics data with STREAM. *Nature Communications*, 10(1), 2019. ISSN 20411723. doi: 10.1038/s41467-019-09670-4. URL http://dx.doi.org/10.1038/s41467-019-09670-4.

Microsoft Corporation and Stephen Weston. *doSNOW: Foreach Parallel Adaptor for the snow Package*, 2020a. URL https://CRAN.R-project.org/package=doSNOW. R package version 1.0.19.

Microsoft Corporation and Steve Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2020b. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.16.

Sebastien Eustace. Poetry - Python dependency management and packaging made easy, 2018. URL https://python-poetry.org/.

Chris Fraley, Adrian E. Raftery, and Luca Scrucca. *mclust: Gaussian Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation*, 2020. URL https://mclust-org.github.io/mclust/. R package version 5.4.7.

P M Hartigan. The Dip Test of Unimodality Author (s): J . A . Hartigan and P . M . Hartigan Published by : Institute of Mathematical Statistics Stable URL : http://www.jstor.org/stable/2241144 Accessed : 02-03-2016 16 : 51 UTC Your use of the JSTOR archive indicates . 13(1):70–84, 2016.

Sascha Jung, Andras Hartmann, and Antonio Del Sol. RefBool: A reference-based algorithm for discretizing gene expression data. *Bioinformatics*, 33(13):1953–1962, 2017. ISSN 14602059. doi: 10.1093/bioinformatics/btx111.

Michael J. Kane, John W. Emerson, and Stephen Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. ISSN 15487660. doi: 10.18637/jss.v055.i14.

Michael J. Kane, John W. Emerson, Peter Haverty, and Charles Determan Jr. *bigmemory: Manage Massive Matrices with Shared Memory and Memory-Mapped Files*, 2019. URL https://github.com/kaneplusplus/bigmemory. R package version 4.5.36.

Martin Maechler. *diptest: Hartigan's Dip Test Statistic for Unimodality - Corrected*, 2021. URL https://github.com/mmaechler/diptest. R package version 0.76-0.

Microsoft. Visual Studio Code - Code Editing. Redefined, 2021. URL https://code.visualstudio.com/.

Sonia Nestorowa, Fiona K. Hamey, Blanca Pijuan Sala, Evangelia Diamanti, Mairi Shepherd, Elisa Laurenti, Nicola K. Wilson, David G. Kent, and Berthold Göttgens. A single-cell resolution map of mouse hematopoietic stem and progenitor cell differentiation. *Blood*, 128(8):e20–e31, 2016. ISSN 15280020. doi: 10.1182/blood-2016-05-716480.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Revolution Analytics and Steve Weston. *foreach: Provides Foreach Looping Construct*.

Kevin Ushey. *renv: Project Environments*, 2021. URL https://rstudio.github.io/renv/. R package version 0.12.5.

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.name/knitr/. ISBN 978-1498716963.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2021. https://github.com/rstudio/bookdown, https://pkgs.rstudio.com/bookdown/.