University of Glasgow

School of Computing Science

Building a data pipeline to extract signatures of evolutionary dynamics from tumours

Razak Nart, Mingfeng Liu, Lukas Rubikas, Yuhong Lin, Jesus Vera

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in partial fulfilment of the requirements of the Degree of Master of Science at the University of Glasgow

December 26, 2018

Abstract

Analysing tumour samples for estimation and clustering of cancer cell fractions is of vital importance when studying intratumour heterogeneity. In regard to this project, we introduce a pipeline containing three tools to extract cancer cell fractions (CCF) to be able to perform clustering from tumour samples. Furthermore, two other tools were then used to complete topic extraction using the CCF values. Separating these tools in two phases, the results from the first phase were compared on accuracy and speed. Whiles the results from the second phase will be compared for similarity. Users will be able to run the pipeline through a Docker environment.

Contents

Chapter 1 Introduction

With the advance of technology and the continuous increment of cancer cases, it has become of vital importance to keep doing research on this matter to create computer tools that can help scientist to find cures or to improve the accuracy of analysis for this disease and to get an early prognosis on patients.

Machine Learning plays an important role in this kind of researches because with the use of different algorithms computers can be taught to do repetitive processes where models can be improved with the data obtained on each model iteration. Thus, making a model better for predicting and analysing results and giving great importance to the creation of new tools that implement best algorithms for machine learning.

The main objective of this project is to build a pipeline that will use 500 tumour samples from different patients and studying their intratumour heterogeneity to estimate and extract the cancer cell fraction (CCF) values using clustering on each of the samples during the first phase. Then, these results will be analysed for topic extraction during the second phase.

The project was divided into two phases. Each phase will require different data to be able to complete its task. During the first phase, there are three important tools that are used to analyse the tumour samples: PyClone, Ccube and PhyloWGS. These tools work in different ways, yet they are employed to do the same job. For this project, these tools will be analysed, implemented and integrated into a pipeline that will be possible to run them together. Furthermore, it will show differences between the tools which will help in the future for deciding which tool works better, although in this project it will be limited to measure only speed and accuracy.

After obtaining the results from the first phase, another two tools will be run using those results: Non-negative Matrix Factorization and Latent Dirichlet Allocation. These tools use two different models for topic extraction, but these models need the results from the first phase to run. Once the models are provided with the necessary data, the CCF values will be analysed to obtain a certain number of topics as a result. The values contained in the topics will represent the probability (LDA) or weights (NMF) of a document to be matched with a certain topic. These two tools will also be integrated into the pipeline that will run the five different tools one after the other.

Finally, once the pipeline is ready with the five different tools, a Docker environment will be built and configured to run it. This environment will contain all the necessary tools already installed and ready to run the pipeline code.

Chapter 2 Background Survey

In this chapter, the five different tools used for analysing each cancer cells will be explained. They are divided into two phases. The first phase includes three tools that will analyse 500 different cancer samples to extract important values from them and with that help the next phase to continue with the analysis. The second phase will include two different tools for topics extraction from the results during the first phase. These topics will be the relations between the different cancer cell fractions (CCF) across all tumour samples.

The first phase will use the tools: PyClone, Ccube and PhyloWGS; and the second phase will consist of the tools for using the models of Non-negative Matrix Factorization and Latent Dirichlet Allocation.

2.1 PyClone

Although the development of measuring allele prevalence by using deep sequencing has grown successfully, there wasn't a statistical method for clustering deep digital sequencing of mutations into biological groupings.

Aiming at classifying sets of deeply sequenced somatic mutations into a presumptive clonal structure, a hierarchical Bayes statistical model for the assumption of clonal population clusters was developed by Andrew Roth and his workmates [1]. Apart from the classification, the model calculates clusters' cellular prevalence (which refers to cancer cell fraction (CCF) in this dissertation) as well as makes an explanation of allelic imbalances. The paper highlights that, during clonal phylogeny, the clusters of mutations which locate at the same point are sharing cellular prevalence. Taking advantage of this property, the model considers it as markers of clonal populations. However, the allelic prevalence of a mutation contributes to several features, which means it is not relevant directly to cellular prevalence. It will be aggravated especially if a single sample estimation is taken.

Based on the property and Bayes theory, a Bayes statistical software tool, PyClone, was developed to overcome the above difficulty. Pyclone consists of four innovative modelling processes, including beta-binomial emission densities, estimation of flexible prior probability for possible mutational genotypes, Bayesian nonparametric clustering and joint analysis of multiple samples.

The software tool was tested in idealized datasets which extracted from four 1000 Genomes project samples and mutational profiles of multiple samples provided from a high-grade serous ovarian cancer (HGSOC) [2][3]. By comparing two results, Pyclone obtains similar conclusions between two datasets, which proves its robustness. The experiments illustrate that PyClone provides a reliable statistic inference method to support researches in the progression of cancer. Additionally, according to cluster accuracy, the paper reveals that PyClone with beta-binomial emission densities with parental and total copy-number priors surpass all other methods in the process of emission densities [1].

2.2 Ccube

For producing the cancer cell fraction (CCF) from somatic point mutation calls and the clusters of every mutation, a good way is to use Ccube to analyse the data and get the output files for the second phase of this project.

Ccube is a tool written with R code for clustering the cancer cell fraction. Moreover, it is a probabilistic framework that includes a variational inference method for model fitting, which allows the user to process samples with a large number of variants while quantifying the uncertainty in a Bayesian fashion [2]. From the whole genome sequencing data, the cancer cell fraction and sub-clonal composition of somatic cell point mutations could be inferred. It requires sequencing read analysis of single nucleotide variants (SNV), correcting their copy number changes and purity, and generating CCF estimates for all mutations in the sample.

Furthermore, the Ccube choose the suitable number of clusters by variational inference which can get the best evidence lower bound and assign the mutations to the clusters according to the probability. Also, inside the Ccube pipeline there is an optional parameter named "runQC" that when is set to TRUE, the Ccube can remove small clusters and re-assign variants to make the result of clustering more accurate.

Thus, using Ccube it is possible to find the CCF and VAF relationship. Hence, the map between them is as follows:

$$f = w\emptyset + \epsilon$$

$$\omega = \frac{t(m(1 - \epsilon) - n_{tot_t}\epsilon)}{(1 - t)n_{tot_n} + tn_{tot_t}}$$

Formula 1: Linear mapping between the probability of observing a variant read at a mutated locus, f, and the CCF of the mutation $\emptyset$. Where $\boldsymbol{\epsilon}$ is a uniform sequencing error, and m is the number of mutated chromosomal copy [4].

2.3 PhyloWGS

The methods of reconstructing a tumour phylogeny mainly fall into two camps: one is based on the tumour simple somatic mutation (SSM) data; others focus on the observed copy number variations (CNVs) of these SSMs. Taken separately, these approaches come with their limitations and varying levels of difficulty, either by making several consequential assumptions about the underlying nature of the evolution of a tumour (SSM-based approaches) or are severely limited to correctly inferring only relatively simple tumour phylogenies (CNV-based approaches). These approaches are generally limited to a certain degree of accuracy and it is even possible to demonstrate situations where using both approaches separately on the same tumour sample yields incorrect results [20].

PhyloWGS [20] is a first method that falls in between of these two camps - it incorporates both the information about the tumour SSMs and their corresponding CNVs while making fewer assumptions about the tumour phylogeny along the way. With important considerations of the order of occurrence between SSMs and CNVs, PhyloWGS accomplishes such a task via a complicated process that involves creating a pseudo-simple somatic mutation for each of the copy number variations and assigning all the affected SSMs within the region to these newly created pseudo-SSMs.

Based on Metropolis-Hastings algorithm that samples from a posterior distribution of a generative probabilistic model with non-parametric Bayesian priors, PhyloWGS also does not report only a single phylogenetic tree, but a tree for every converged Markov chain Monte Carlo simulation.

Written in Python and C++, PhyloWGS reports an increased performance over PyClone due to the usage of powerful scientific C++ library GSL [21] and incorporation of both CNV and SSM data. From a software engineering perspective, it might be best described as an end-to-end piece of software, bundled with input parsers for several popular data formats for this type of task (such as Battenberg and TITAN) with a framework to implement your own (which we make use of) and a web-based locally hosted interface for interpreting results.

While the full scope of functionality that PhyloWGS offers is broader that what we need in constructing our data pipeline, we were able to pick parts of it that we made use of, despite having to make a few compromises along the way, in a method that will be described in the chapters that follow.

2.4 Non-negative Matrix Factorization (NMF)

Non-negative Matrix Factorization (NMF) introduced by Lee and Seung [5][6], is a model that is used to achieve a dimension reduction on a large complex data matrix to obtain valuable features. It works much like Principal Component Analysis (PCA) but in NMF, each feature in the data matrix, must be greater or equal to zero (non-negative) [7].

Since the introduction, other researchers had successfully implemented NMF in different areas, such as document clustering [8]; information retrieval [9]; facial expression recognition [10]; gene expression analysis [11][12]. The aim is to factorise a non-negative data matrix A with dimension $m \ x \ n$ to produce approximation matrices WH with dimensions $m \, x \, k$ and $k \, x \, n$ respectively, that is,

$$A \approx WH,$$

where the k serves as the number of component factors in the model and normally selected so that $mn > k(m+n)$ [5][11].

Regarding this project, Cancer Cell Fraction (CCF) expression values from a set of tumour samples are presented in a matrix A with dimension $m \, x \, n$, where the rows m relates to the band of the CCF, the columns n relates to the tumour samples and entries are the total values of the CCF that falls within the bands. After applying NMF on the CCF expression value matrix A to WH, where W has dimension $m \, x \, k$, such that k columns determine the number of topics in the expression and where H has dimension $k \, x \, n$, such that n columns reveal the weight of the expression of the tumour sample. Furthermore, the W can be described as a feature matrix and H as the coefficient matrix.

To factorise the data matrix A to get the product of WH, an objective function must be defined to measure the approximation and the reconstruction error. This function can be designed by adjusting the distance between A and the multiplication result WH [11]. The commonly used method for measuring the distance is the squared *Frobenius norm* a branch of the *Euclidean norm* [8].

$$\frac{1}{2}\|A - WH\|_F^2 \sum_{i=0}^{n} \sum_{j=0}^{m} \left(A_{ij} - (WH)_{ij}\right)^2$$

Formula 2: The Frobenius norm for measuring the distance separating A and its product WH, [5][18].

A different method for measuring the distance is *Kullback-Liebler* [5][11]. The commonly used process for adjusting the W and H in *Formula 2* to reduce the objection function is to use the multiplicative update rules to iterate between the W and H until convergence [5]. The rules are as follow,

$$H_{aj} \leftarrow H_{aj} \frac{(W^T\boldsymbol{A})_{aj}}{(W^T\boldsymbol{WH})_{aj}}, \quad W_{ia} \leftarrow W_{ia} \frac{(\boldsymbol{A}H^T)_{ia}}{(\boldsymbol{WH}H^T)_{ia}}$$

Formula 3: The Multiplicative Update Rules for W and H [5].

2.5 Latent Dirichlet Allocation (LDA)

LDA is a generative probabilistic model that can abstract topics from collections of discrete data. It can also be seen as a model with three levels, and every item of a collection is matched with a topic (or topics) and their probabilities, therefore showing a clear representation of text corpora [13].

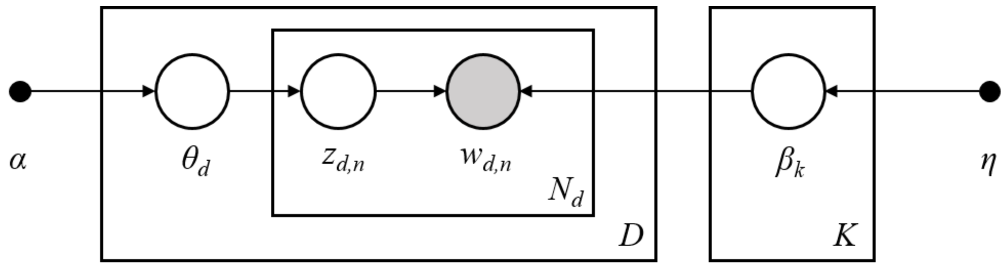


Figure 1: Graphical model representation of LDA, from scikit-learn "Decomposing signals in components (matrix factorization problems)" [14]. The corpus is represented by D, the document is a sequence of N words, K is the topics in the corpus and the boxes are repeated sampling.

Explained differently, LDA assumes that each word is generated by an assortment of topics, then it models the relationships between documents and topics and the relationships among such topics and words.

LDA also assumes, in this project, for each document w in a corpus D *[13][15]:*

1. Choose a multinomial ξ_k ($k \in \{1, \cdots, K\}$) for each topic from a Dirichlet distribution (β);
2. Choose a multinomial θ_s ($k \in \{1, \cdots, S\}$) for each CCF band from a Dirichlet distribution (a);
    a. Choose a topic z from a multinomial (θ_s).
    b. Choose a word w_n ($n \in \{1, \cdots, N\}$, where N is the number of words in the current document) from a multinomial (ξ_z).

Therefore, the output of LDA model are two matrices, a document-topic and a topic-words matrix, which can be represented with dimensions (N, K) and (K, M) respectively, where N is the number of documents, K is for the number of topics and M is the vocabulary size. These matrices represent the probability distributions of documents-topics and words-topics.

Chapter 3 Requirements

Since this project was a mixture of different tools, the requirements varying depending on the tool, hence the requirements would be better explained if separated. Furthermore, the requirements for the pipeline with the five tools will be explained and it will include how it should be implemented and run.

3.1 PyClone

For the PyClone tool, the only operative system employed was Unix (like Linux or MacOs) and the version of Python must be 2.7.

Additionally, it should be mentioned that PyClone is the main library used in this project and it's also an open-source code [4]. According to its documents, the main dependencies are PyDp, PyYAML. Those two libraries are used for analysing. Besides, Matplotlib, Numpy, Pandas, Scipy and Seaborn (higher or equal than 0.6.0) are required for plotting result and clustering. PyClone and all the required libraries can be installed via `pip install` (note: refer to README.txt for installation instructions).

Additionally, R scripts are used to pre-process input files and generate output files. The required libraries for R scripts are "dplyr" and "mcclust".

The final requirement is the data. According to the documents of PyClone, the input is a set of deeply sequenced mutations from one or more samples extracted from a single patient as well as allele-specific copy number at each mutation trajectory in each sample [1]. In our case, there are 500 samples from different patients.

In summary:

- Programming languages: Python (2.7) and R (3.4.0 or above)
- Main library used: PyClone (latest version)
- Other Python libraries:
    - PyDp (0.2.3 or above)
    - PyYAML (3.10 or above)
    - Matplotlib (1.2.0 or above)
    - Numpy (1.6.2 or above)
    - Pandas (0.11 or above)
    - Scipy (0.11 or above)
    - Seaborn
- Other R libraries: dplyr, mcclust
- 500 samples data (previously provided by the supervisor)

3.2 Ccube

Similarly, in the case of the tool Ccube, the required operating system is Linux. It works using the programming language R. Additionally, the core package is

“Ccube” with version 0.0.0.9000; which includes all functions of Ccube to analyze the mutations and get the CCF values. Ccube also needs the package “dplyr” to add new variables that work as functions of existing variables and the package “doParallel” to run it using multi-cores.

Another important requirement is the data used, which should be converted to a suitable format. The code of this process is based on the Python. It needs the package “OS” to read the path from the command shell, and the package “NumPy” to stack the arrays.

In summary:

- Programming languages: Python (2.7), R (3.4.0 or above)
- Main library used: Ccube (latest)
- Other Python libraries:
    - Numpy (1.6.2 or above)
    - Os (latest for Python 2.7)
- Other R libraries: DoParallel, Dplyr

3.3 PhyloWGS

In order to use PhyloWGS natively, we need the following tools and libraries. While the requirements in the official documentation are rather parsimonious, we will include the major version numbers we used in the pipeline below, in order to achieve maximum reproducibility:

- Programming language: Python (2.7)
- Python libraries:
    - NumPy (1.11)
    - SciPy (1.1)
    - ETE2 (2.3)
    - PyVCF (0.6)
    - pandas (0.23)
- Other libraries:
    - GSL – GNU Scientific Library for C and C++ (2.5)
    - getopt, the C library for parsing command line arguments in shell scripts (from the util-linux package, version 2.31)

While it is not explicitly mentioned, and a compiled version of GSL for Windows exist as part of Cygwin for Windows, for the ease of use we will assume the user uses a Unix-based operating system, such as Ubuntu Linux or MacOS.

In addition to downloading these tools, a PhyloWGS GitHub repository [22] must be cloned, navigated to, and the following line must be executed in the terminal:

```
g++ -o mh.o -O3 mh.cpp  util.cpp `gsl-config --cflags --libs`
```

Figure 2: Compilation instructions for the C++ file

This ensures that the C++ file that uses GSL within PhyloWGS is compiled and the tool is ready to be used.

Note, that because of the provided framework within the source files to implement your own data parser and the differences of our input data versus what PhyloWGS normally expects, we have modified the original PhyloWGS source code in order to comply with our data, instead of writing our own external parsers for the PhyloWGS native parsers; a solution we judged to be the best at the time. Therefore, we bundled our data pipeline with our own modified version of PhyloWGS and do not recommend it to clone it separately as it won't work with our data natively.

In addition to the native requirements of PhyloWGS, we used number of other libraries in the PhyloWGS-related part of the pipeline, all conveniently part of the standard Python library, with a notable exception of pandas. As part of the pipeline, we had to convert widely different output formats of the three tools into a single, common one. We made use of time-saving and more elegant pandas DataFrame management capabilities to neatly save our transformed data into the common format files.

3.4 NMF and LDA

For the NMF and LDA, and because of the nature of this tools, the requirements can be explained together.

To begin with, it should be mentioned that the operating system (OS) doesn't have to be any in specific if the Python language can be used for programming. Nonetheless, when working with both tools Windows was selected as the main OS.

For both models, the main library needed is scikit-learn [16], which is a tool for doing Machine Learning using the Python language. This tool will require the installation of NumPy and SciPy, which are another two libraries used for development in Python.

The final requirement for these two models to work would be the results from the previous phase, but more specifically, the Subclonal Structure files. From these files, a large dataset will be created containing the 500 tumour samples and the cancer cell fractions (CFF) values.

There were no problems during the installation of any of these libraries or using these tools.

In summary:

- Programming language: Python (2.7)
- Main library used: scikit-learn (0.19.2)
- Other libraries:
  - Matplotlib (1.2.0 or above)
  - seaborn (0.9.0)
  - NumPy (1.6.2 or above)
  - pandas (0.11 or above)
  - SciPy (0.11 or above)
- CCF dataset of 500 tumour samples.

Chapter 4 Design and Implementation

Since this project is only for analysis of cancer cells, it uses five different tools and each tool has its own implementation. There are no designs related whatsoever. Nonetheless, the construction of the pipeline does have a design and a different implementation.

This report contains the description of the different tools used for the analysis of cancer tumours samples, for which every tool required a different implementation. In the description of every implementation, it will be mention what the tool needed for running, the parameters that the tool would use and the output at the end.

Further, the pipeline needed to be designed and build in a way that every tool from phase one was running and producing a result. Then, in phase two and with the produced result from phase one, the other two tools could run the analysis of the data to extract the information needed, which in this case will be the relations between topics and documents.

4.1 PyClone

The main framework for PyClone contains pre-processing, analysing and post-processing. In pre-processing, 500 simulated tumour samples are inputted as source data. They are constructed as follows:

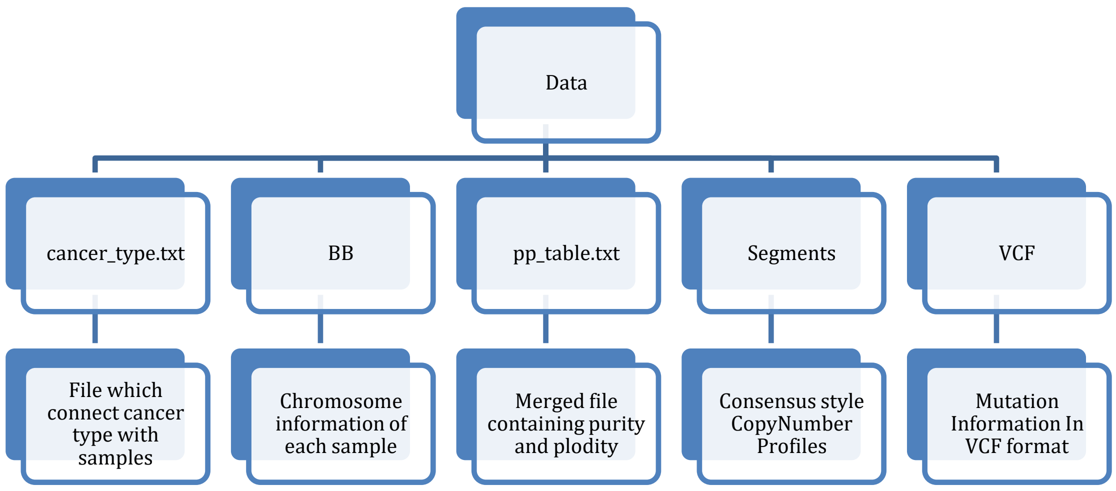

Data
cancer_type.txt
BB
pp_table.txt
Segments
VCF
File which connect cancer type with samples
Chromosome information of each sample
Merged file containing purity and plodity
Consensus style CopyNumber Profiles
Mutation Information In VCF format

Figure 3: How the input data for PyClone is built.

However, this data cannot be analysed directly. The required fields for PyClone are [4]:

- Unique ID (mutation_id)
- Number of reads covering the mutation which contain the reference allele (ref_count)

- Number of reads covering the mutation which contain the variant allele (var_count)
- Copy number of the cells in the normal population (normal_cn)
- Minor copy number of the cancer cell (minor_cn)
- Major copy number of the cancer cell (major_cn)

Thus, a translator program is needed to extract useful information from given files and generate the target file containing the required fields. Additionally, a configuration file is also needed for analysing. In such file, the working directory, density method, number of iterations and sample information are needed to be specified.

The next step is analysis, it can be done by calling the function “run_analysis” in PyClone. This stage will generate a folder named “trace” to store the clustering process.

In the final stage, all the result produced by PyClone are post-processed by R script. The output files from this process are the sub-clonal structure, which contains the cancer cell fraction (CCF) for each putative cluster, mutation assignment for each mutation and multiplicity.

PyClone adheres to the purpose of this project by exporting the interfaces for the final code integration, including:

- Prefix name for the inputted samples (prefix)
- The burnIn number N will drop out first N row of output data for analyzing (burnIn)
- Number of iterations (num_iter)
- Purity of input sample (purity)

4.2 Ccube

Ccube can be divided into three parts. The first part consists on passing the parameters to the main functions, which will be used during the process of formatting the input and the analysis, this includes seed values, number of repetitions, clusters number, the maximum number of iterations, number of cores to run, the path and the sample to analyse. However, Ccube will only be run if the parameter “run_ccube” is set to TRUE. Also, the number of samples which will be converted and analysed is based on the parameter “selectedfile” and “sampleNum”.

The second part is to do a pre-process where the chromosome, position, vaf_number, ref_number, the copy number of major and minor must be obtained from the VCF, segment and pp_table files of the samples selected. Then, this data will be stored into a TSV file on the target folders. Moreover, this data will be formatted in a suitable way for Ccube.

The third part is the analysis of the mutations and outputting the target files, where the accuracy and speed of the Ccube analysis will depend on the parameters passed. Consequently, according to the default value of the parameters, the average working time for the first ten samples is 54.33 seconds.

The following flowchart shows the different processes inside Ccube when it's running in the pipeline from start to end, and the user decisions when pre-processing the data:

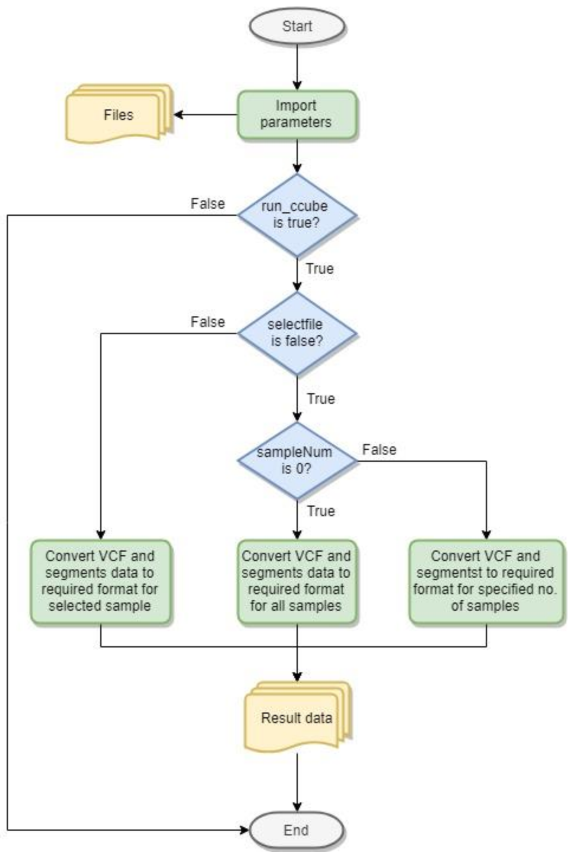

Start
Files
Import parameters
False
run_ccube is true?
True
False
selectfile is false?
True
sampleNum is 0?
False
True
Convert VCF and segments data to required format for selected sample
Convert VCF and segments data to required format for all samples
Convert VCF and segmentst to required format for specified no. of samples
Result data
End

Figure 4: Ccube flowchart inside the pipeline

The output files include three text files which are saved on different folders. Moreover, the subclone structure file has the number of simple semantic mutations (SSMS) and the CCF of all clusters. The mutation assignments file has the assignment of mutations in the different clusters and the CCF/proportion value from every mutation. The multiplicity file shows the chromosome, position, tumour copy numbers and multiplicity.

Finally, when testing the pipeline, the command shell will show the name of samples selected and the running time of analysing the sample. Also, the code will create a file named "test_time" which will show the Ccube running time for every sample in Ccube's result folder.

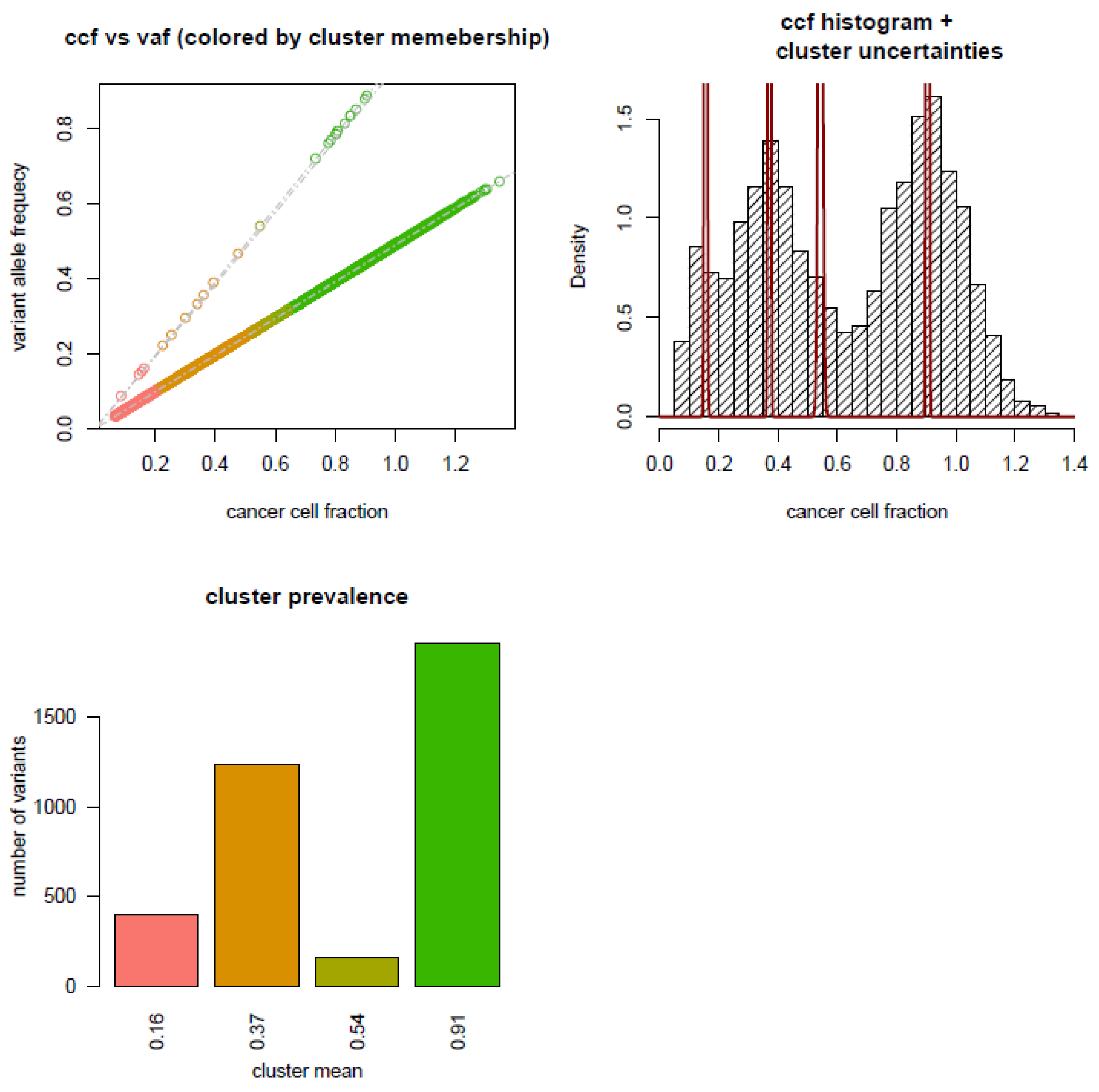


Figure 5: Summary of the Ccube results, the relationship between CCF and VAF, and the situation of clusters

4.3 PhyloWGS

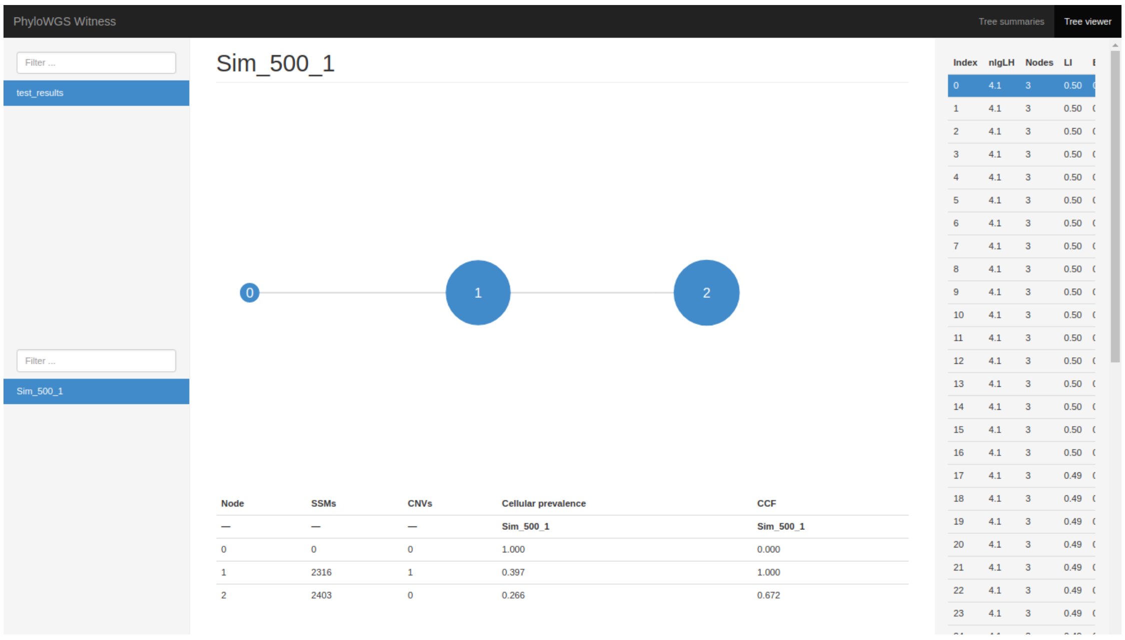
As previously mentioned, PhyloWGS is an end-to-end framework more than a library, and its native usage reflects that. In order to use PhyloWGS alone, a user must write a script that pre-parses the copy number variation (CNV) data (should a user wish so, as this is optional) for the main parser, which generates the inputs for the main scripts.

After executing the main script (either by running a single or multiple concurrent MCMC chains), a user must use another script to uncompress the resulting trees.zip file, restructuring the data into now three separate re-compressed .json files and move it to the result-viewing folder.

Lastly, a user must uncompress the three files again, launch the data indexing script, launch an HTTP server, open a browser and interpret the results, as visible in the **Figure 6**.

None of the interim scripts outputs simple text files with relevant data, none of them offers to disable the compression, and some of the most important metrics for our results (such as cancer cellular frequency (CCF) of a cluster, or determining the 'best' phylogenetic tree) is only done by front-end-supplying JavaScript files, executed only when the user navigates to the locally hosted interface to interpret the results.

After the initial impressions, we chose rather to focus on fast conversion of one of the interim results (to the common format), than the lengthy process of untangling and making changes to each and every script to get the output that we want.



Sim_500_1

Node	SSMs	CNVs	Cellular prevalence	CCF
—	—	—	Sim_500_1	Sim_500_1
0	0	0	1.000	0.000
1	2316	1	0.397	1.000
2	2403	0	0.266	0.672

Index	nlgLH	Nodes	LI
0	4.1	3	0.50
1	4.1	3	0.50
2	4.1	3	0.50
3	4.1	3	0.50
4	4.1	3	0.50
5	4.1	3	0.50
6	4.1	3	0.50
7	4.1	3	0.50
8	4.1	3	0.50
9	4.1	3	0.50
10	4.1	3	0.50
11	4.1	3	0.50
12	4.1	3	0.50
13	4.1	3	0.50
14	4.1	3	0.50
15	4.1	3	0.50
16	4.1	3	0.50
17	4.1	3	0.49
18	4.1	3	0.49
19	4.1	3	0.49
20	4.1	3	0.49
21	4.1	3	0.49
22	4.1	3	0.49
23	4.1	3	0.49

<u>Figure 6</u>: PhyloWGS native web-based interface for interpreting the results (for sample 'Sample_500_1'), Tree Viewer tab

Therefore, in order to adapt PhyloWGS efficiently to the main pipeline, we had to complete three main software engineering tasks, illustrated by **Figure 7**:

1. Supplied with the framework, create a custom pre-parser for the copy number variation data and a custom main parser for the raw .VCF file inputs and the output of the former to create the inputs for the main PhyloWGS pipeline: files ssm_data.txt and cnv_data.txt.
2. Create a shell script executing all the relevant native PhyloWGS scripts up to an acceptable level of the output. Execute the shell script from inside the pipeline class with the relevant parameters passed in each loop.
3. Devise a scalable, vectorized post-processing logic to extract and calculate the needed statistics in order to transform the compressed .json files into the common format text files.

Remarkably, we managed to achieve the running times of data post-processing logic only as slow as 0.14 seconds per sample, achieving the combined running time comparable to those of PyClone, when running with comparable parameters.

As with any tool that aims to reconstruct the tumour phylogeny, PhyloWGS can be customized via a plethora of parameters. However, in our data pipeline, we exposed the user only to the common ones that we judged are most likely to be changed:

1. Number of MCMC chains to be run concurrently;
2. Number of MCMC samples for each chain;
3. Number of burn-in samples for each chain;
4. Number of Metropolis-Hastings iterations;

Lastly, there are a couple of very important distinctions need to be made at his point. As may be noticeable in **Figure 6:**

1. The native way of choosing the 'best' phylogenetic tree is different in our pipeline and in the source code of PhyloWGS. We choose the best tree as the most likely one, in other words, the one with the highest reported likelihood. In the PhyloWGS source code, the 'best' tree is considered to be a tree that is most similar to the other trees [23]
2. Some of the clusters might have cellular prevalence and cancer cell fraction numbers approaching the value of 1. This is because of there are two different ways to define the cellular prevalence, and a user might be puzzled if he is accustomed to an opposite one. Cellular prevalence might be defined from a tumour perspective or from a sample perspective [24]:
   Sample cellular prevalence – fraction of cells that contain the mutation in the entire sample; whereas:
   Tumour cellular prevalence – fraction of cells that contain the mutation in the cancerous cells only;
   In PhyloWGS, cellular prevalence is defined from a sample perspective, and our CCF is calculated by dividing the cellular prevalence by the sample purity. The results demonstrated in **Figure 6** are consistent with the 'truth' values supplied by the project supervisor.

It is also important to note that the inclusion for the copy number variations (CNVs) data for running PhyloWGS is optional; skipping it would require one less change to the original source code (one less input parser would be required). However, since it's a distinctive feature of the tool we chose to include it. We have not made comparisons whether it contributes to an improvement in the final results in our case (compared with the 'truth' values) as we trusted the findings of the authors of PhyloWGS [20].

Lastly, PhyloWGS does not work well if a majority of MCMC chains report a polyclonal tumour structure and is unable to yield observable results in these cases [25]. Out of 138 simulated samples we tested PhyloWGS-related part of the pipeline with, PhyloWGS failed to converge in two cases (Samples 117 and 127). We believe this might have been due to:

1. An error in the interim ssm_data.txt outputs (as reported by other users [26])
2. Relatively low parameter values we tested the tool with, due to our hardware limitations.
3. The true nature of the tumours being, in fact, polyclonal.

In case of encountering a polyclonal tumour, the pipeline will report a runtime error during the execution of the native write_results.py script and the results for those specific samples will not be converted to the common format, but the execution for other samples will continue. Should adjusting the PhyloWGS-related pipeline parameters not solve the issue, we recommend trusting the inference of the other two Phase One tools about those samples.

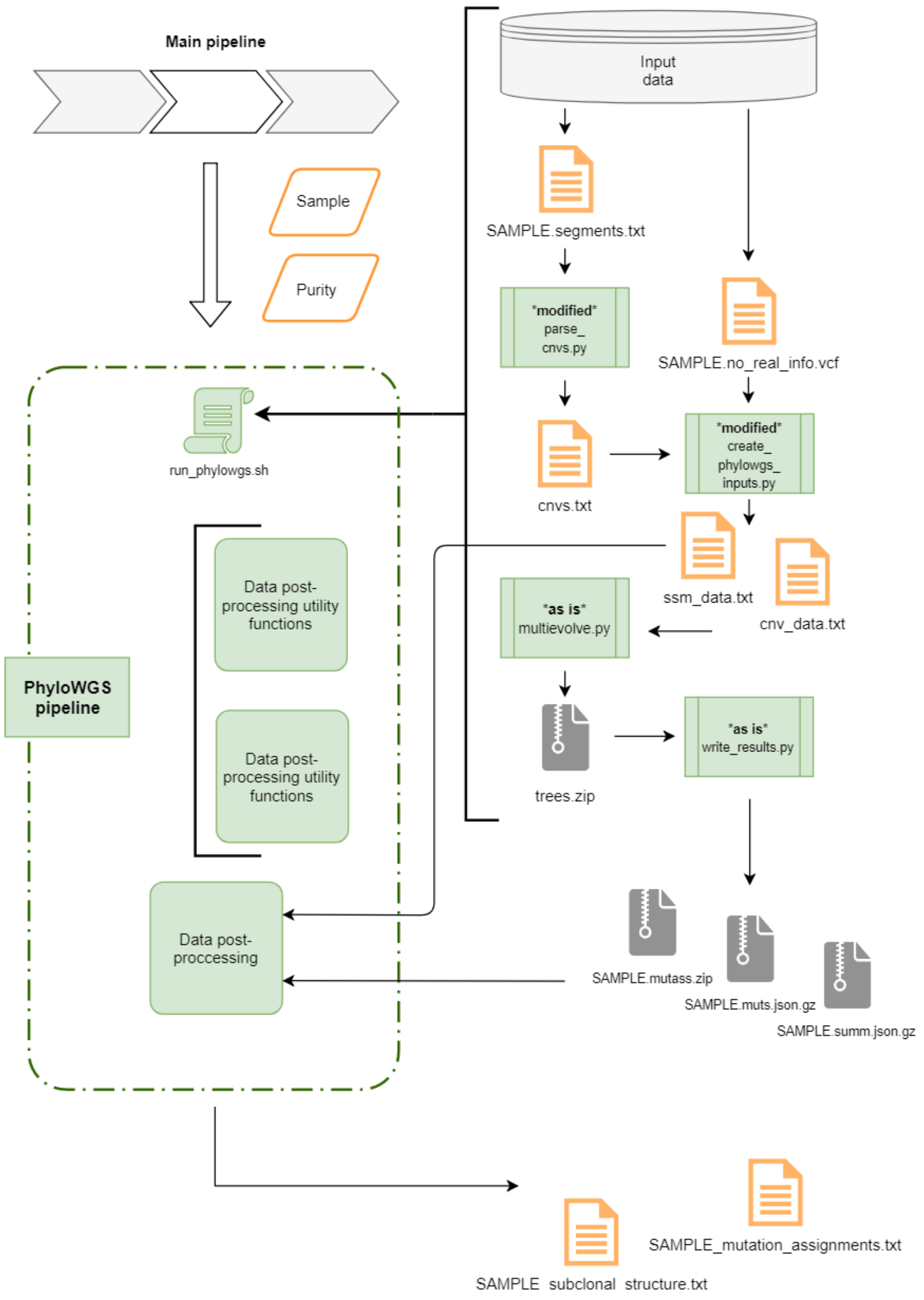


Figure 7: The workflow of PhyloWGS-related portion of the pipeline

4.4 Non-negative Matrix Factorization

To implement the NMF model, a data matrix was created using the 500 tumour sample files from the Subclonal Structure after phase one. Each sample file has a corresponding CCF values. Thus, all the CCF values from each file were counted and arranged in bands that start from 0.1 to 1 with step 0.1. The shape of the data matrix is 10 x 500 (row x column).

Figure 8 shows the average (mean) count of the total CCF value from each of the tumour samples.

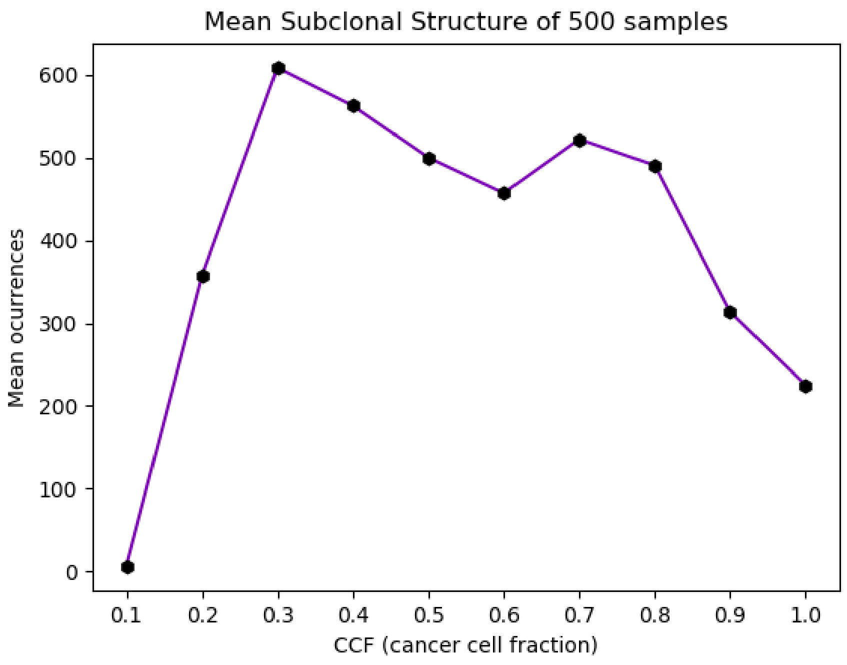


Figure 8: The CCF against the mean number of occurrences from the subclonal structure results during the first phase.

Regarding NMF, the data matrix can be defined as *A* with dimension *m x n*, where the row *m* defines the 10 different bands *{0.1, 0.2, ..., 1}* and where column *n* defines the value of tumour mutation from the 500 samples.

Also, for this project, data pre-processing such as using Count Vectorise to return the count of the data is not needed as the data matrix is already in numbers. But to get a better factorisation of the data matrix *A* into a feature matrix *W* and coefficient matrix *H*, we had to perform a few analyses such as tuning the parameters to find the less reconstruction error and estimating the number of components (*k*) for the model to determine the optimal parameters for the NMF model. The first phase of the analysis was to estimate the number of components to use for the NMF model. This was achieved by measuring the explained variance from the data matrix *A* to reveal how many variances (information) can be associated with each of the components. We measure this more directly by creating a method to utilise a metric library from scikit-learn to get the explain variance score from the NMF model and the data matrix *A*.

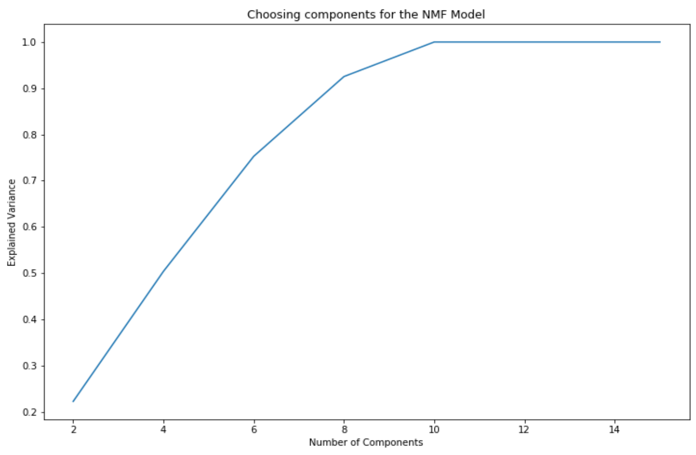


Figure 9: Estimating the number of components (k) for NMF model

Judging from **Figure 9** above, it evidently shows that 6 to 9 components are needed to approximately explain 92% of the variance. However, 99.95% to be specific of the variance can be explained using components from 10. The next phase of the analysis was to tune the model to determine the parameters with least reconstruction error. To achieve this, we created a method that takes in the data matrix *A* and some NMF model parameters and returned the reconstruction error. Considering this project the NMF parameters are, n_components, this states the number of components to use; init, this defines the initialisation method utilised for the model; max_iter, this determines the maximum iterations before timeout; random_state, this is stated to check reproducibility; l1_ratio, this defines the L1 and L2 penalties to regularise the model; alpha, this is used to control the strength of the regularization [19]. Subsequently, a form a grid search was performed to find the optimal parameters for the model. The table below shows the three smallest and largest reconstruction errors for different parameter combinations.

Table 1: Three combinations with the smallest and largest error.

Three Smallest Error	
Parameters	Reconstruction Error
(10, 'nndsvd', 50, 0, 0, 0)	1.024160
(10, 'nndsvd', 50, 1, 0.5, 0.75)	1.235355
(10, 'nndsvd', 50, 1, 1, 0.75)	1.653756
Three Largest Error	
(4, 'nndsvda', 2, 1, 1, 0)	61975.362915
(4, 'nndsvda', 2, 0, 1, 0.25)	61967.598853
(2, 'nndsvda', 2, 1, 1, 0)	61965.296342

Prior to searching for the optimal parameters for the model, it was explicit that most of the variance can be explained using 10 components. As a result, the maximum number of components used for tuning the model was 10. Furthermore, when tuning the model, it clearly shows that the higher the component the lower the reconstruction error of the model will be (**Figure *10*10***)*.

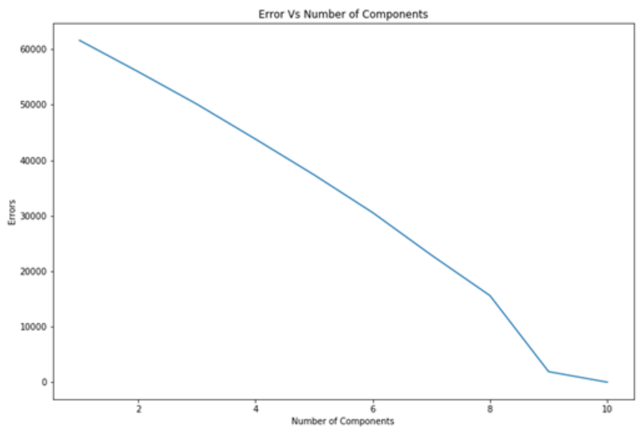


Figure 10: The number of components against the reconstruction error

The NMF model was applied to the data matrix A utilising the optimal parameters identified during the analysis phase. The number of components was set to 10; init = ‘nndsvd’, which means the initialization method utilised for the model is "Nonnegative Double Singular Value Decomposition"; max_iter = 50; random_state = 0; l1_ratio = 0, this means the L2 (Frobenius Norm) penalty for regularisation; alpha = 0. After factorising the data matrix *A* with dimension 10 x 500 (*m x n*), the model produced approximation matrices *WH*, where *W* is the feature (topic) matrix with dimension 10 x 10 (*m x k*) and *H* the coefficient matrix with dimension 10 x 500 (*k x n*). For the feature matrix, the rows are the bands of the CCF and the columns are the number of components (topics) used for the model. While for the coefficient matrix, the rows are the number of components and columns are the tumour samples. Furthermore, the feature matrix was normalised using a library part of the scikit-learn pre-processing libraries to get a unit value.

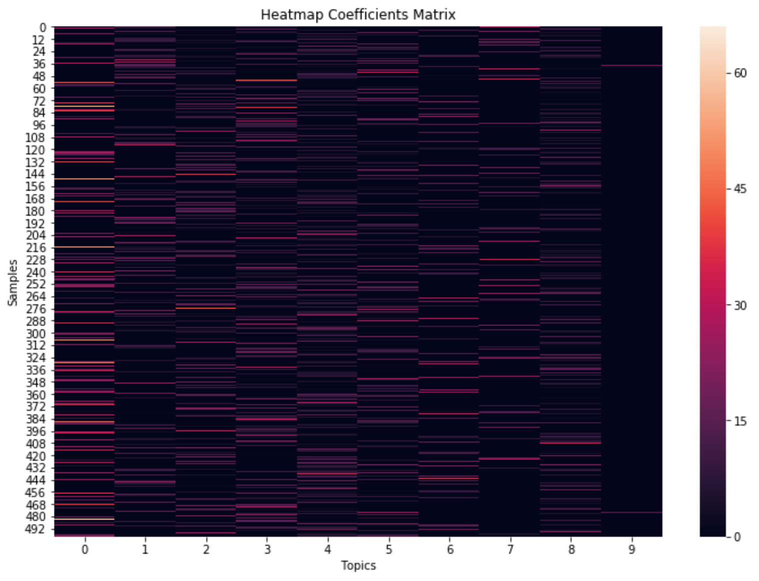


Figure 11: Heatmap of the coefficient matrix

4.5 Latent Dirichlet Allocation

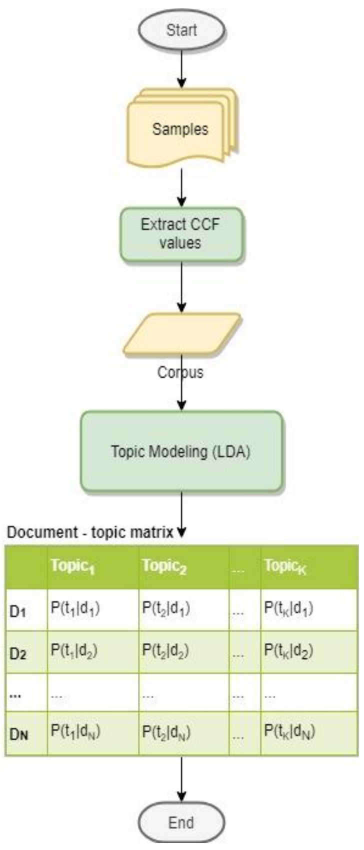
For this case in specific, there's a given dataset containing 500 tumour samples and their CCF values. As explained previously, LDA is utilized to model the relationships between these samples and extract the document-topic distribution matrix.

Based on a generative process describe in a previous chapter, the probability of a given dataset D = {D_1, …, D_s} is formalized as

$$p(D|\alpha,\beta) = \prod_{s=1}^{s=S} \int p(D_s|\varphi,\theta_s)p(\varphi|\beta)p(\theta_s|\alpha)d\theta_s$$

Formula 4: Probability of a given dataset D, from Topic modelling for cluster analysis of large biological and medical datasets [7].

The LDA is implemented in this project taking as a document every tumour sample containing the values from every one of the ten CCF (cancer cell fraction) bands. Thus, creating a corpus with 500 documents (or rows) with ten different bands (or columns) {0.1, 0.2, …, 1} and the number of mutations from each sample working as our text corpora.



	$Topic_1$	$Topic_2$	…	$Topic_K$
D1	$P(t_1\|d_1)$	$P(t_2\|d_1)$	…	$P(t_K\|d_1)$
D2	$P(t_1\|d_2)$	$P(t_2\|d_2)$	…	$P(t_K\|d_2)$
…	…	…	…	…
DN	$P(t_1\|d_N)$	$P(t_2\|d_N)$	…	$P(t_K\|d_N)$

Figure 12: The workflow of the topic modelling using Latent Dirichlet Allocation.

Moreover, since the words from the corpus are only numbers, there was no pre-processing needed for this. Normally, the pre-processing would include lemmatizing the words, removing all stop words, removing numbers, etc.

The corpus for this project was created reading the Subclonal Structure files resulted from phase one and extracting the CFF values from every file (represented as “proportion”), then these CCF values were inserted into an array and sent to the LDA tool for topic extraction.

In order to get the best results from the LDA tool utilized, the hyperparameters used were decided after doing an analysis with a different tool. This tool is *GridSearchCV* [17], which it’s also from scikit-learn, and it helps to look for the best hyperparameters for certain model (in this case, LDA model) using cross-validation. This tool takes the log-likelihood (score) and the perplexity obtained from the results to analyse the best set of hyperparameters.

In this project what it was tried to achieve was high likelihood and low perplexity. In the case of LDA, a lower perplexity score indicates better performance. Hence, with a test set of M documents, the perplexity would be:

$$perplexity(D_{test}) = \exp\left\{-\frac{\sum_{d=1}^{M} \log p(\mathrm{w}_d)}{\sum_{d=1}^{M} N_d}\right\}.$$

Formula 5: Perplexity for a test set of M documents [5].

As for the log likelihood, it can also be expressed with:

$$\log p(\mathrm{w} \mid \alpha, \beta) = E_q[\log p(\theta, \mathrm{z}, \mathrm{w} \mid \alpha, \beta)] - E_q[\log q(\theta, \mathrm{z})].$$

Formula 6: Likelihood of a document [5].

Nevertheless, when searching for the best perplexity and log likelihood, it was better to rely on a tool that could help us to achieve these values without going deeper into working with many calculations. Thus, making *GridSearchCV* a powerful tool for us.

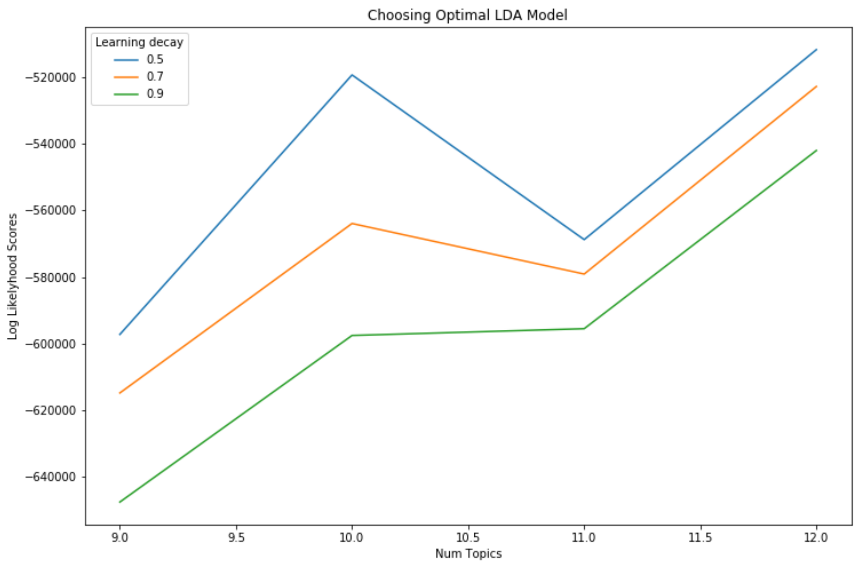


Figure 13: Choosing the Optimal LDA model using log-likelihood score.

In the plot above (**Figure 13**) can be seen that picking an LDA model with 12 topics, a learning decay of 0.5, a number of 11 maximum iterations and with a random state of 2018 is better for this dataset. Giving as a result:

- Best Log Likelihood Score: -511789.38469719695
- Model Perplexity: 2.1247306713534675

The following graph shows the results of using the optimal set of hyperparameters:

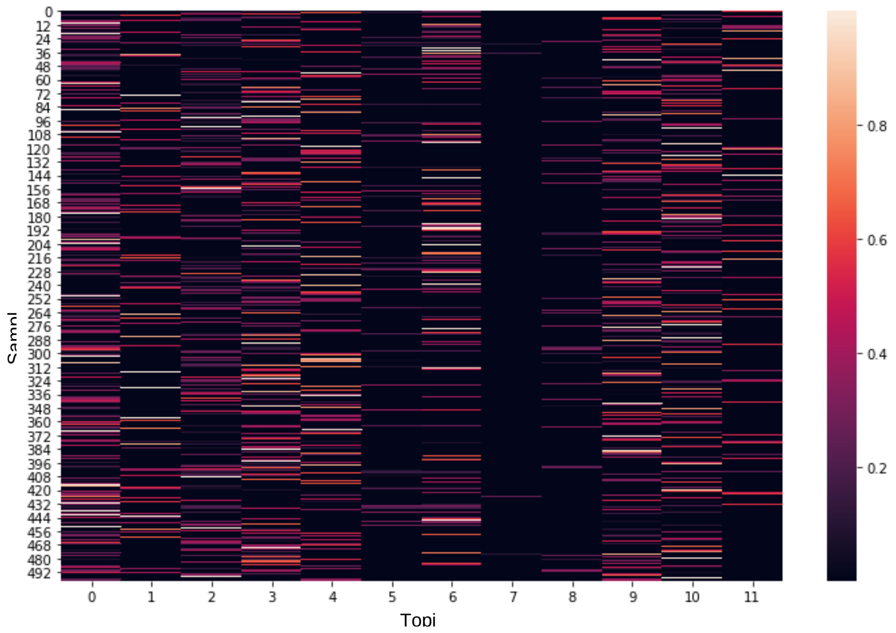


Figure 14: Heatmap showing 12 topics and 500 samples with probability values from 0 to 1.

Chapter 5 Evaluation

In addition to the development of pipeline, we were interested to see which one of the Phase One tools performed the best and were able perform relevant analyses thanks to the supplied 'truth' values for our generated data. We chose to approach the evaluation of Phase One at a couple of angles:

1. Measuring the running times of each tool,
2. Measuring the accuracy in determining the current number of SSM clusters in the samples that were run with all three tools,
3. Measuring the mean squared error of mutation assignments to each SSM cluster.

Performing such analyses objectively proved to be challenging, mainly because of the difference between the underlying methods that are employed within each tool and the assumptions about the true nature of the data they make, making the tools share little resemblance in their running parameters. This posed several design decisions that we had to make while performing our evaluation of the tools, like:

1. Should we approach the evaluation of the tools from their running time perspective, that is, should we aim to make the tools run for roughly the same time and measure how their accuracy is affected, or conversely:
2. Should we approach the evaluation of tools from their accuracy perspective, that is should we aim to make the tools report the results of roughly the same level of accuracy and measure the time it took to reach that level instead;
3. How should our analysis reflect the different assumptions each tool makes about the data?

Much of the evaluation analysis and accompanying figures are described in the appendix B of our report, however for the purpose of this section and the conclusions of our work that follow, we can state with full confidence that Ccube performed the best from both running time and accuracy perspective.

During the first phase of this project, three tools were implemented to carry out the analysis of 500 cancer samples. These cancer samples were a mix of different files: VFC files, for mutation information; segments files, with a consensus style copy number profiles; and the BB files, containing the BAF values (barrier-to-autointegration factor).

These three tools had different input data which was previously provided by the supervisor. Since each tool required slight changes to the our dataset, there was a pre-process step required to get the data into a correct format.

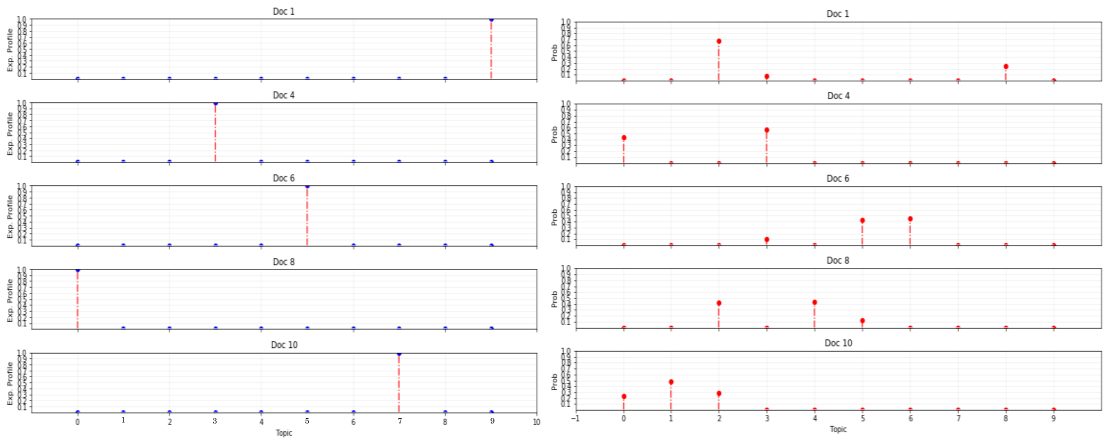
Furthermore, after setting the necessary parameters for the tools, each one was ran obtaining the results required for the second phase. The necessary files for the second phase were the Subclonal Structures containing the clusters created and the CCF values needed for the topic extraction. These files were examined in order to extract the CCF values and then create a dataset to be used with each of the models for topic extraction. The models selected for doing topic extraction from the 500 cancer samples were the Non-negative Matrix Factorization and Latent Dirichlet Allocation.

During the second phase, two models were used for topic extraction: Non-negative Matrix Factorization and Latent Dirichlet Allocation. In order to carry out an evaluation of these models, it will be shown a comparison to see how similar or different they were when doing the topic extraction.

Farther, when working with both models, the same dataset was used to ensure that the results from both can be compared and the differences accounted. Such dataset was created using the CCF values {0.1, …, 1.0} from the 500 tumour samples giving.

To make the comparisons between both models, the most important value that must be equally fed to the models is the number of topics to extract. Hence, after meticulously examining the best hyperparameters for both models, it was decided that the number of topics to be extracted would be ten.

Thus, the analysis was carried out selecting ten topics in both models. The results are as shown below.



<u>Figure 15</u>: Random documents for comparisons. Left, document – topic table from the NMF model. Right, document – topic table from the LDA model.

Furthermore, from the previous plots (**Figure 15**), it can be seen that the results are very different. Although, it's very noticeable that there isn't much similarity between both results in this random sample, it's visible that more than one document has a small resemblance. This means there could be more small matches in the document – topic matrices from both models.

Chapter 6 Conclusion

6.1 Phase One

During Phase One of building the data pipeline, we adapted three separate tools for uncovering the tumour subclonal compositions. Due to our hardware limitations, we were unable to test these tools with the recommended parameter values, however the pipeline is capable of running the tools with any parameter values a user requires. In our testing and evaluation phase, we only tried the tools with relatively low parameter values that our hardware was capable of handling.

Due to the different nature of the algorithms behind each of the Phase One tools and little-to-none adaptability of the same parameter values, running time and accuracy analyses between PyClone, Ccube and PhyloWGS should be taken with a grain of salt. In practise, a fair comparison could only be made using the tools with at least their recommended default parameter values. Other confounding factors might contribute to such analyses, such as:

1. Different programming languages used to implement the tools (Python for PyClone, R for Ccube, Python/C++ for PhyloWGS)
2. Different distribution and area of focus between the tools (PyClone and Ccube are distributed as libraries, PhyloWGS is distributed as an end-to-end piece of software that requires many areas of attention)
3. Diffencies between the levels of support by the tool creators (Some tools might be regularly updated and improved in the future, others might be updated rarely if at all)
4. Differences between the efficiencies in our implementations of the tools.

Regardless of these shortcomings, during our testing of the Phase One tools, we observed that Ccube to be orders of magnitude faster than the PhyloWGS and PyClone, that closely follow each other. Hence, using Ccube, we were able to generate (and bundle with our submission) the cluster cancer cell fractions for the whole of our dataset (500 tumour samples) to be used in the Phase Two of the project: performing latent Dirichlet allocation and non-negative matrix factorization.

6.2 Phase Two

Similarly, during Phase Two, from the evaluation and comparison of the topic extraction models we can conclude that when working with models for topics extraction and numbers instead of words, there are not clear results if we just stop the analysis after the topic extraction. Still, it can be seen some similitudes in the results. Similarly, it should be mentioned that when running both models, LDA took more time to complete the execution because of the use of GridSearchCV to find the optimum model.

6.3 Future Work

For Phase One of the project (running PyClone, Ccube and PhyloWGS) the future work consists of the following:

1. Finishing to vectorize the remaining portions of the code in order to achieve a better scalability.
2. Incorporating design patterns to transform our pipeline into a framework or a template; make adding additional tools or libraries for recovering the subclonal composition of tumours seamless and easy.
3. Finish running all of the 500 simulated samples with all of the three tools.
4. Performing a final, comprehensive code review, changing code style to be more readable for the curious users and consistent with the styling guidelines of Python 2.7 and R.
5. Making the pipeline insensitive to the file naming patterns that were evident to our simulated input data. Right now, the pipeline expects segments data to be named 'Sample.segments.txt' and .VCF files to be named 'Sample.no_real_info.vcf'. In reality, only the identifying sample name should matter. We believe that would make our pipeline more flexible for a user to use.
6. For PhyloWGS only: encapsulate the changes that we have made to the original source code of the tool and compile instructions in the Docker file to clone the original repository with the overwrite of the files that we have modified. Note that this could render the related portion of the pipeline unusable should a significant change in the original repository occur.
7. For PhyloWGS only: follow the development of the tool and make significant changes to the pipeline once the authors enable the support of easily readable output files.

As for Phase Two, the future work could be:

1. Find a better model for finding the best set of hyperparameters for NMF model.
2. More analysis to obtain more in-depth data from the topic extraction.
3. Obtain if the topic shows any closeness with the cancer types.
4. More pre-processing of the values to obtain more topics and different probability distributions and/or weights.
5. Try using different benchmarking models besides the ones from scikit-learn.

Chapter 7 Contributions

In this chapter is presented a description of everyone's contribution in both the final product development and submitted report.

7.1 Final product

Tool/item/doc	Student(s)
PyClone	Mingfeng Liu
Ccube	Yuhong Lin
PhyloWGS	Lukas Rubikas
NMF	Razak Nart
LDA	Jesus Vera
Pipeline	Mingfeng Liu, Lukas Rubikas
Code review	Mingfeng Liu, Lukas Rubikas
Docker	Yuhong Lin, Mingfeng Liu
User Manual/README.txt	Lukas Rubikas, Jesus Vera

7.2 Report

Chapter/Section	Student(s)
Abtract	Jesus Vera, Razak Nart
Chapter 1 – Introduction	Jesus Vera
Chapter 2 – Background Survey	Jesus Vera
Section 2.1 – PyClone	Mingfeng Liu
Section 2.2 – Ccube	Yuhong Lin
Section 2.3 – PyhloWGS	Lukas Rubikas
Section 2.4 – NMF	Razak Nart
Section 2.5 – LDA	Jesus Vera

Chapter 3 – Requirements	Jesus Vera
Section 3.1 – PyClone	Mingfeng Liu
Section 3.2 – Ccube	Yuhong Lin
Section 3.3 – PyhloWGS	Lukas Rubikas
Section 3.4 – NMF and LDA	Jesus Vera
Chapter 4 – Design and Implementation	Jesus Vera
Section 4.1 – PyClone	Mingfeng Liu
Section 4.2 – Ccube	Yuhong Lin
Section 4.3 – PhyloWGS	Lukas Rubikas
Section 4.4 – NMF	Razak Nart
Section 4.5 – LDA	Jesus Vera
Chapter 5 – Evaluation	Jesus Vera
Section 5.1 – First Phase	Lukas Rubikas
Section 5.2 – Second Phase	Jesus Vera and Razak Nart
Chapter 6 – Conclusion	Lukas Rubikas and Jesus Vera
Chapter 7 – Contributions	Jesus Vera
Section 7.1 – Final product	Jesus Vera
Section 7.2 – Report	Jesus Vera
References	Everyone
Appendix A – User Manual	Lukas Rubikas, Jesus Vera
Appendix B – Runtime and accuracy analysis of Stage One tools	Lukas Rubikas

References

[1] Roth, A., *et al.* PyClone: statistical inference of clonal population structure in cancer. Nature Methods, 11(4), p.396. 2014.

[2] 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. Nature, 467(7319), p.1061. 2010.

[3] Harismendy, O., *et al.* Detection of low prevalence somatic mutations in solid tumors with ultra-deep targeted sequencing. Genome biology, 12(12), p. R124. 2011.

[4] Yuan, K., Macintyre, G., Liu, W. and Markowetz, F. Ccube: A fast and robust method for estimating cancer cell fractions. 2018.

[4] Roth, *et al.* PyClone: statistical inference of clonal population structure in cancer. PMID: 24633410. Bitbucket repository: https://bitbucket.org/aroth85/pyclone

[5] Lee, D., Seung H. Learning the parts of objects by non-negative matrix factorization. Nature. 1999;401(6755):788-791.

[6] Lee. D., Seung, H. Algorithms for non-negative matrix factorization. In Advances in neural information processing systems. 2001; 556-562.

[7] Müller, A., Guido, S. Introduction to Machine Learning with Python. 1st ed. Sebastopol: O'Reilly Media, Inc.; 2016.

[8] Shahnaz, F., *et al.* Document clustering using nonnegative matrix factorization. Information Processing & Management. 2006;42(2):373-386.

[9] Tsuge S, Shishibori M, Kuroiwa S, Kita K. Information Retrieval Using Non-Negative Matrix Factorization. IEEJ Transactions on Electronics, Information and Systems. 2004;124(7):1500-1506.

[10] Salim, W., Santika, D. Human Facial Expression Recognition Using Two-dimensional Non-Negative Matrix Factorization. Procedia Engineering. 2012;50:758-767.

[11] Devarajan, K. Nonnegative Matrix Factorization: An Analytical and Interpretive Tool in Computational Biology. PLoS Computational Biology. 2008;4(7):e1000029.

[12] Lee, C., *et al.* Simultaneous Non-Negative Matrix Factorization for Multiple Large Scale Gene Expression Datasets in Toxicology. PLoS ONE. 2012;7(12):e48238.

[13] Blei, D., Ng, A. and Jordan, M. Latent Dirichlet Allocation. Journal of Machine Learning Research 3. 2003.

[14] Hoffman, M., Blei D., C. Wang, J. Paisley. Latent Dirichlet Allocation. Reference from Schikit-learn.org: https://scikit-learn.org/stable/modules/decomposition.html#latentdirichletallocation

[15] Zhao, W., *et al.* (2014). Topic modeling for cluster analysis of large biological and medical datasets. BMC Bioinformatics.

[16] Scikit-learn: Machine Learning in Python, Pedregosa, *et al.*, JMLR 12, pp. 2825-2830, 2011. Scikit-learn.org tool: https://scikit-learn.org/

[17] GridSearchCV – scikit-learn 0.19.2 documentation. Scikit-learn.org: https://scikit-learn.org/0.19/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

[18] Decomposing signals in components (matrix factorization problems) — scikit-learn 0.20.1 documentation. Scikit-learn.org: https://scikitlearn.org/stable/modules/decomposition.html#nmf

[19] Sklearn.decomposition.NMF — scikit-learn 0.20.2 documentation. Scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html

[20] Deshwar, A. G., *et al.* PhyloWGS: Reconstructing subclonal composition and evolution from whole-genome sequencing of tumors. Genome Biology 16. 35. 2015.

[21] GSL – GNU Scientific Library: http://www.gnu.org/software/gsl/

[22] PhyloWGS GitHub repository: https://github.com/morrislab/phylowgs

[23] Q. Morris, On the topic of determening the best PhyloWGS tree: https://github.com/morrislab/phylowgs/issues/61#issuecomment-308578688

[24] G. Ha, On the topic of differences between definitions of cellular prevalence: https://github.com/gavinha/TitanCNA/issues/11#issuecomment-336874279

[25] Snippets of PhyloWGS source code where the removal of polyclonal trees is evident: https://github.com/morrislab/phylowgs/search?q=polyclonal&unscoped_q=polyclonal

[26] GitHub's user suggestion for dealing with PhyloWGS's polyclonal trees: https://github.com/morrislab/phylowgs/issues/95#issuecomment-433264817

Appendix A User Manual

Project 18: Data pipeline for extracting signatures of evolutionary dynamics from tumours

This is a User Manual file describing a piece of software that is accompanying the submission report of University of Glasgow MSc Data Science programme's MSc Team Project course, authored by Razak Nart, Mingfeng Liu, Lukas Rubikas, Yuhong Lin and Jesus Vera and supervised by Dr Ke Yuan.

Our task, described in the report and implement in this pipeline, is twofold:

1. Reconstructing the subclonal composition of tumours given the data about clusters of simple somatic mutations (SSMs) and their corresponding copy number variations (CNVs) using the following tools and libraries:
   1.1. PyClone, a Python library developed by Andrew Roth et al. and adapted to this data pipeline by Mingfeng Liu,
   1.2. Ccube, a R library developed by Ke Yuan and adapted to this data pipeline by Yuhong Lin,
   1.3. PhyloWGS, an end-to-end open-source software written with Python/C++, developed by Quaid Morris et al. and adapted to this data pipeline by Lukas Rubikas,
2. Performing Latent Dirichlet allocation (LDA), implemented for this project by Jesus Vera, and non-negative matrix factorization (NMF), implemented by Razak Nart, using these mutation assignments for each and every SSM in every tumour sample.

Is this therefore why we will refer to action steps described in (1) as the Stage One of the project and action steps described in (2) as Stage Two.

Our data

The test data that was used while developing this tool was generated and supplied to us by the project supervisor, Dr Ke Yuan, and was structured as following below:

- `` `data/` ``:
  - `` `VCF/` `` folder, containing the mutation information for every tumour sample, formatted as `` `SAMPLE_NAME.no_real_info.vcf` ``, in a widely recognized VCF format,
  - `` `Segments/` `` folder, containing copy number profiles for each tumour sample, formatted as `` `SAMPLE_NAME.segmets.txt` ``,
  - `` `pp_table.txt` `` text file, containing the purity and ploidy information for each sample.

It is therefore our requirement that in order to run our pipeline, the input data should follow the same pattern and structure.

Moreover, when working on Stage Two of the project, we needed to have data already processed by Stage One with the 500 result files that would include the proportion values. In order to do this, Ccube was run (as it was the fastest one to run). The data is located as follows:

- `data/Data/Phase_two/`:
    - `Subclonal_Structure/` folder, containing the 500 subclonal structure files, formatted as `SAMPLE_NAME_subclonal_structure.txt`.

Our project structure

In this data pipeline we combine a number of different tools and algorithms, therefore it is necessary to be aware of how our source code is structured and how specific parts of it can be accessed, modified or updated, should there be a need for it. Our source code is therefore structured as the following:

- `team-project-cs/`:
    - `docker/`, containing necessary files and instructions how to set up a Docker image for our pipeline with the numerous dependencies we require
    - `Pipeline/`, all of the source code of Stage One and Stage Two
    - `data/`, the input folder for Stage One and Two, with the folder structure and file-naming pattern described in the "Our data" section in this README.md,
    - `ccube/`, containing all relevant files and folders used by/generated while using Ccube
    - `pycloneL/`, containing source code of our adaptation of PyClone library and the post-processing source code of its interim output,
    - `phylowgs/`, containing files associated with PhyloWGS-related part of our project, most notably:
        - `phylowgs.source.code.project18`, containing cloned GitHub repo of PhyloWGS with our modifications to make it adaptable to our project,
        - `phylowgs.results` (generated), containing the final output of PhyloWGS (but rather interim output for our pipeline) should a user would like to use PhyloWGS's own inference about the tumour subclonal composition capabilities
    - `LDA/`
        - `Code/` source code of our implementation of the LDA algorithm, part of Stage Two
    - `NMF/`
        - `Code/` source code of our implementation of the NMF algorithm, part of Stage Two
    - `inputTmp/` (generated), temporary folder for storing interim PyClone results (can be disabled)
    - `results/` (generated), containing the results, converted to a common format, of each of the Stage One tools and possible input folders of Stage Two algorithms:
        - `ccube/`
            - `multiplicity/` (unused) sample SSMs multiplicities, as inferred by Ccube

        - `mutation_assignments/` sample SSMs mutation assignments, as inferred by Ccube
        - `subclonal_structure/` sample subclonal structure, as inferred by Ccube
        - `runtimes.tsv`, a merged file containing the Ccube running times (in seconds) for each sample
    - `pyclone/`
        - `multiplicity/` (unused) sample SSMs multiplicities, as inferred by PyClone
        - `mutation_assignments/` sample SSMs mutation assignments, as inferred by PyClone
        - `subclonal_structure/` sample subclonal structure, as inferred by PyClone
        - `runtimes.tsv`, a merged file containing the PyClone running times (in secods) for each sample
    - `phylowgs/` *[1]
        - `mutation_assignments/` sample SSMs mutation assignments, as inferred by PhyloWGS
        - `subclonal_structure/` sample subclonal structure, as inferred by PhyloWGS
        - `runtimes.tsv`, a merged file containing the PhyloWGS running times (in seconds) for each sample
    - `pipeline.py`, the entry point of using the pipeline, combining all of the programming logic of our tools.

*[1] `multiplicity/` folder is skipped by the common format output generator of the PhyloWGS interim output.

Our output

Since all of the Stage One tools outputs their results in widely different formats, it was necessary to agree upon a common output, which our resulting data had to be transformed to. The common format was suggested by the project supervisor Dr Ke Yuan and must follow such pattern:

1. In `subclonal_structure/` folder, for each sample in the input data, resulting file must be named as `SAMPLE_NAME_subclonal_structure.txt` and must have the following columns:
   `| cluster | n_ssms | proportion | ccf (optionally) |`

2. In `mutation_assignments/` folder, for each sample in the input data, resulting file must be named as `SAMPLE_NAME_mutation_assignments.txt` and must have the following columns:
   `| chr | pos | cluster | proportion | ccf (optionally) *[2] |`

3. In `multiplicity/` *[3] folder (optional), for each sample in the input data, resulting file must be named as `SAMPLE_NAME_multiplicity.txt` and must have the following columns:
   `| chr | pos | tumour_copynumber | multiplicity |`

*[2] in Ccube common format results, this column is named `ccfmean`, in PyClone - `average_ccf`, in PhyloWGS it is left as just `ccf`.
*[3] `multiplicity/` data is unused in our project and was skipped in the common output of PhyloWGS results.

The output for Stage Two tools is very different from the previous Stage. The resulting output from Stage Two is a series of plots in PNG format showing the how the selection for the best models was accounted and the results; also, a document-topic CSV file will be saved. Normally, this is what the output should be from both, the NMF and LDA processes:

1. NMF:
   1. `best_component.png`
   2. `nmf_error.png`
   3. `coe_matrix.png`
   4. `nmf_topic_result.png`
   5. `nmf_topic.csv`
2. LDA:
   1. `mean_subclonal_500_samples.png`
   2. `optimal_lda_model.png`
   3. `heatmat_doc_topics.png`
   4. `random_topics_results.png`
   5. `document_topic.csv`

Both of the CSV files for the topics will have the format with the columns:

```
|   -   | Topic 1 | Topic 2 | ... | Topic K |
| Doc 1 | Val 1.1 | Val 1.2 | ... | Val 1.K |
| Doc 2 | Val 2.1 | Val 2.2 | ... | Val 2.K |
|  ...  |   ...   |   ...   | ... |   ...   |
| Doc N | Val N.1 | Val N.2 | ... | Val N.K |
```

Our dependencies

We supplied a `docker/` folder containing all the installation instructions for the numerous dependencies we use throughout our pipeline in a form of a Docker file, and a test script which runs it and tests our pipeline with small parameter values. Therefore, the only true dependency the user should install before using the pipeline is Docker (https://www.docker.com/) and a way to run our `test.sh` script, should the user's operating system (most notably Windows) not have such a native capability.

IMPORTANT NOTE: On Windows System it can be run `test_windows.sh`. Nonetheless, please first read carefully the instructions inside the README file in the `docker/` folder.

Running the pipeline

Running `python pipeline.py --test` inside the `Pipeline/` directory provides a useful summary for the pipeline parameters:

```
usage: pipeline.py [-h]
```

```
                  [--path WORKPLACE]
                  [--random-samples RANDOM_SAMPLES]
                  [--samples-to-run SELECTED_SAMPLES]
                  [--phylowgs-burnin-samples PHWGS_BURNIN_SAMPLES]
                  [--phylowgs-mcmc-samples PHWGS_MCMC_SAMPLES]
                  [--phylowgs-mh-iterations PHWGS_MH_ITERATIONS]
                  [--pyclone-burnin-samples PYCLONE_BURNIN_SAMPLES]
                  [--pyclone-mcmc-iterations PYCLONE_MCMC_ITERATIONS]
                  [--ccube-max-clusters CCUBE_MAXCLUSTER]
                  [--ccube-vbem-max-iters CCUBE_VBEM_MAX_ITERS]
                  [--ccube-repeats CCUBE_REPEAT]
                  [--ccube-random-seed RANDOM_SEED]
                  [--num-cores NUM_CORES]
                  [--run-pyclone] [--run-ccube] [--run-phylowgs]
                  [--pyclone-delete-tmp-folder]
                  [--lda-nmf-input RESULTS_FOLDER]
                  [--run-nmf] [--run-lda]

Pipeline

    optional arguments:
      -h, --help            show this help message and exit
      --path WORKPLACE      Specifies working directory for analysis.
                            All paths in the rest of the PyClone and
                            Ccube files are relative to this
      --random-samples RANDOM_SAMPLES
                            Number of randomly selected samples to
                            run
      --samples-to-run SELECTED_SAMPLES
                            A newline-seperated file of explicitly-
                            stated tumour sample names to be tested
                            with Stage One tools (PyClone, Ccube,
                            PhyloWGS)
      --phylowgs-burnin-samples PHWGS_BURNIN_SAMPLES
                            Number of burn-in samples for PhyloWGS
                            (default: 1000)
      --phylowgs-mcmc-samples PHWGS_MCMC_SAMPLES
                            Number of true MCMC samples for PhyloWGS
                            (default: 2500)
      --phylowgs-mh-iterations PHWGS_MH_ITERATIONS
                            Number of Metropolis-Hastings iterations
                            for PhyloWGS (default: 5000)
      --pyclone-burnin-samples PYCLONE_BURNIN_SAMPLES
                            Number of burn-in samples for PyClone (10
                            percent of total MCMC samples are
                            suggested in the official documentation,
                            50 used here)
      --pyclone-mcmc-iterations PYCLONE_MCMC_ITERATIONS
                            Number of MCMC iterations for PyClone
                            (default: 500)
      --ccube-max-clusters CCUBE_MAXCLUSTER
                            Maximum number of clusters for Ccube
                            (default: 6)
      --ccube-vbem-max-iters CCUBE_VBEM_MAX_ITERS
                            Number of VBEM iterations for Ccube
                            (default: 1000)
      --ccube-repeats CCUBE_REPEAT
                            Number of repeated Ccube runs for each
                            candidate number of clusters (default: 1)
      --ccube-random-seed RANDOM_SEED
```

```
                        Random seed (used by Ccube), required to
                        run the the tool deterministically
  --num-cores NUM_CORES
                        Number of processor cores to be employed
                        in computations concurrently (used by
                        Ccube and PhyloWGS) (default: 1)
  --run-pyclone         Flag for running PyClone (default: False)
  --run-ccube           Flag for running Ccube (default: False)
  --run-phylowgs        Flag for running PhyloWGS (default:
                        False)
  --pyclone-delete-tmp-folder
                        Flag to delete temporary folder
                        ("./inputTmp") containing configuration
                        files generated while performing PyClone
                        runs (default: False)
  --lda-nmf-input RESULTS_FOLDER
                        Input folder for LDA and/or NMF analysis
                        (may be one of the result folders of
                        Stage One tools)
  --run-nmf             Flag for running NMF (default: False)
  --run-lda             Flag for running LDA (default: False)
```

Therefore, it is possible to launch each of the Stage One tools separately. For example:

```
python pipeline.py --run-phylowgs --num-cores 4 --phylowgs-burnin-
samples 2 --phylowgs-mcmc-samples 20 --phylowgs-mh-iterations 50

python pipeline.py --run-pyclone --pyclone-burnin-samples 2 --
pyclone-mcmc-iterations 20

python pipeline.py --run-ccube --ccube-max-clusters 6 --ccube-
repeats 1 --ccube-vbem-max-iters 1000
```

Or, all of them together:

```
python pipeline.py --run-pyclone --run-ccube --run-phylowgs --
pyclone-burnin-samples 2 --pyclone-mcmc-iterations 20  --ccube-max-
clusters 6 --ccube-repeats 1 --ccube-vbem-max-iters 1000 --
phylowgs-burnin-samples 2 --phylowgs-mcmc-samples 20 --phylowgs-mh-
iterations 50
```

If a user wants to use the pipeline for a specific subset of the samples, he can create a newline-seperated file, containing the sample names he wishes to use, for example `samples.to.run.tsv` containing:

```
Sample_500_22
Sample_500_23
Sample_500_24
Sample_500_25
...
```

And supplying the file as a `--samples-to-run` parameter:

```
python pipeline.py --run-pyclone --run-ccube --run-phylowgs --
pyclone-burnin-samples 2 --pyclone-mcmc-iterations 20  --ccube-max-
clusters 6 --ccube-repeats 1 --ccube-vbem-max-iters 1000 --
```

```
phylowgs-burnin-samples 2 --phylowgs-mcmc-samples 20 --phylowgs-mh-
iterations 50 --samples-to-run ./samples.to.run.tsv
```

This perhaps useful for such purposes of resuming an interrupting lenghty run. The sample names supplied with `samples.to.run.tsv` must match those described in `data/pp_table.txt` and have their corresponding input data named in a same pattern and located in the same folder structure as described in the "Our data" section.

If all of Stage One tools are run as a single command, regardless of their order in the command line, PyClone will be run first, followed by Ccube, followed by PhyloWGS. The results of these commands are stored in `Pipeline/results/pyclone`, `Pipeline/results/ccube` and `Pipeline/results/phylowgs` respectively.

Stage Two algorithms are launched in a similar manner. If the user already has the output from one of the Stage One tools, it is possible to launch Stage Two as a seperate command:

```
python pipeline.py --run-lda --run-nmf --lda-nmf-input
/results/ccube/
```

Or any other source, perhaps not generated by the tools implemented in Stage One, assuming their output is following the common format:

```
python pipeline.py --run-lda --run-nmf --lda-nmf-input
/some/other/input
```

Important note:
We supplied the submission with previously run 500 ccube results and Stage Two can be started by executing the pipeline with the following line:

```
python pipeline.py --run-lda --run-nmf --lda-nmf-input
/data/Data/Phase_two/Subclonal_Structure/
```

Licence

The usage of the source code and input data is entirely subject to University of Glasgow and Dr Ke Yuan of the School of Science and Engineering.

Appendix B Runtime and accuracy analysis of Stage One tools

Briefly introduced the Chapter 5 of the main report, this section expands upon our evaluation techniques and our conclusions that we made.

To briefly remind the reader how we approached our pipeline evaluation, we considered the following:

1. Measuring the running times of each tool,
2. Measuring the accuracy in determining the current number of SSM clusters in the samples that were run with all three tools,
3. Measuring the mean squared error of mutation assignments to each SSM cluster.

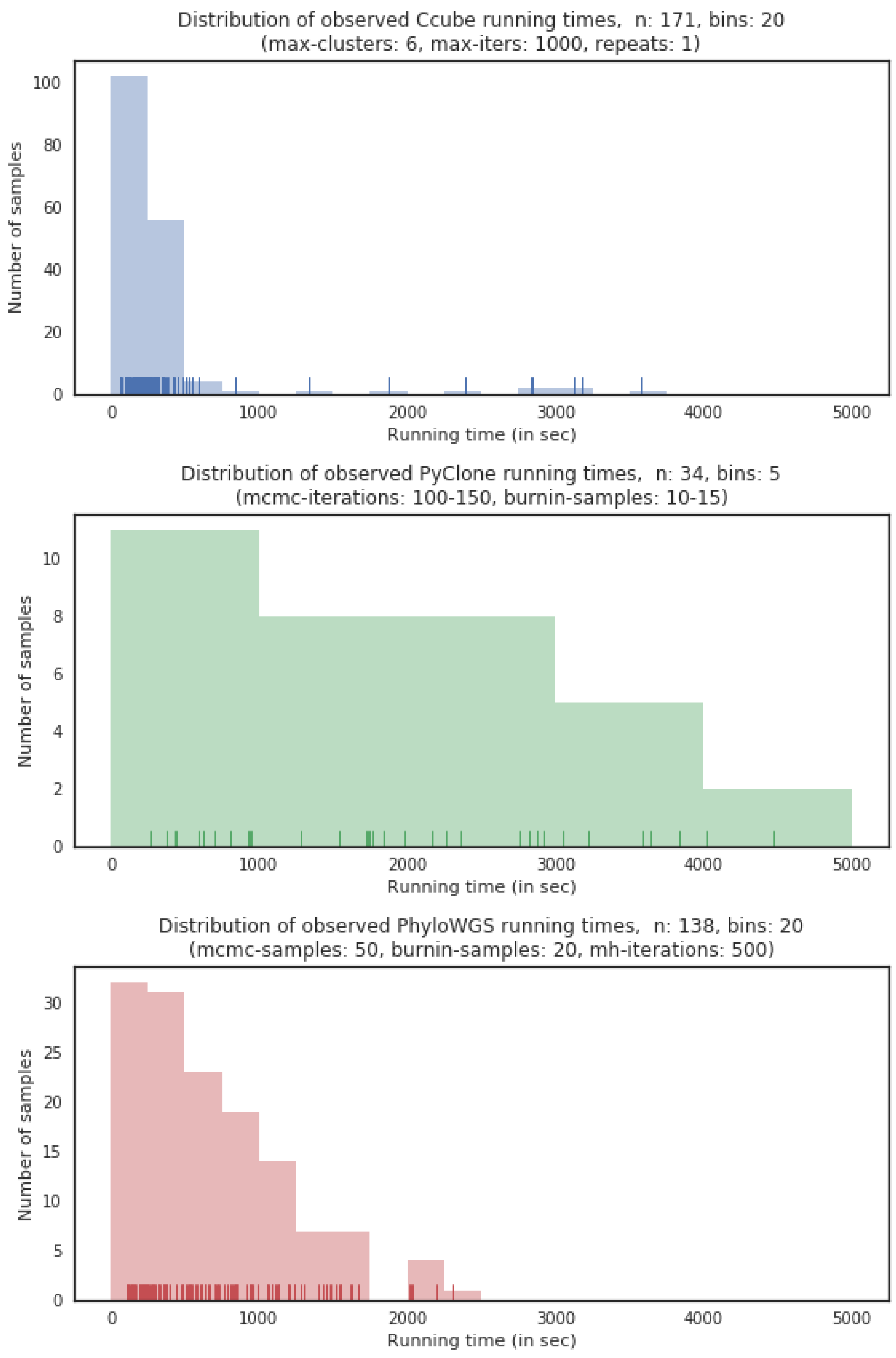
Notable challenges that we had to adapt to and accomplish a fair and objective comparisons were our hardware limitations and forcing the tools work under same artificial restrictions, since there were little-to-no overlap between the underlying inner algorithms behind each tool, and therefore no common parameters (with the exception of utilized parallel cores for PhyloWGS and Ccube) shared between these algorithms, something that would have helped us simulate the same running conditions dearly.

The three main questions about the evaluation strategy that needed to be solved in other to get around the limitation described in the paragraph above were:

1. Whether to 'anchor' the parameters for each tool in order to force the tools to run for a roughly the same time, and evaluating what level of accuracy can be achieved in a specific amount of time;
2. Whether to 'anchor' the parameters for each tool in order to try to achieve roughly the same level of accuracy, and evaluating how long did it for each tool to reach it;
3. How to balance one of the two above considerations with the varying degree of assumptions each tool makes about the data;

Ideally, we would want to perform these analyses with at least the default values of their parameters and (in a way) taking the second approach, firstly because those values are what makes the developers confident in the effectivity of their tools, secondly because it allows reproducibility of the experiments published in research where those tools were employed, eliminating some of possible confounding factors along the way, the most basic of all being accounting for the quality of our implementation of said tools.

However, since accounting for the same level of accuracy is hard, we chose accounting for the similar amount of runtime. Accounting for accuracy would also have meant that the evaluation strategies for this project were needed to be developed early on during the development of the pipeline and required the data post-processing steps to be implemented quickly as well, not to mention less room for error in our understanding of the processes behind discovering tumour philogeny. Failure to do would essentially require us to repeat the experiment once the errors would be eliminated.



<u>Figure B1</u>: Runtime analysis of all three Stage One tools, run on Intel Core i7 2nd Gen 2670QM (2.20 GHz) machine

For these reasons we decided to make a decision to focus on runtime-based evaluation. This didn't come without its own problems, as very early on we discovered an interesting problem with this approach: one of the three tools, Ccube, was orders of magnitude faster than the remaining two, PhyloWGS and PyClone. Trying to match the running times of Ccube was a fruitless effort as the tools reported unusable results with low-enough running parameters; conversely, trying to up-scale Ccube runtime would mean increasing our already significant level of downtime by a third.

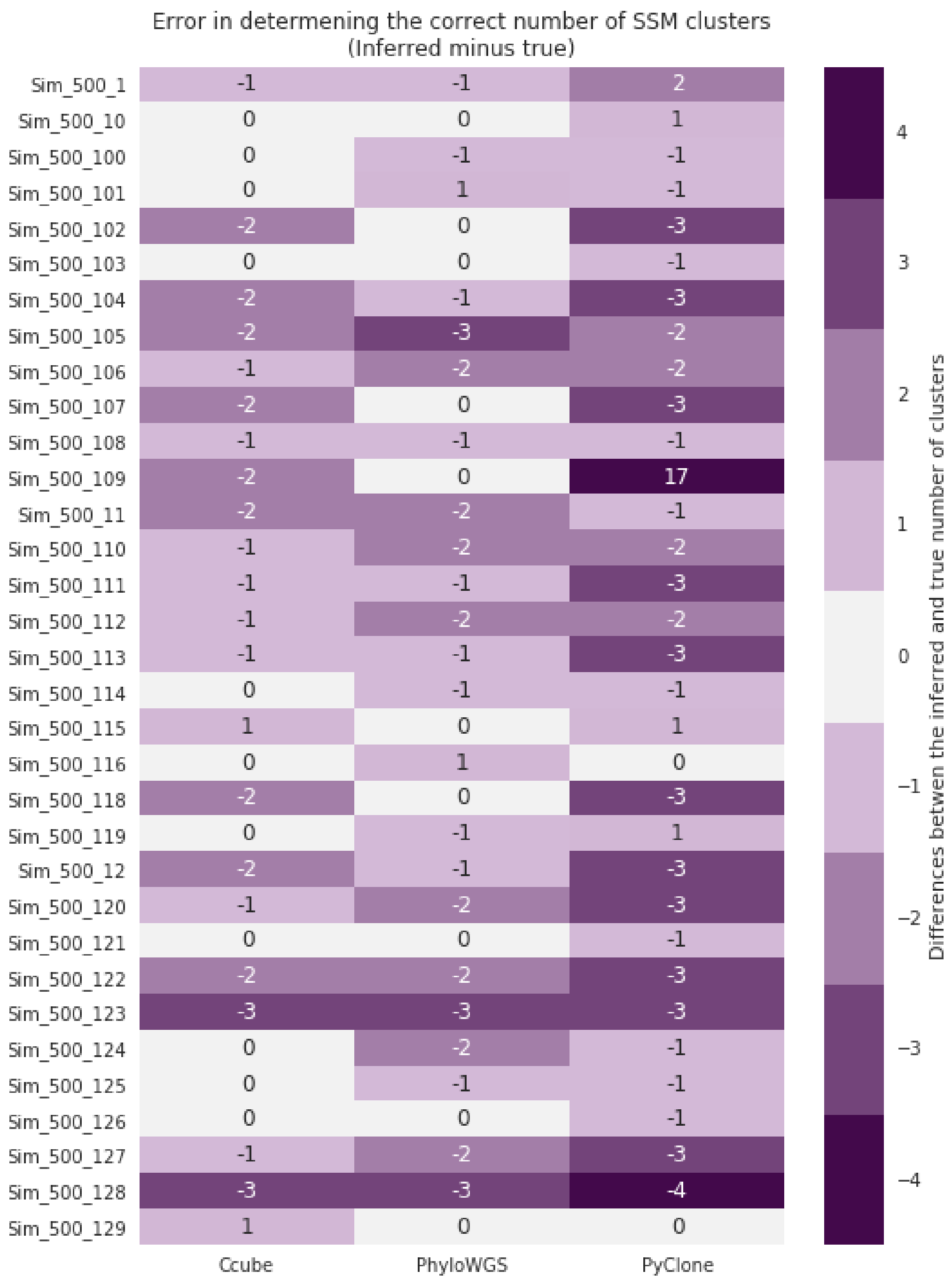


Figure B2: Heatmap of clustering analysis evaluation results. The values on the heatmap are calculated as a predicted number of clusters minus the true number of clusters; hence, values of zero and close are desired.

Instead, we chose to continue with our experiment as is, taking on a position that a stark difference between runtime results in one of the tools should be viewed a significant feature of the said tool, rather than a weakness in our evaluation strategy.

In regard to the assumptions about the data consideration, we heavily consulted with the project supervisor who has a substantial level of expertise in the field and who helped us define sensible inner parameter values for the data assumptions required by the tools to hold.

For running time profiling, we ran 171 samples with Ccube, 34 samples with PyClone and 138 samples with PhyloWGS on an Intel Core i7 2nd Gen 2670QM (2.20 GHz)-powered machine. The results of this experiment, as well as the parameter values we used, are summarized in **Figure B1**. The reason for two pairs of parameter values that we used during the evaluation of PyClone were our dissatisfaction with the qualitative results and may have yielded partial and unfavourable conclusions about the tool, so we slightly increased the parameters midway through our experiment.

It is clearly visible in the **Figure B1** just how much faster Ccube is in comparison with the other two tools, with perhaps only as much as 10% of the samples taking longer than 500 seconds to be run.

While PhyWGS and PyClone report a more similar right-tailed shape of their runtime distribution, it is important to acknowledge the fact that all of the 34 samples that PyClone was tested with were tested with other tools as well. With only 34 samples run, PyClone already reports occurrences of runtimes inside the 2500-3000 seconds bin, from which we can conclude that, at least in our pipeline, PyClone was the slowest of the three tools.

Next, we were interested to see how well the tools inferred the true number of simple somatic mutation clusters. **Figure B2** illustrates our findings. Looking at the figure, our initial impression tells us that all of the tested tools underperform in this task, predicting more clusters than there actually is. We speculate it's due to the low parameter values we tested the tools with which results in an inability of the tools to correctly merge the excess clusters. Most notable example here is Sample_500_109, which when run with PyClone, reported 17 excess clusters.

Unsure whether Ccube or PhyloWGS performed better in this task, we plotted a box-and-whiskers plot, available here as **Figure B3**, to check the quartiles of their result distribution. We removed the extreme outlier of Sample_500_109 in this plot to get a less distorted plot of PyClone. As visible in the plot, Ccube's and PhiloWGS's result distribution quartiles are matched and both of their medians just 1 cluster away from zero, in contrast to PyClone's corresponding quartiles, which show much more variation of its results.

It is important to note that findings from **Figure B2** and **Figure B3** are not very useful on their own. Theoretically, the measured differences between predicted SSMS's cancer cell fractions and true CCFs could be smaller for subclonal compositions with more clusters than subclonal compositions with the correct number of clusters but wrong mutation assignments. In such a case, the most that these plots could tell us is the fact that some tools have smaller threshold values for when to merge the clusters.

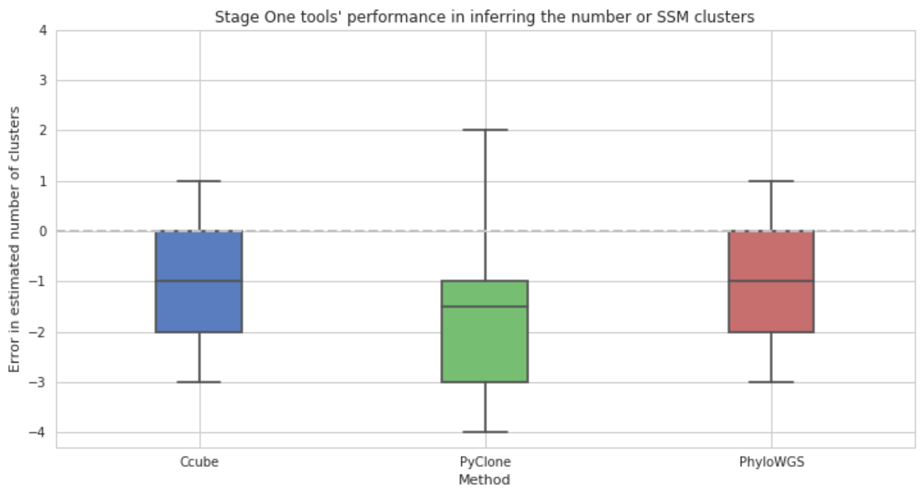


Figure B3: Box-and-whiskers plot to supplement the findings of the former heatmap.

In order to have a better sense of the tools' accuracy, differences between individual mutation cancer cell frequencies needed to be measured. In the last part of this analysis, we calculate the mean squared error between predicted and actual cancer cell frequencies for each of the 34 samples that all three tools were ran with.

Values close to zero would indicate correct inference made by the tool about the tumour CCFs and, by definition, (since the CCF ranges between zero and one) values that are closer to 1 would indicate completely wrong inference about the tumour phylogenetic tree.

As visible in **Figure B4**, Ccube achieves the smallest mean square error in almost every sample, wheres PhyloWGS seems to be the least accurate by a small margin, at least from inspecting the data visually. PyClone, on the other hand, has a distinction of achieving the highest margin of error.

Also, take note of Sample_500_109 in this plot, where PyClone reports a commendable result, despite having inferred that the sample had 17 more SSM clusters than the true value; and Sample_500_129, where all tools reported an error of zero (due to the tumour having most basic structure).

In this supplementary appendix of our report, we performed comprehensive runtime and accuracy analysis for the three tools of Stage One of our data pipeline. We discussed important considerations and design decisions we had to face for an effective analysis of the results. While we note that the tools were too different to be examined under the identical conditions, we have found that Ccube performed the best on both fronts virtually exclusively. Lastly, have also found an example to demonstrate a situation where the wild inaccuracy in determining the number of SSM clusters does not make the inference about the mutation cancer cell fractions automatically worse.

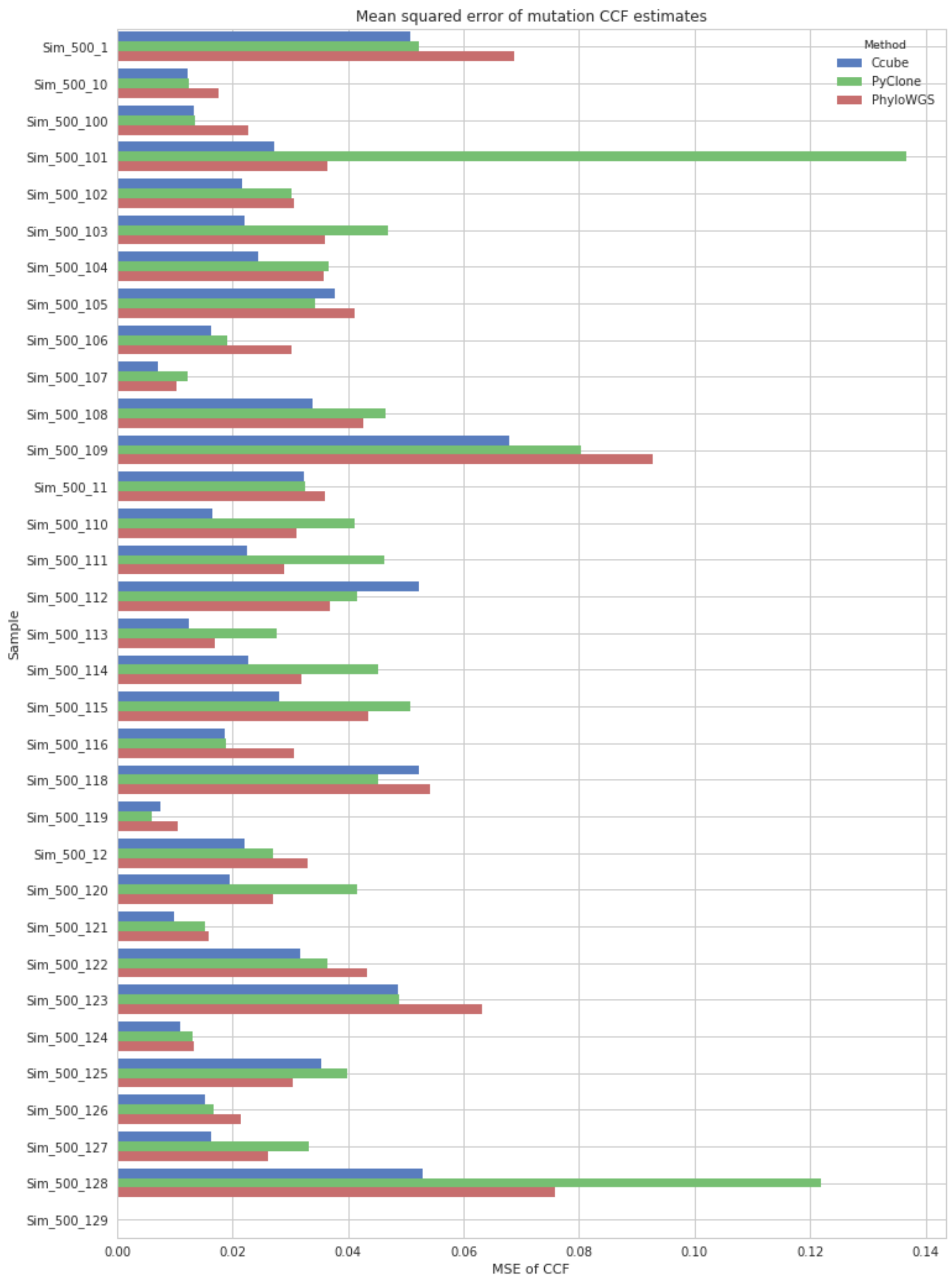


Figure B4: Mean squared error between the predicted and actual mutation assignments.