Department of Computer Science

Integrating simulations, data management and visualisation of a highly complex model (Whole Cell model).

Zhaozhen Xu

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering

---

September 2017 | CSMSC-17

Declaration:

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Zhaozhen Xu, September 2017

Executive Summary

The whole cell computational model simulates the life cycle of the cell. One of the applications of the model is to use it to find out the minimal genomes for cell life cycle by simulating the behaviour of the cell after genetic knockouts. With this computational model, it can predict the result of genetic knockouts which is hard to test in the laboratory. During the simulation, a huge amount of data will occur. To automatically identify the result of the genetic knockouts, data classification and analysis are essential to this project.

Data processing was one of the main parts in this project. In this project, it was applied in the biology to describe the mutants of the cell after the process of genetic knockouts. The data of the modified cells were compared with the wild type simulation in order to find out the influence of genetic knockouts. The mutants of the cells were categorised by the machine learning skills. K-Means, hierarchical clustering, DBSCAN, and spectral clustering were applied in this project. The result of the clusters was analysed and compared in search of the most suitable clustering models for the dataset. After that, several improvements were made to enhance the accuracy of the model.

The objective of this project is to have an idea of computational whole cell model and learning state-of-the-art data processing and machine learning skills. After that, a model which contains the data processing function and clustering which could categorise the mutant of the cell was designed and implemented. After all, the outcome was analysed to acquire the further result. This project is mainly about software development which accounts for 80%. And it also contains part of the investigatory and theoretical model which consists 10% for each part.

- The research carried out and reported in this thesis contains the comparison of 4 clustering methods and the analysis of the performance of the system based on the improvements of model
- The methods I used to reduce the dimension of the data which produced by the whole cell model were introduced, see page 22-25
- K-Means, hierarchical clustering, DBSCAN, and spectral clustering were implement using scikit-learn library to compare the performance of the clustering algorithms, see page 26-32
- I created a model which use data processing skills and agglomerative clustering to categorise the mutant of the cell, see page 33-45

Acknowledgements

I would like to express my gratitude to my supervisors Dr. Lucia Marucci and Prof. Claire Grierson who gave me the opportunity to research on this project. And They provided lots of advice for me during the planning and development of this research.

I would also like to acknowledge Dr. Oliver Ray. He provided a lot of technical supports which were really useful during the accomplishment of the project.

In addition, I would like to appreciate Oli Chalkley and Joshua Rees who also helped me in doing a lot of research and I came to learn so many essential knowledge on whole cell model and data classification. Oli also provided the dataset for this project. And Joshua provided the labels of the dataset. With their help, I can start my project much easier.

Finally, I wish to thank May Zhou who provided very valuable discussions and suggestions on the system design throughout my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

The whole cell model (Karr et al., 2012) is a computational model which can simulate the cell cycle of the Mycoplasma genitalium. The whole cell model can predict the behaviour of the cell after the genetic knockouts. In order to identify the essential gene for the cell, the mutant of the genetic knockouts should be determined. Compared the time series between simulations can figure out various kinds of mutants which mean if the genetic knockout has an influence on the cell life cycle or not. With this computational model, it can simulate the result of genetic knockouts which is hard to test in the laboratory.

However, there are some substantial difficulties while working with whole cell models. While using the Whole cell model (Karr et al., 2012) to simulate the cell life cycle, a huge amount of data has been produced and needed to be processed and classified. There are over 60,000 time series in one simulation. It?s difficult to compare all of them manually.

The aim of this project is managing data and classifying the distinction between the simulation automatically. The goal of the classification is to use the cell's characteristics to determine the different cell divisions with the wild type simulations and genetic knockout simulations. Although the behaviour of the simulations may look similar by eyes, the cell still working as usual without some genes remains uncertain. Finding the variance between the cell with diverse gene can contribute to identifying the functions of the gene.

As it is shown in figure 1.1, dynamics significantly various from wild type (WT) are highlighted in red. However, the protein of the WT and the cytokinesis seems like similar but there the small difference might lead to the change of the behaviour of the cell. On the other hand, the behaviour of the wide type simulations also varies for each other. The RNA of metabolic seems like having a bias from WT simulation, but it might be similar to other wild type simulations. And it could be recognised as normal. With a variety of data during the simulations, a classifier is needed to classify the variance among the cell automatically. Thus, a high accuracy system is required to automatically classified the cells. This can reduce the time on dealing with

the genetic knockouts and identify the minimal genomes.

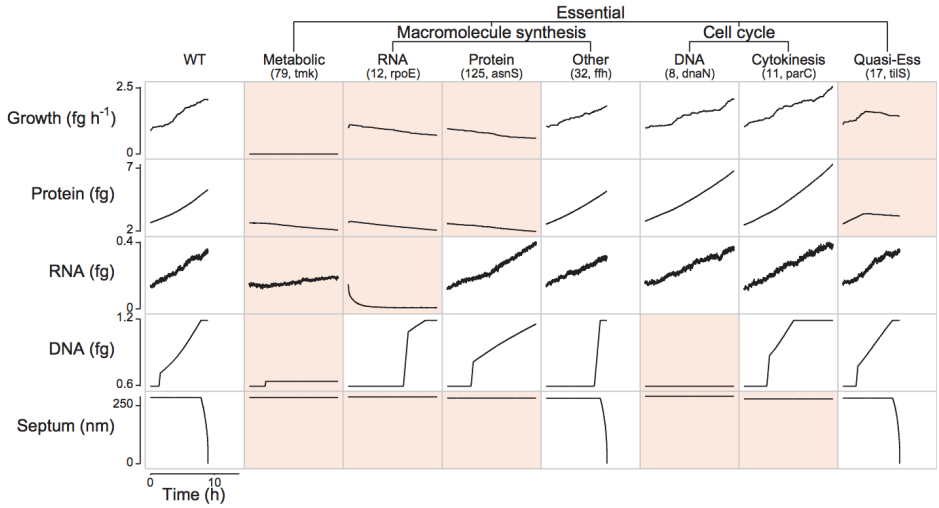


FIGURE 1.1: Model Identifies Common Molecular Pathologies Underlying Single-Gene Disruption Phenotypes (Karr et al., 2012).

With the implement of the classifiers, the mutants of the cells were separated. Further analysis and comparison between different classifying methods and diversified feature using to categorise the data were delivered in this project to find the most efficient way to identified and comprehend the mutant of genetic knockouts which changed the status and behaviour of the cell.

1.2 Deliverables and related techniques

This project aimed to create a data pre-processing method and a clustering model which could manage the data of whole cell computational model with high-efficiency and distinguish the mutants of the cell. After all, an analysis based on the result of the models was presented to evaluate the performance of the data classification. The deliverables are listed as follow.

- Literature survey of whole cell model, genetic knockouts, and mutant of the cell
- Literature survey of state-of-art data processing techniques and machine learning techniques related to the aim
- Programme which can pre-process the data for clustering
- Clustering model to classify the mutant of the cell simulated by the whole cell computational model

- A performance to evaluate the outcome of the models

To recognise the state of the cells, the knowledge on cell division and computational cell model was required. The knowledge of cell division was beneficial in data processing of the simulation data. The simulation of the cell was realised by using the whole cell computational model. This will be introduced in the following chapter. The wild type simulation was compared with the generic knockouts simulation in this project.

To classify the mutant of the simulations automatically, data processing and machine learning skills were applied. Machine learning is widely used in many industry and business to analyse the data these days. Analysis of datasets can find the internal connection in business trends, prevent diseases, combat crime and so on (McCune, 1998). In this case, data processing was the process to acquire the simulation data, transform it into an understandable structure and find the typical features of the distinct time series inside the simulation. The state-of-the-art data processing and machine learning skills such as feature scaling, clustering algorithms, principal component analysis etc. were introduced and applied in this project.

Machine learning is one of the important technologies which widely used in automatically data classification. It is difficult to classify the data with high accuracy. Different machine learning models have their own strength and implementation. The accuracy of classification depended on the learning and classified methods and the features which selected to classify the data. There is a "no free lunch" theorem (Wolpert and Macready, 1997) in applying the machine learning model. As a result, there are no optimum in different methods, only the one most suitable for this project. A design of the clustering models was made after analysis among several clustering algorithms.

The simulation data of this project which used to be classified was stored in the SQLite database and pandas dataframe which could be easily accessed and managed. For each simulation, it took around 30 hours. Thus, in this project, the simulations of various kind of cells had already done and stored in the database in advance. To accomplish this project, the knowledge of whole cell computational model and database related techniques were required.

The dataset which used in this project contained wild type simulations and single gene knockouts simulations. The size of the dataset was small, and the gene knockouts were uncompleted. Due to the limitation of the dataset, the first run of the single gene knockouts simulations were input into the unsupervised machine learning methods which used to categorise the mutants of the cell in this project. At the meanwhile, the accuracy of the model had been improved. After the model was with the high accuracy, it was used to predict the mutants of the second run of the single gene knockouts simulations.

In this project, after evaluating the features used to classify the data, several clustering methods were used which could automatically group the simulations of the cell with the same mutant. The accuracy and efficiency of the classifier were enhanced by applying different functions and using various features to manage and analyse the data. To improve the result of classification, the model and the feature will be modified during the implementation. After testing, the clustering model could be used to realise other function and further analysis in the whole cell model.

1.3 Added value

Nowadays, the minimal genomes group are still uncertain, because it is hard to knock out the gene and test if they are essential in the laboratory. Therefore, the research on the computational whole cell model will benefit the biology research. Classifying the mutants can improve the efficiency to find out the minimal genomes group.

The computational whole cell model is a new model which produced in 2012 (Karr et al., 2012), no one had made a classification for different mutants of the simulations. With this classifier, it is convenient to realise the difference between the simulations and enhance the research on minimal genomes. Finding the minimal genomes will make a big progress on the systems and synthetic biology communities.

1.4 Outline of the thesis

The related techniques of the project were explained in chapter 2. It included the research of the whole cell model, genetic knockouts, mutants of the cell, data processing techniques and clustering models. The theory of these techniques and how they work in this project were stated in this part.

And in chapter 3, the methodology of this project was explained in this section which including the methods applied to pre-process the data, how to use the clustering algorithms and the approach to measure the performance of the clusters.

For chapter 4, it analysed and compared the results of the all the clustering algorithms. After that, three steps of improvement were introduced in this chapter. The results of feature scaling, feature adjustment, and feature weighted were analysed based on the agglomerative clustering.

After building a model with high accuracy, the test of the model which meant the prediction of the data in this project was described and investigated in chapter 5.

And the evaluation of whole models was presented in chapter 6. In this

chapter, the advantage and limitation of the system were evaluated. The limitation was analysed on the basis of accuracy calculating, dataset, and parameter and scale modulation.

The conclusion and future work for this project were shown in chapter 7. It includes the analysis of the extent to which the project had obtained the objective. The possible further work was contained using the genetic algorithm to improve the performance of the DBSCAN and feature scaling of the clustering model.

Chapter 2

Literature review

2.1 Whole cell model

The whole cell model is a computational model which can simulate the cell cycle of the Mycoplasma genitalium which is a human urogenital parasite with 525 genes (Fraser et al., 1995a), including all its molecular components and interactions. Whole cell model was produced in 2012 for the first time (Karr et al., 2012).

2.1.1 Mycoplasma genitalium

Mycoplasma genitalium (Razin and Hayflick, 2010) has the smallest known genome of the self-replicating organism. It is a pathogenic bacterium which first mentioned in 1981 (Tully et al., 1981) and used as a basis and the most popular initiation to find the minimal possible genomes for life (López-Madrigal et al., 2011). The genome of Mycoplasma genitalium has one circular piece of DNA which consists of 580,070 base pairs. The DNA contains 525 genes. 482 genes are protein coding genes, and 43 genes are RNA coding genes (Fraser et al., 1995b). The structure of the Mycoplasma genitalium is shown in the figure 2.1.

2.1.2 Cell life cycle

The life cycle of the cell is known as cell division cycle. Cell division cycle takes place in a cell which leading to its division into two daughter cells. During the process of the division, the DNA will be duplicated (Wang and Levin, 2009).

The stages of the cell cycle are divided into three phases: interphase, the mitotic phase, and cytokinesis. During the interphase process, the cell grows and duplicates its DNA. This lead to the increase of the weight of the cell. For the mitotic phase, with the nutrients from the interphase, the chromosomes separate. At the final stage, the chromosomes and cytoplasm separate into two new daughter cells (Blackstone, 2001). The cell life cycle will influence the simulation data of the cell which is useful in predict the condition of the cell. The performance of Mycoplasma genitalium during this life cycle will be predicted in the whole cell model.

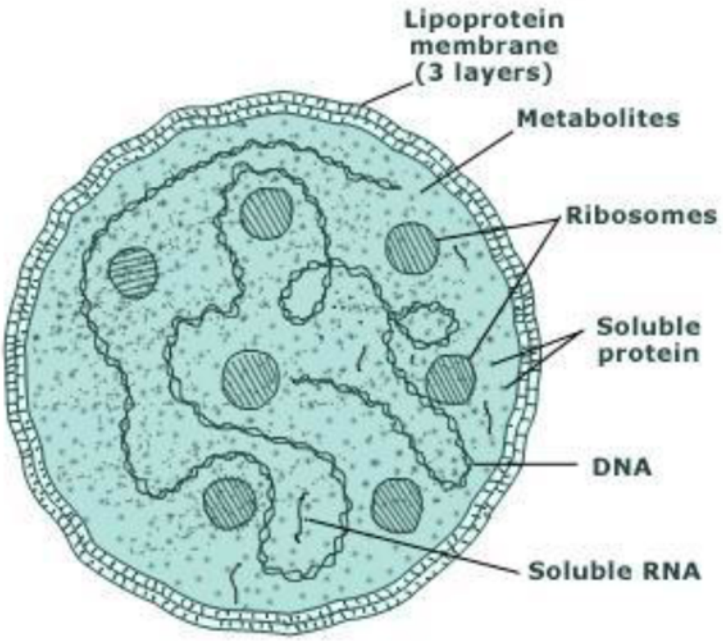


FIGURE 2.1: The structure of Mycoplasma genitalium (*Cyanobacteria Mycoplasmas*).

2.1.3 Design of whole cell model

The whole cell model contains 16 cellular states and 28 cellular processes. Each process is mapped with a sub-model. The algorithm of the whole cell model is illustrated in the figure 2.2.

The whole-cell model (Karr et al., 2012) is composed of 28 sub-models which span six major areas of cellular physiology. The sub-models were modelled using different mathematics and trained using different experimental data. Computationally, the inputs and outputs of each sub-model are the copy numbers of metabolites and macromolecules; the configurations of RNA, protein and DNA polymers; and the catalytic capacity and configurations of the enzymes which participate in each sub-model. The process is repeated until the cell divides or the maximum time is reached. The maximum time in this project is 50,000 seconds.

The whole cell model attempts to describe the life cycle of the single cell and accounts for the function of every gene product. In addition, it also can predict the cellular behaviour. It not only describes the molecular properties, but also cellular properties like growth, mass etc. While using the whole cell model to simulate a cell, the wild type simulation refers to the simulation of the natural cells which contains all the gene. Each of simulation contains more than 60,000 time series.

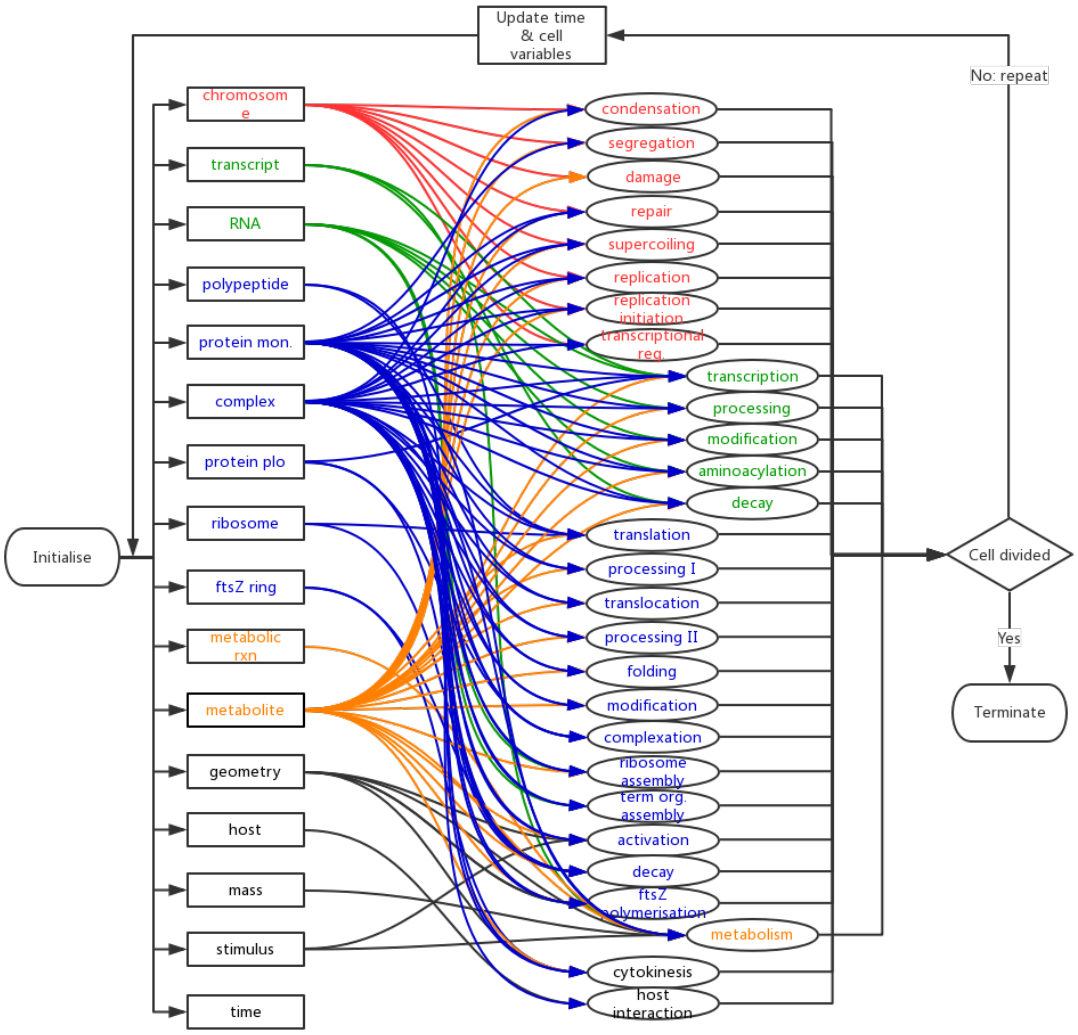


FIGURE 2.2: The algorithm of Mycoplasma genitalium Whole Cell Model (Karr et al., 2012). It integrates 16 cellular states which are the nodes in the left column and 28 sub-models which are the nodes in the right of column of the cellular division. The nodes and edges are separated into different categories where red represents DNA, green represents RNA, blue represents protein, and black represents others.

2.2 Genetic knockouts

Genetic knockouts remove one or several genes in the whole cell model (Sung et al., 2016). This may change properties, such as cell division. Figure 2.3 below illustrates the comparisons between two simulations.

The comparison of two simulations is obviously different in DNA mass after genetic knockouts. This shows the cell might be failed to division because the mass of DNA remains unchanged which means the DNA does not duplicate itself. Even though the growth rate, RNA mass, and protein mass seem to have the same trend as the wild type simulation, it still shows the difference that they are more than wild type at the end of the simulation.

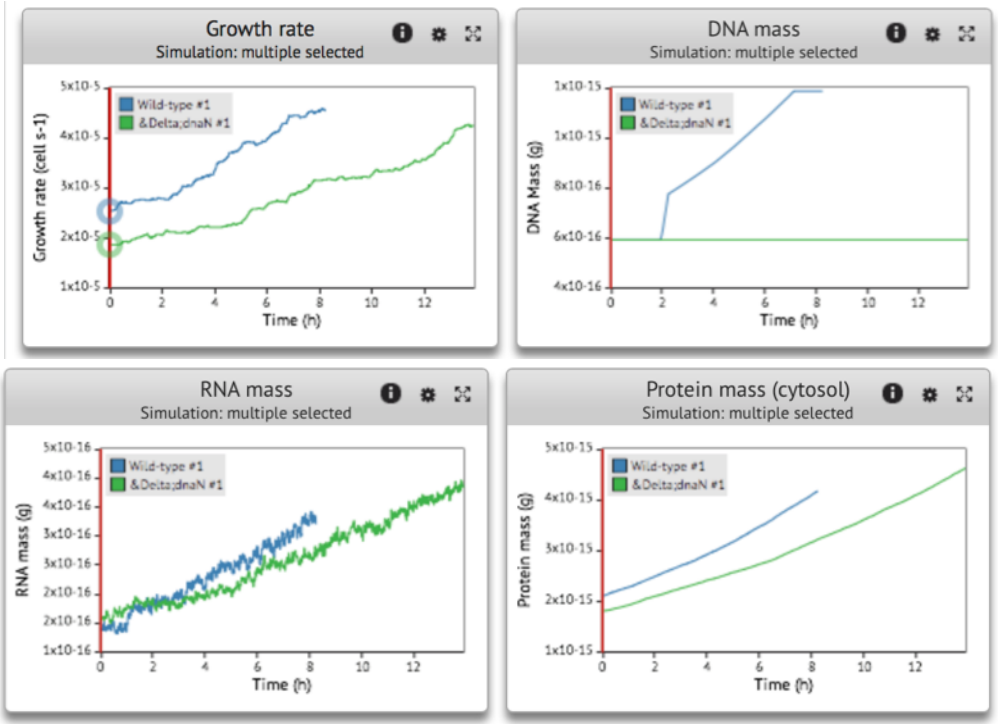


FIGURE 2.3: The comparisons of growth rate, DNA mass, RNA mass, and protein mass between wild type and genetic knockouts (Lee, Karr, and Covert, 2013). The green line represents the genetic knockout simulation. And the blue one illustrates the wild type simulation.

Not every genetic knockout will influence on the cell cycle. Some genetic knockouts just behave as wild type simulations.

As Karr (Karr et al., 2012) mentioned, a gene is essential if the cell will not divide without this gene. If the cell did not divide, a mutant occurs. Comparing the cells between wild type and the single gene knocked out, some of the genes can help to find out the essential gene for the cell. The minimal genomes can be found by using the technique of genetic knockouts.

In this project, there are 359 types of single gene knockouts which knockout are labelled.

2.3 Types of mutant

There are 8 types of mutant after single genetic knockout: DNA, septum, protein, metabolic, non-essential, slow growing, protein and DNA, RNA. Non-essential means the cell behaviour like wild type simulation, the gene which knocked out is recognised as non-essential. The definitions of various mutants are listed in the following table 2.1. The mutants can be predicted by data from the specific time series which produced by the whole cell model.

Types of Mutant	Description
DNA	DNA not reproducing. Chromosome stays at value 1.
Septum	Cell does not divide.
Protein	Protein is not being produced.
Metabolic	Cell is not growing at all. Dead on arrival.
Non-essential	Gene knock out does not affect the behaviour of the cell. Gene is not essential for life.
Slow growing	Shows abnormally slow growth rate.
Protein and DNA	Protein is not being produced and DNA is not reproducing.
RNA	RNA is not being produced.

TABLE 2.1: The description of different types of mutant.

The mutant for the single gene knockout is labelled manually by Joshua Rees who is the PhD student researching in the minimal genomes based on the behaviour of the whole cell simulation and biology. The results of simulations are various in the different run of the model. The labels in the project are based on the first run of 359 single gene knockouts. The count of the labels is illustrated in figure 2.4.

According to the figure, it is obvious that the dataset was imbalanced. There was a large difference between the number of the samples for each mutant. The non-essential, metabolic and protein mutant was the main part of the dataset.

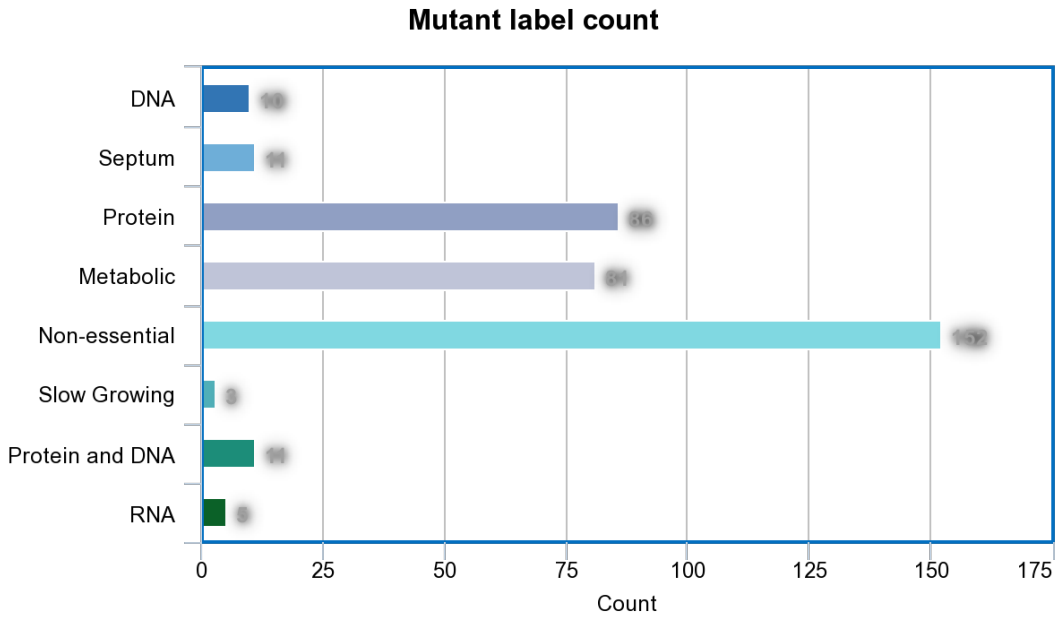


FIGURE 2.4: The count of mutant labels. Most of the single gene knockouts are labelled as non-essential because removing them one at a time does not significantly change the behaviour of the whole cell model. The slow growing is the least among the mutants which only 3 of the knockouts are recognised.

2.4 Machine learning

Data processing is the process to collect and manipulate the data in order to produce the meaningful information (French, 1996). Data processing involves several processes. In this project, data analysis and data classification are mainly used. Data analysis contains cleansing, collection, interpretation, organisation, modelling, and presentation of the data. Data classification is a process to decide which category the data belongs to.

To categorise the data, the state-of-the-art machine learning technology was applied. Applying the machine learning skills enables the data to be classified automatically. Accurate classifications can produce accurate predictions (Ştefan, 2012). As a result, using the appropriate classifying method is very important. There are two main types of machine learning: supervised and unsupervised learning.

2.4.1 Supervised learning

Supervised learning is a learning method that a teacher involves which can be recognised as labelled training data (Mohri, Rostamizadeh, and Talwalkar, 2012). A supervised learning algorithm investigates the training data and builds a function which can be used to categorize new examples. There are several processes in supervised learning.

First, decide the type of training data. The users need to determine what type of data is used as the training set. Then the training set needs to be gathered. Before inputing the data into the model, the input feature representation of the learned function need to be determined. The accuracy of the learned function strongly depends on how the input object is represented. After that, the structure of learned function and learning algorithm are chosen, such as support vector machine, K nearest neighbour etc. Finally, run the algorithm and evaluate the performance of the model.

2.4.2 Unsupervised learning

If training machine learning task only with a set of inputs and without any label, it is called unsupervised learning, which will be able to find the structure and relationships between inputs data (Ghahramani, 2004). Clustering is the most important unsupervised learning technique, which can create different clusters of input data.

Compared to supervised learning, the clustering algorithms do not need the label for the data and a training set (Donalek, 2011). In this project, the dataset was imbalanced. For example, the slow growing mutant only had 3 samples, which was hard to separate into the training set, validation set,

and testing set that was essential for supervised learning. As a result, clustering algorithms were used in this project.

Clustering has a long history, the importance and the crossover of other research have been approved. The goal of clustering is to determine a finite set of clusters of data according to the similarities of the objects. Clustering model is one of the important research contents of data mining, pattern recognition and so on. In the identification of the internal structure of data Has a very important role.

In machine learning, it is applied to image segmentation and machine vision. Another major application of clustering is data mining. In addition, clustering is also applied to statistic. It is worth mentioning that clustering analysis also plays an important role in biology, psychology, geography Learning and marketing research (Carson et al., 2002; Cadez, Smyth, and Mannila, 2001; Jain, Murty, and Flynn, 1999).

Four of the clustering algorithms which were used in this project will be introduced.

K-Means

K-Means is one of the most widely used clustering algorithms. The algorithm takes K as a parameter. N objects are divided into K clusters. The objects inside the cluster have a high similarity. And the similarity between the clusters is low. First, the algorithm randomly selects K objects. And each object is set to represent an initial average or center of a cluster. Calculating the distance between the object and all the centres of the clusters in order to assign it to the nearest. Then re-calculate the average of each cluster. Continues to repeat this process until the criterion function is converged. This means that the stable clusters have been built and further relocation of the objects is impossible (Han, Pei, and Kamber, 2011). The criteria function is as follows.

The K-means algorithm is described as follows (MacQueen, 1967):

1. Randomly select K objects record as the initial cluster centres
2. Calculate the distance between each object and K clustering centres. And use the nearest cluster as the class which the object belongs
3. Calculate the centroid of each cluster and the distance to each object objects. Re-classify the objects according to the minimum distance. Repeat this step until the cluster does not change significantly

The advantage (Huang, 1997) of K-Means is that it can categorise large datasets with high efficiency. The computational complexity is O(tKmn), where t is the number of iterations, K is the number of clusters, m is the number of features in every instance, n is the number of objects to be classified. When

clustering the large datasets, K-Means is much faster than the hierarchical clustering.

The disadvantage (Huang, 1997) of K-Means is that it usually terminates when a local optimal value is obtained. And it is only suitable for numerical data clustering. It only works on clustering the spherical clusters.

Hierarchical clustering

Hierarchical clustering (De Sa, 2012; Fred and Leitão, 2000) is a tree based clustering. The data will be split or aggregated to form a solution for the clustering problem through a hierarchical architecture. This section only focuses on the agglomerative clustering which is a bottom-up hierarchical clustering model. The computational complexity of the aggregative clustering algorithm is $O(n^2)$ which is suitable for the classification of the small datasets.

There are four main steps in the agglomerative clustering.

1. Each object is identified as a class which gives N clusters, and each cluster contains only one object. The distance between clusters is the distance between the objects they contain.
2. Find the two of clusters which are the closest to each other and merge them into one cluster. Thus, the total number of classes is one less.
3. Recalculate the distance between the new cluster and all the previous clusters.
4. Repeat steps 2 and 3 until the last merge into the number of clusters set before.

There are three different methods used to calculate the linkage between the clusters in this project. Complete linkage recognises that the distance between clusters is equal to the maximum distance between two objects which can be found in two clusters. Average linkage considers that the distance between clusters is equal to the average distance between the objects in two clusters. Ward linkage is the distance that minimises the sum of squared differences within all the clusters.

The advantage of hierarchical clustering is that the multi-level clustering structure can be obtained by setting different parameter values. It is suitable for clustering of all kinds of shapes for the clusters. And the order of the input of the samples is not sensitive.

The disadvantage of hierarchical clustering is that the time complexity of the algorithm is high, and the result of hierarchical clustering depends on the choice of clustering points and splitting points. And the process of hierarchical clustering is irreversibility. Since the object is merged or split, the next cluster will continue to merge or split on the basis of the previous cluster. That is, once the clustering result is formed, it is impossible to re-merge

to optimise the performance of clustering. Another drawback of hierarchical clustering is the terminating condition is inaccurate because it requires an aggregation or decomposition termination condition to be specified.

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) (Ester et al., 1996) is a typical density-based clustering algorithm, which assumes that categories can be determined by the compactness of distribution of the samples. Samples of the same category are closely linked with each other. This means any sample in the category must found the same type of sample exists which is not far from them. By classifying closely connected samples into one class, a clustering category is obtained. By determining all the closely connected samples into different categories, the result of all clustering categories is obtained.

DBSCAN describe the compactness of the sample set based the neighbourhoods. The parameters (ϵ, MinPts) are used to describe the compactness of the distribution of the sample in the neighbourhood. Whereas ϵ describes the threshold of the neighbourhood distance for a sample. And MinPts describes the threshold of the number of samples in the neighbourhood for a sample to become a core point of the cluster. The algorithm works as follows (Ansari et al., 2013):

1. All samples are marked as unvisited at the beginning
2. Randomly visit an unvisited object X. If X has more than MinPts number of samples in its neighbourhood within the distance of ϵ, a new cluster N will be created for it. Otherwise, it is marked as a noise point (outlier)
3. Once a cluster is created, each point Y in this newly formed cluster will be visited iteratively. If point Y is unvisited, mark it as visited. And if point Y has MinPts number of samples in its neighbourhood, those samples are also added to the cluster N. If Y does not belongs to any cluster, it is added to the created cluster N.
4. Repeat steps 2 and 3 until all samples have been visited.

The advantage of DBSCAN it that it is suitable for categorising of arbitrary shape for the clusters. It can find the noise while clustering. And it is not sensitive to the abnormal points in the dataset. The number of clusters can be determined by the algorithm which means it does not need to be input by the users. In addition, the result of clustering is not biased. In contrast, the initial values of clustering algorithms such as K-Means have a great influence on the clustering results.

The drawback of DBSCAN is that If the density of the samples is not uniform or the distance between the clusters is very large, then DBSCAN clustering

is not suitable due to the poor quality of the clustering. When the number of samples is large, the time for converging time of clustering is long. Comparing to the traditional clustering algorithm like K-Means, the tuning of parameters is more complex. Different combinations of parameters have a great influence on the final clustering outcomes.

Compared with K-Means, DBSCAN can be applied to spherical clusters or non-spherical clusters. Compared with the K-Means, the biggest difference of DBSCAN is that the number of categories K does not need to be entered. At the same time, it can also find out the outlier of clusters.

Spectral clustering

Spectral clustering (Von Luxburg, 2007) is a widely used modern clustering method. Compared with K-Means, spectral clustering is more adaptable to the distribution of the data. And the performance of category is good. The computational cost of clustering is also much smaller and easy to realise. When dealing with the clustering problems, spectral clustering is one of the best algorithms that should be considered to use first.

Spectral clustering is an algorithm which evolves from graph theory (Fiedler, 1975) and widely used in clustering. The main idea of the model is to consider all the data as a point in space. These points can be connected by the edges. If two points are far away from each other, the weight of the edge is small. By contrast, the distance between two points is close the weight of the edge is large. The sub graphs are obtained by cutting the graph which consisted of all the data points. The aim of the cutting is to make the sum of the weight of the edges between each sub-graph as small as possible so that it can achieve the purpose of clustering. There are two ways that are widely used to cut the graph, which is Ratio cut and normalised cut.

There are 4 main steps in the spectral clustering:

1. Build a similarity graph, W is the weighted adjacency matrix of the similarity
2. Calculate unnormalised graph Laplacian matrix L. Whereas L = D - W, D is the degree matrix
3. Calculate K smallest eigenvectors of the Laplacian matrix
4. Arrange K eigenvectors to form an N * K matrix, where each row is regarded as a vector in k-dimensional space which can be used to clustering by the K-Means algorithm

Based on the main steps of the algorithm, the spectral clustering can consider as applying dimensional reduction methods before applying the data to the clustering model.

Spectral Clustering has many advantages compared to traditional clustering methods, such as K-Means. Spectral Clustering only requires a similarity matrix built among the data, rather than requiring the data to be a vector in an N dimensional Euclidean space like K-Means. Because of focusing on main contradiction and ignoring others, spectral clustering is stronger than other traditional clustering algorithms. It is not sensitive to the anomalous error of data which results in the better performance. In fact, in the comparison of various clustering algorithms, K-Means is normally used as a baseline. Another advantage of the spectral clustering is that the computational complexity of it is less than K-Means, especially when running on dataset with very high dimensions, such as plain text or image.

The main shortcoming of the clustering algorithm is that the performance of the clustering is on the basis of the similarity matrix. Different similarity matrix will lead to a completely different result of the clustering model. If the dimensions which applied to the clustering model after dimensional reduction are still very high, the speed and the performance of the spectral clustering will be bad because the reduction of dimensions is not enough. Furthermore, spectral clustering requires the number of clusters to be specified in advance. It only performs well for a small number of clusters. For a large number of clusters, spectral clustering is not advised.

2.5 Principal component analysis

For plotting purposes, 2 or 3 principal components are usually more sufficient and easier for the user to virtualise the high dimensional data. The principal components analysis was used to reduce the dimensions of the instances when plotting the diagrams.

Principal components analysis (PCA) was first formulated in statistics (Pearson, 1901). The goals of PCA are simplification and data reduction (Wold, Esbensen, and Geladi, 1987). After data reduction, the noise in the data is removed. It is useful when large amounts of data are approximated by a complex model structure. Almost all data matrix can be simplified using PCA. PCA has been called one of the most important results from applied linear algebra (Shlens, 2014). It is used widely as the first step to analyse the datasets with high dimensions.

PCA searches the orthogonal vector which can best represent the original data and creates a smaller variable set to replace the original one. This process combines the essential attribute and projects the raw data onto the smaller set. To achieve the low dimension data matrix, there are four steps (Richardson, 2009).

First of all, the original data matrix should be zero mean. The averages of every column are found respectively. And each value on this column minuses their average. After this process, the new matrix is created. Then use this

new matrix to find the covariance matrix. Use the linear model to find the Eigenvalues and eigenvectors of the covariance matrix. Every feature has an eigenvector. The eigenvector and eigenvalues are the one-to-one correspondence. Assumed there are m eigenvalues, then the eigenvectors of the first N eigenvalues will be reserved. These eigenvectors make up a new Eigen space. Finally, using this new Eigen space multiplied the zero-mean data to get the low dimensional matrix.

The value of the parameter N decides the performance of the data processing. It is important to find the most appropriate N so that the most of the noise can be reduced, and the valuable information can be kept as much as possible. The percentage of the variance is calculated to design the value of N. If N is too small, then we might be using a very bad approximation to the data. The equation is listed as follow.

$$percentage = \frac{\sum_{j=1}^{K} \lambda_j}{\sum_{j=1}^{N} \lambda_j} \tag{2.1}$$

Whereas λ is the eigenvalue. K is the first K component needed to be kept. The smallest value of N is picked to satisfy the percentage of the variance is between 90-98%.

2.6 Scikit-learn and Pandas library

This project was developed based on python. There was two libraries which mainly used to realise the machine learning and data processing functions.

Scikit-learn (Pedregosa et al., 2011) is an open source library for machine learning in Python. It contains simple and efficient tools for data mining and data analysis. It is accessible to everybody, and can be used in various contexts. The library is built on the basis of NumPy, Scipy and Matplotlib library. According to the function of the algorithm, the scikit-learn library can be divided into supervised learning including classification and regression and unsupervised learning which is clustering. The library also provides test datasets, data pre-processing methods, and test data selection, test algorithm, and parameters modulation.

While applying scikit-learn to your own data, preprocessing is one of the most important steps. The pre-processing includes feature selection, normalisation, and data formatting etc.

Python Data Analysis Library or pandas (McKinney, 2014) is a tool based on NumPy, originally developed by AQR Capital Management. It was created to deal with data analysis tasks. Pandas incorporates a large number of libraries and some standard data models. It provides the tools which needed to operate large datasets efficiently. Pandas provides a number of functions

and methods that enable users to process data quickly and easily. It is one of the important factors that make Python become a powerful and efficient data analysis environment.

The main data objects in pandas are series and dataFrame. Although they are not a common solution for every problem, they provide an easy-to-use basis for most applications.

Series is a one-dimensional array of similar objects, contains an array of data and a data tag that associated with the array, called the index. It likes an array, and the index can be used like an array. A dataframe represents a table that similar to the data structure of the spreadsheet. It contains a set of sorted list. Each of them can have different types of values such as numbers, strings and so on.

Dataframe has a row and column index which likes a two-dimensional array rather than lists, dictionaries, or other one-dimensional arrays. The dataframe was used to represent the raw simulation data in this project because the dataset had two dimensions which were time series and time steps.

Chapter 3

Methodology

The simulations of the cells contained 200 wild type cells and 359 types of one gene manipulated cells which were given in this project. There were two runs of the simulations of the one gene manipulated cells. The first run contained 353 simulations, and the second one contained 256 simulations. The simulations was crashed which lead to the lack of the data. The data is stored as pandas dataframe which is an open source library providing high-performance data structures and data analysis tools with simple operation for the Python (McKinney, 2011). The simulations recorded the information of every time steps during the cell division. In this research, the first run of the simulations of the single gene knockouts were used as the instances to apply to the machine learning model. The second run was used to test the performance of the model.

However, as each simulation had more than 60,000 time series, the size of the dataset was too large to apply to the clustering model. As a result, the dimension reduction of the dataset was essential. After minimising the dimension of the data, they were applied to the several clustering models to find out which model had the best performance. Finally, the prediction of single gene knockouts mutants was made base on the model with the highest accuracy.

In this chapter, the method used to pre-process the data, how the clustering algorithms were designed, the standard to evaluate the performance of the models, and the process of using the model to predict the mutant will be introduced. The methodologies are illustrated in figure 3.1.

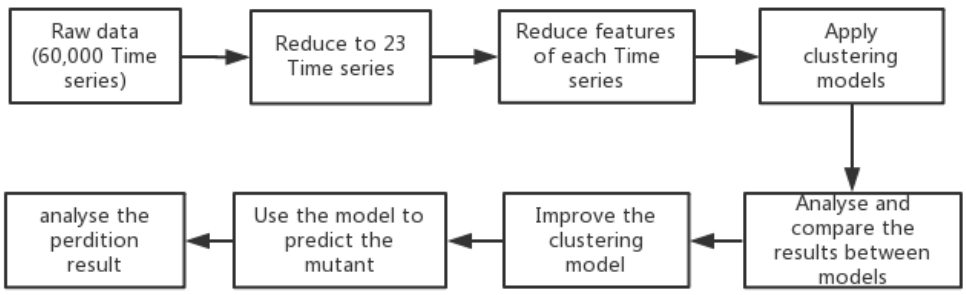


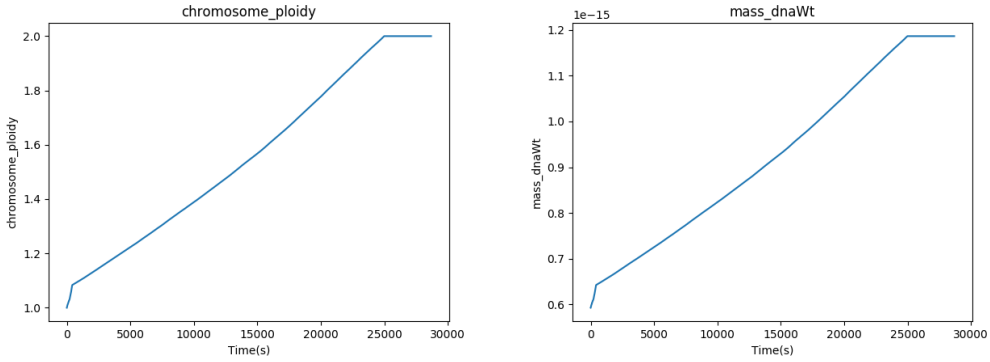
FIGURE 3.1: The main methods used in the project

3.1 Data preprocessing

The aim of the data pre-processing was to reduce the size of the dataset and removed the redundant data. Data cleansing can enhance the accuracy of the clustering model because it can reduce the noise of the data. The wild type simulation was used to visualise the raw data which helps to find the normal behaviour of the data. The data cleansing in this project had 2 main steps: reducing time series and determining feature.

3.1.1 Reducing time series

Two kinds of time series were recognised as redundant in this project. 1) Some of the time series are not related to categorising the mutant of the cell. 2) Some time series had a connection with others which meant only kept one of them can represent the status of the cell. For example, the time series which described the total mass of the cell was removed due to it could be separated to mass of cell, media, water weight, metabolite weight, DNA weight, RNA weight, and protein weight etc. With the wild type simulation, the number of time series can be easily cut down. As figure 3.2 shows, ploidy of chromosome was related to DNA weight.



(A) Ploidy of chromosome (B) DNA weight

FIGURE 3.2: The ploidy of chromosome and DNA weight in wild type simulation. Although they had different scale, the shape of the curve was the same. These were two different ways to describe the same change. Thus, either one of them could be cut.

In this process, the time series which were not related to translation, transcription, division, DNA, RNA, Protein, and cell growth were cut down. After this process, 23 time series remained which listed in the following table 3.1.

After reducing the time series of each simulation, the size of the data was significantly decreased from 60,000 to 23 time series.

Time series	Description	Behaviour of the data in the wild type simulation
mass_cell	The weight of cell. The weight increase during the duplication of the cell	Increasing tendency, nearly linear
mass_media	The weight of external of the cell	Decreasing tendency, nearly linear
mass_waterWt	the weight of water	Increasing tendency, nearly linear
mass_metaboliteWt	The weight of metabolite. The metabolite consumed during the duplication	Decreasing tendency, nearly linear
mass_rnaWt	The weight of RNA	Rise with fluctuation
mass_proteinWt	The weight of protein	Increasing tendency, nearly linear
chromosome_ploidy	The ploidy of the chromosome	Start from 1, and reach 2 near the end of the simulation
chromosome_segregated	The cell segregated or not which means the chromosome has successfully replicated	Reach 1 near the end of the simulation
geometry_pinchedDiameter	The diameter of pinched part of the cell	Remain unchanged until the cell start dividing and drop to zero at the end of the simulation
geometry_pinched	The cell pinched or not which means whether the cell is divided	Reach 1 at the end
ftszRing_numEdgesOneStraight	The number of one straight edges of ftsZ ring	Start from 0, reach a value during the simulation then start decreasing
ftszRing_numEdges TwoStraight	The number of two straight edges of ftsZ ring	Start from 0, reach a value during the simulation then start decreasing with fluctuation
ftszRing_numEdgesTwoBent	The number of two bent edges of ftsZ ring	Start from 0, reach a value during the simulation then start decreasing
ftszRing_numResidualBent	The number of residual bent edges of ftsZ ring	Start from 0, reach a value during the simulation then start decreasing
ftszRing_numEdges	The number of edges of ftsZ ring. It decreases while the duplication of the cell	Remain unchanged until the cell start dividing and drop step by step to 0 at the end of the simulation
ribosome_nActive	The number of active ribosome	Rise with fluctuation
ribosome_nNotExist	The number of not existed ribosome	Decreasing tendency at first, has a dramatically increase during duplication
ribosome_nStalled	The number of stalled ribosome	2 value occur: 0 and 1
rnaPolymerase_nActive	The number of active RNA polymerase	Fluctuating
rnaPolymerase_nSpecifiallyBound	The number of specifically bound RNA polymerase	4 value occur: 0, 1, 2, and 3
rnaPolymerase_nNonSpecificallyBound	The number of not specifically bound RNA polymerase	Rise with fluctuation
rnaPolymerase_nFree	The number of free RNA polymerase	The value is fluctuating between 0 to 15
metabolicReaction_growth	The growth rate of the cell	Increasing tendency

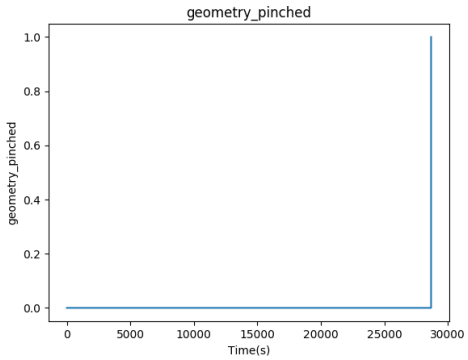
TABLE 3.1: The description of 23 remaining time series after reduction. The distribution of the data was inspected manually.

3.1.2 Feature selecting

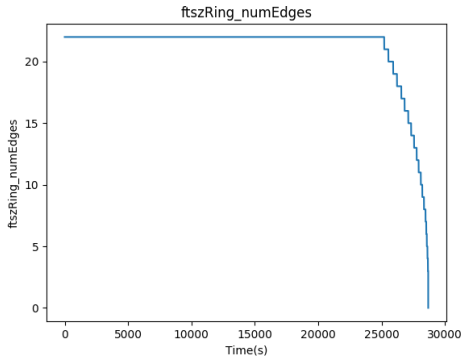
After decreasing the time series of the simulation, each simulation still had 26,000 to 50,000 time steps. One of the problems occurred during data cleansing was that the length of the simulations was different. This led to the different length of the instances which applied to the clustering model. To build all the instance for the same length, the time series were represented by several features and regardless of the time length in each time series.

According to the visualisation of the wild type simulation, the data of the time series could be categorised into 4 types. As it shows in figure 3.3, for the different type of data, various features were selected.

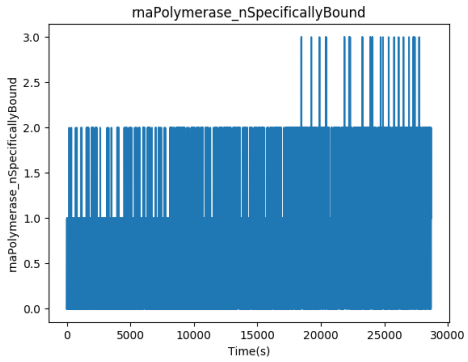
It can be observed from the figure 3.3 A that only the final value in this time series was valuable. Consequently, one feature was kept for this kind of time series. If the value was changed at the end of the simulation, then the feature was recorded as 1. Otherwise, it was set to -1 which made it differed



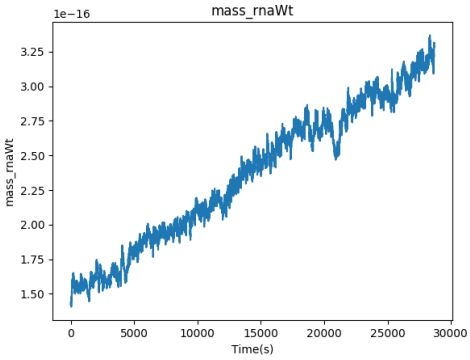
(A) If the cell is pinched



(B) The number of edges of ftsz Ring



(C) Number of specifically bound for RNA Polymerase



(D) The weight of RNA

FIGURE 3.3: The visualisation of the data in wild type simulation.

from others.

For the second type of data which illustrated in figure 3.3 (B), it had an initial value then changing at some point during the simulation such as pinched diameter and number of edges in ftsZ Ring. 2 features were kept in this standard of data: 1) The initial value and 2) the time step the data start changing. If the data remains unchanged, the data changing point was set to -1.

For the time series like the number of specifically bound in RNA polymerase, they had several constant values. The variation was how many time these values transpired. Therefore, the number of each value was counted as the feature of the data. For example, the features of the specifically bound in RNA polymerase are the count of value 0, 1, 2, and 3 respectively.

The last type of data seems more complicated. It seems like every data point contained valuable data. Most of the cases, there was an increasing or decreasing tendency. And the data was nearly linear. The first and the last value were recorded to describe the initial and final status of the data. Then

the linear regression (Seal, 1968) was applied to the data to find a linear representation in order to calculate the gradient and standard deviation of the complex data. The gradient represents the slope of the data. The standard deviation measures the amount of variation of the data. 4 features were remained for the complicated data.

After the time series were cleaned based on different type of data form, the dimensions of the instance were decreased rapidly. On the other hand, the time length of the time series was removed. The length of wild type simulation was in the range of 26,000 to 46,000 seconds. Nevertheless, if the cell did not divide, the time length was 50,000 seconds which was the longest running time in this project. This means time length was the essential feature to classified the mutant from wild type simulation. The time length of the simulations was accounted for a feature. Meanwhile, it was equal to -1 when the simulation time was 50,000 seconds to enlarge the distance between wild type simulation and mutant simulation.

There were 65 features left for 23 time series after dimension reduction. Compared to the data before, which has 50,000 features per time series, it significantly reduced the size of the data. This allowed the more time efficiency computation (Ailon and Chazelle, 2010) due to the key elements of the data remained and the noise reduced. The features are shown in the table 3.2.

Time series	Features
mass_cell	First value, last value, gradient, standard deviation
mass_media	First value, last value, gradient, standard deviation
mass_waterWt	First value, last value, gradient, standard deviation
mass_metaboliteWt	First value, last value, gradient, standard deviation
mass_rnaWt	First value, last value, gradient, standard deviation
mass_proteinWt	First value, last value, gradient, standard deviation
chromosome_ploidy	Time step data reach 2
chromosome_segregated	Time step data start changing
geometry_pinchedDiameter	Initial value, time step data start changing
geometry_pinched	End value
ftszRing_numEdgesOneStraight	Time step data start changing
ftszRing_numEdges TwoStraight	Time step data start changing
ftszRing_numEdgesTwoBent	Time step data start changing
ftszRing_numResidualBent	Time step data start changing
hline ftszRing_numEdges	Initial value, time step data start changing
ribosome_nActive	First value, last value, gradient, standard deviation
ribosome_nNotExist	First value, last value, gradient, standard deviation
ribosome_nStalled	Count of non zero value
rnaPolymerase_nActive	First value, last value, gradient, standard deviation
rnaPolymerase_nSpecifiallyBound	Count of 0, 1, 2, 3 respectively
rnaPolymerase_nNonSpecificallyBound	First value, last value, gradient, standard deviation
rnaPolymerase_nFree	First value, last value, gradient, standard deviation
metabolicReaction_growth	First value, last value, gradient, standard deviation
time length	the length of simulation

TABLE 3.2: The features of 23 remaining time series after reduction.

3.2 Clustering

After cleaning the data, the data from 359 single gene knockouts was applied to the clustering model. The clustering models were built by the scikit-learn library using python in this project. There were 4 clustering models used in this project: K-Means, agglomerative clustering, spectral clustering, and DBSCAN. The results of different clustering model were analysed to achieve the most appropriate model to implement further improvements.

There were 8 types of mutant after single gene knockout. Thus, 8 clusters were set to every model.

3.2.1 K-Means

As using scikit-learn library, it speeds up the converge by deciding the initial centre of the K-Means clustering in a smart way (Pedregosa et al., 2011). Comparing to the random initial centre, the result is stable. The K-Means algorithm will be run 10 times using different centroid seeds. The result was the best output of 10 consecutive runs in terms of inertia. The maximum iteration was 300. It used the expectation maximization algorithm to find the maximum likelihood.

3.2.2 Agglomerative clustering

The key parameter for the agglomerative clustering is the way to calculate the distance. In this project, the Euclidean distance was used to calculate the distance between clusters. Based on the way agglomerative clustering decides the distance between clusters, there are three different linkage criterions. All average linkage, complete linkage, and single linkage were used in this research. The reason why used all linkages was because there was no empirical method to decide which linkage was the best. The performance of the clustering model based on different linkage was analysed to find out the most suitable linkage for the dataset.

3.2.3 Spectral clustering

The spectral clustering is kind of method contains dimension reduction techniques. Therefore, it is a clustering that does the dimension reduction then applies the data to another clustering model. The strategy which used to assign labels in the embedding space was K-Means. The kernel function which implemented to find the affinity was Gaussian kernel also knows as Radial basis function (RBF).

3.2.4 DBSCAN

The DBSCAN clustering will automatically decide how many clusters are in the data, so the number of the clusters do not need to set previously.

The maximum distance between two samples to become neighbourhood is 20,000. The number of samples in a neighbourhood to make a point become a core point is 15. The parameter was set referred to the empirical method. With these parameters, the outliers were fewer so that more instances were clustered.

3.2.5 Performance measurement: adjust rand index

To measure the performance of different clustering models, a standard was required. This is very useful in providing a quantitative criterion for estimating the quality of clustering algorithm (Campello, 2007). Evaluating the accuracy of a clustering model is not just counting errors or the precision and recall like a supervised learning algorithm. Especially, the evaluation matrix should not count the accuracy using the absolute value of the clustering label. It should decide whether the separations of the data clustering by the model are similar to the ground truth. In this project, adjusted rand index (ARI) was calculated. It measures the similarity between two assignments, ignoring permutations and with chance normalisation.

The rand index (Rand, 1971) is an objective criterion for the evaluation of clustering models. It lies between 0 to 1. When two partitions are perfectly matched, it equals to 1.

$$RI = \frac{a+b}{\binom{n_{sample}}{2}} \tag{3.1}$$

Where a(b) is the number of pairs of elements that are in the same (different) set in ground truth class and in the same set in the clustering, the denominator is the total number of possible pairs in the dataset.

Nevertheless, the expected value of rand index of two random partition does not take zero (Yeung and Ruzzo, 2001). To reduce this effect, the expected RI of random labelling can be discounted by introducing adjusted rand index. The adjusted Rand index (Hubert and Arabie, 1985) assumes that external criterion and clustering result is picked at random so that the number of objects in the clusters and criterion is fixed.The adjust rand index is calculated as follow.

$$ARI = \frac{RI - E[RI]}{MAX\,(RI) - E[RI]} \tag{3.2}$$

Where E[RI] is the expected of RI, MAX(RI) is the maximum of RI. Now the random ARI score of the random labelling is 0. The range of ARI scores is between 0 to 1. The negative value means independent labelling. And 1 means perfectly matched. According to scikit-learn library, the labels of the clustering is random due to the clustering algorithm only gives the clusters without labels. With the ARI, while calculating the rand index, the disorder of the label won't affect the result of the result. For example, label (1, 1, 1, 2, 3) is

the same as the label (2, 2, 2, 3, 1).

In this research, the performance of the models was compared using the ARI scores. The labels of the mutants for each single gene knockout were given as the ground truth. The variation between the clustering labels and ground truth was estimated by ARI scores. The result of the clustering was also visualised by graph using principal component analysis in order to figure out the distribution of the clustering comparing to the ground truth. Principal component analysis reduces the dimension of data into 2-dimensions which can be plotted in a 2D graph. This helps to evaluate the result comprehensively. Due to the imbalanced dataset, sometimes the ARI score was high because it classified the main group of data in the dataset correctly. However, the clusters with a small amount of data performs badly in the clustering model. With the graph, the accuracy on each cluster can be compared directly.

Chapter 4

Results and Improvements

After the implement of clustering models, the result showed the characteristic of the different algorithms. The model which was most suitable for the dataset was found. But the accuracy of the model might not good enough, so improvements based on the data processing and the algorithm were made to enhance the performance of the model so that it could be used to predict other simulations. In this chapter, the elementary result of the clusters will be listed and analysed. And the improvements of the agglomerative clustering will be described.

4.1 Result and analysis

The result of the clustering models using 23 time series are shown in the table 4.1 and figure 4.1 below. The diagram was generated by using the principal component analysis to reduce the dimension of the data to 2.

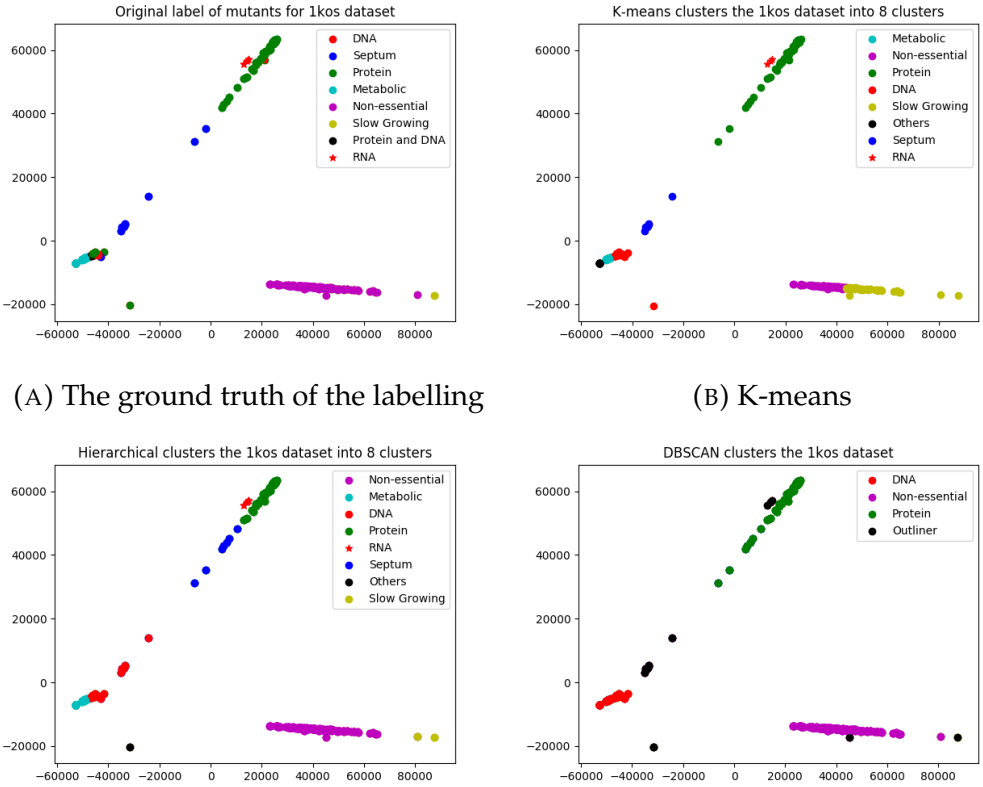
	K-Means	Agglomerative Clustering			DBSCAN	Spectral Clustering
		Complete linkage	Average linkage	Ward linkage		
ARI	0.5621	0.8462	0.6893	0.6865	0.6935	0.4603

TABLE 4.1: The ARI score for clustering models.

From the visualisation of the ground truth which was the manual label of the single gene knockout, it is obvious that the non-essential mutant and slow growing are sparse from others which were easier to be categorised. Slow growing mutant is close to the non-essential due to the slow growing mutant is very close to wild type simulation. Compared to wild type simulation, the slow growing mutant has the slow growth rate. Hence, the slow growing mutant is difficult to categorise. After dimension reduction, the variation was reduced. On the other hand, the protein, DNA, and protein and DNA mutant gather together and mix with each other. This leads to the low accuracy of the clustering. Furthermore, the septum mutant is too sparse that hard to be categorised as one group.

4.1.1 K-Means

K-Means is one of the most widely used clustering models. There were two main problems while using K-Means. One was how to decide the value of



(A) The ground truth of the labelling

(B) K-means

(C) Agglomerative clustering (complete)

(D) DBSCAN

FIGURE 4.1: Visualisation of the different clustering. The plots were applied principal component analysis to reduce the dimensions to 2. A is the ground truth labelling. B is the K-Means. C is the agglomerative clustering with complete linkage. D is the DBSCAN where $\varepsilon = 20000$ and MinPts = 15.

K. And another was how to determine the initial point of the clusters. As the number of the mutants was already decided, the K is set to 8 in this project. Scikit-learn library provides the smart way to find the initial point which speeds up the convergence and optimise the result. These made the K-means easier to apply for the dataset.

As the results show in figure 4.1 (B), compared to the ground truth, K-Means miss labelling the slow growing mutant and non-essential. As 152/359 of the gene knockouts were labelled as Non-essential, only K-Means had a bad performance on the non-essential simulations. This was the reason why the ARI score was low comparing to other clustering models. K-Means worked well on the clusters of protein, RNA, and septum mutants. Because of the merge of protein, DNA, protein and DNA mutants, the clustering model could not identify the protein and DNA mutants. Meanwhile, the number of clusters is determinate. This led to the over categorising of metabolic mutant.

One of the weaknesses of K-Means is that it can only find the spherical clusters. The distribution of the data in this project was more complicated which

was hard to categorise by K-Means. K-Means is also not suitable for imbalanced data in the single gene knockouts simulations. However, the computation cost of K-Means is low. In this research, it was used as a baseline of the clustering.

4.1.2 Agglomerative clustering

The agglomerative clustering with the complete linkage had the best the performance. From the figure 4.1 (C), most of the non-essential metabolic, RNA, and protein mutants were correctly classified. However, according to the empirical method, average linkage should perform better than the complete linkage because it considers every data point in the cluster while calculating the linkage between two clusters.

Comparing with K-Means, it had a higher overall accuracy. It behaved better on the non-essential and metabolic mutant but had a poor performance on septum and DNA mutant.

Empirical studies show that Ward linkage behaves well when the number of data in the different cluster is similar, but it does not perform well when the dataset is imbalanced. The dataset of the mutant was imbalanced, the number of DNA, septum, slow growing, RNA, and protein and DNA mutants were much fewer than protein, metabolic, and non-essential mutant. Consequently, the ward linkage had low accuracy in this research.

Agglomerative clustering is suitable for all shapes for clusters, and it is not sensitive in the order of input of samples. However, the disadvantage of agglomerative clustering is that the time complexity of the algorithm is larger than others. And the result of agglomerative clustering depends on the selection of clustering points and splitting points. The result of the clustering is irreversible. Since the object merges or splits, the next cluster will continue to merge or split on the basis of the previous cluster. As a result, once the clustering result is formed, it is impossible to optimize the performance of clustering.

The calculation of the distance between clusters is based on the absolute value of the dimensions so that applying Euclidean distance needs to ensure that all the features have the same scale. In the single gene knockouts simulation, the different feature had the various scale. For example, comparing to the RNA weight which had the scale of 10^{-16}, the time length was in the range of 26,000 to 50,000. While calculating the distance, even the time steps were 1 second difference which was very tiny and meant nothing will influence a lot comparing to RNA weight while the value was 0 which was completely abnormal than usual.

4.1.3 DBSCAN

The DBSCAN relies on two main parameters: the maximum distance of the neighbourhood and the minimum point in the neighbourhood. The decision of these two parameters is based on experience. And DBSCAN automatically decides the number of clusters. Therefore, there were only 3 clusters in this clustering model. And most of the data are recognised as outlier.

As most of the genes were labelled as non-essential, protein and metabolic mutants. In this experiment, DBSCAN worked well on these parts even it only had 3 clusters. This led to a higher ARI score than K-Means, even it did not classify any other mutants.

DBSCAN can cluster any form of the dense dataset that is better than K-Means which only applies to spherical clusters. And it can recognise the outlier of the dataset that is suitable for clustering the mutant due to there were some mutants had a bias from others. Nonetheless, it performance mainly depends on the setting of the parameters which is more complicated on adjusting parameter than other clustering models. Different combinations of the parameter have a great impact on the clustering result.

4.1.4 Spectral clustering

The spectral clustering had the lowest ARI score among all the clustering methods which was 0.4603. This was a result from the graph is not fully connected. The spectral embedding may not work as expected.

While processing the data, the gradient was calculated as one of the feature. Nevertheless, while the data had a decreasing tendency, the gradient was negative. This led to Negative correlation. If the values of the similarity matrix are not well distributed which means with negative values in this project, the spectral problem will be singular and the problem not solvable (Pedregosa et al., 2011). In this case, a transformation can be applied to the entire matrix. It is common to apply a heat kernel.

$$similarity = e^{-\frac{distance^2}{2\sigma^2}} \tag{4.1}$$

Whereas σ is the standard deviation of the distance.

The computational complexity of spectral clustering is less than K-Means, especially when working on the high dimension data. Spectral Clustering requires the number of the clusters to be specified. It works well for less clusters. However, in this project, the aim was to categorise 8 single gene knockouts mutants. Thus, the spectral clustering is not suitable for clustering the mutants. It might have a better performance on separate the non-essential and essential gene knockout.

4.2 Improvements

According to the result of the clustering models, the reasons which led to the low accuracy of the clustering were 1) the scale of the features was various; 2) the difference between some of the mutants was small, the data was not sparse enough; 3) the limitation of the clustering model; 4) outliers in the raw data.

Based on these problems, some improvements were built to enhance the accuracy of the clustering models. The improvements were mainly tested on the agglomerative clustering which had the best performance so far. It worked well on the large dataset with multiple clusters which was suitable for our dataset.

The improvements which including in this section are feature scaling, feature adjustment, and feature weighted.

4.2.1 Feature scaling

To apply the Euclidean distance, it needs to ensure that all the features have the same scale. After data preprocessing, the features did not have the same scale. All the K-Means, agglomerative clustering, spectral clustering and DBSCAN using Euclidean distance to measure the distance between points and clusters. Consequently, the variation of the scales results to the bad performance of the clustering methods. Since the scale of the raw data varies widely, the feature scaling was essential.

Feature scaling is a method that wildly used to build a standardisation for the range of the features of data (Forman, 2008). In data processing, it is also known as data normalisation.

One of the simplest methods (Aksoy and Haralick, 2001) is to rescaling the raw data in the range of 0 to 1. The formula is given as:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{4.2}$$

In this project, the wild type simulations were used as the standard to normalise the single gene knockouts simulations. In another word, wild type simulations were traversed to find the max(x) and min(x) in the equation above.

Using wild type simulations to normalise the knockout simulations led the data which behaved differently and widely out of the range 0 to 1. This was benefited to sparse different types of data and easier to categorise. After applying feature scaling, the result of the clustering models are shown in table 4.2.

	K-Means	Agglomerative Clustering			DBSCAN
		Complete linkage	Average linkage	Ward linkage	
unnormalised	0.5621	0.8462	0.6893	0.6865	0.6935
normalised	0.5778	0.7996	0.8878	0.7871	0.8694

TABLE 4.2: The comparison of the ARI score of the clustering between unnormalised and normalised data

After normalisation, the ARI score for most of the clustering models had increased. This shows the normalisation was improved the accuracy of the clusters. Before, the time series with the time step feature weight more while calculating the distance; and the time series like RNA weight, pinched value, and growth rate etc. did not influent too much on the result of clustering. However, these time series also the key to determine the mutants.

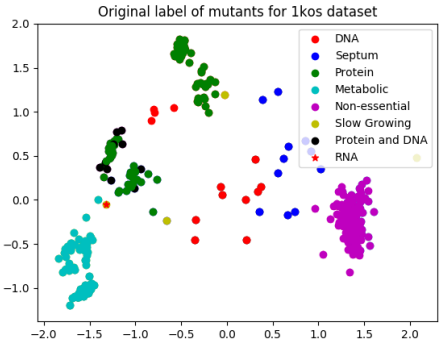
On the other hand, the ARI score of agglomerative clustering using complete linkage was decreased. Because the result before contains the fortune. Complete linkage using the furthest distance between 2 points in two clusters as the distance of the cluster. This made all the time series with time steps features in charge of the clustering. Coincidentally, these features were the key to categorise the non-essential, metabolic, protein, and RNA mutant which including most of the data in the dataset. After normalisation, the importance of the time steps was dramatically decreased. As a result, complete linkage behaved worse than average and Ward linkage which was confirm to empiricism.

From the figure 4.2, it can be found that the group of data was sparser than before. The boundaries between non-essential, protein, metabolic, and the group of protein and protein and DNA mutant were clear which was good to the clustering models. As it shown in the figure 4.2 (C), DBSCAN recognised less data point as outliers than before. And it could find more clusters. But the boundaries between DNA and septum mutants and the boundaries between protein and protein and DNA mutant remained blurred.

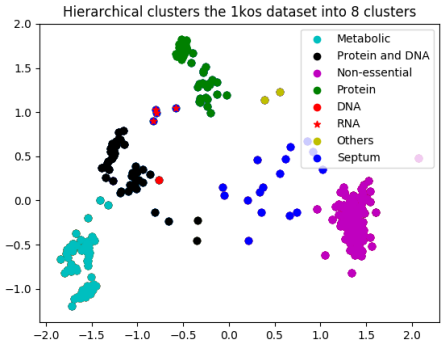
4.2.2 Feature adjustment

After normalisation, the accuracy was not good enough. The highest ARI score was 0.88. To improve the performance of the system, the feature needed to be adjusted to remove the redundant. As it shown in table 3.2, for the ploidy of chromosome and the segregation of chromosome, both of them kept the time step as their feature.

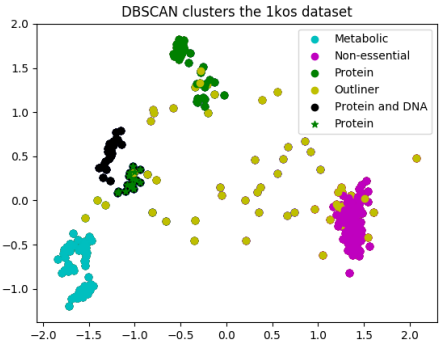
After cleaning the data, it was found that time step started changing for these 2 time series. This also happened on ftsz Ring. The changing point of the number of all kinds of edges was the same. Therefore, the instances contained many redundant features. The cutting off of the time series was not enough. In order to reduce the noise, only five time series were left after improving the cleaning of the data which were growth rate, protein weight, RNA weight, ploidy of chromosome and pinched diameter.



(A) The ground truth of the labelling



(B) Agglomerative clustering (average)



(C) DBSCAN

FIGURE 4.2: Visualisation of the different clustering after normalisation. The plots were applied principal component analysis to reduce the dimensions to 2. A is the ground truth labelling. B is the agglomerative clustering with average linkage. D is the DBSCAN where $\varepsilon = 0.9$ and MinPts = 10.

The result after normalisation indicated that the protein and protein and DNA mutants were merged and needed to be separate. To solve this problem, the raw data of these two mutants was analysed so that the contrast between two mutant could be revealed. The distinction of these mutants was illustrated in figure 4.7.

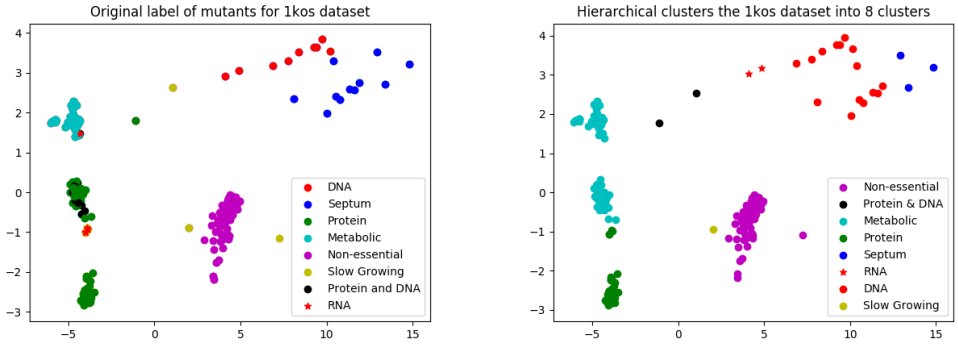
As shown in the figures, the main diversities of protein and protein and DNA mutant were the behaviour of RNA weight and ploidy of chromosome. Before, for the ploidy, the feature was the time step that ploidy reached 2. Both of the protein and protein and DNA mutant did not reach 2. Thus, they were considered as the same after data cleansing.

To overcome this problem, one more feature was added to describe ploidy which was the end value of the ploidy. This could state the variation between protein and protein and DNA mutant. The dimensions of the dataset were reduced to 17 after feature adjustment which listed as table 4.3 below.

Time series	Description	Feature
metabolicReaction_growth	The growth rate of the cell	First value
		Last value
		Gradient
		Standard deviation
mass_proteinWt	The weight of protein	First value
		Last value
		Gradient
		Standard deviation
mass_rnaWt	The weight of RNA	First value
		Last value
		Gradient
		Standard deviation
chromosome_ploidy	The ploidy of the chromosome	Time step data reach 2
		Last value
geometry_pinchedDiameter	The pinched diameter of the cell	Initial value
		Time step data start changing
Time length	The time costing of the simulation	Length of simulation

TABLE 4.3: The time series and features used after feature adjustment.

After improvement, the new instances were applied into the agglomerative clustering. The ARI score of the agglomerative clustering with average linkage was 0.7426 which was worse than before. The results was illustrated in figue 4.3.



(A) The ground truth of the labelling (B) Agglomerative clustering (average)

FIGURE 4.3: Visualisation of the clustering outcomes. The plots were applied principal component analysis to reduce the dimensions to 2. A is the ground truth labelling. B is the agglomerative clustering with average linkage.

From the ground truth, it can be determined that the DNA and septum mutants were sparser than before. However, the protein and protein and DNA mutants were merged. The agglomerative clustering performed better than before one the DNA, Septum, and slow growing mutant. By contrast, it had a low accuracy on the metabolic mutant so that the ARI score was declined.

If the clustering model wanted to recognised protein and protein and DNA mutants, it needed to distinguish the groups of these two mutants from each other. Based on the raw data of these two simulations, the RNA weight and ploidy of chromosome were more important than other time series. Hence, further improvement tried to increase the weight of these two time series.

4.2.3 Feature weighted

Feature weighted is a kind of feature scaling. Compared to the normalisation made before, the aim of weighted the feature is to emphasise some of the features and make others less effect on the clustering result.

The aim of feature weighted was to separate the protein and protein and DNA mutant and categorise them. The decision tree of the types of mutant used in this project is determined as figure 4.4.

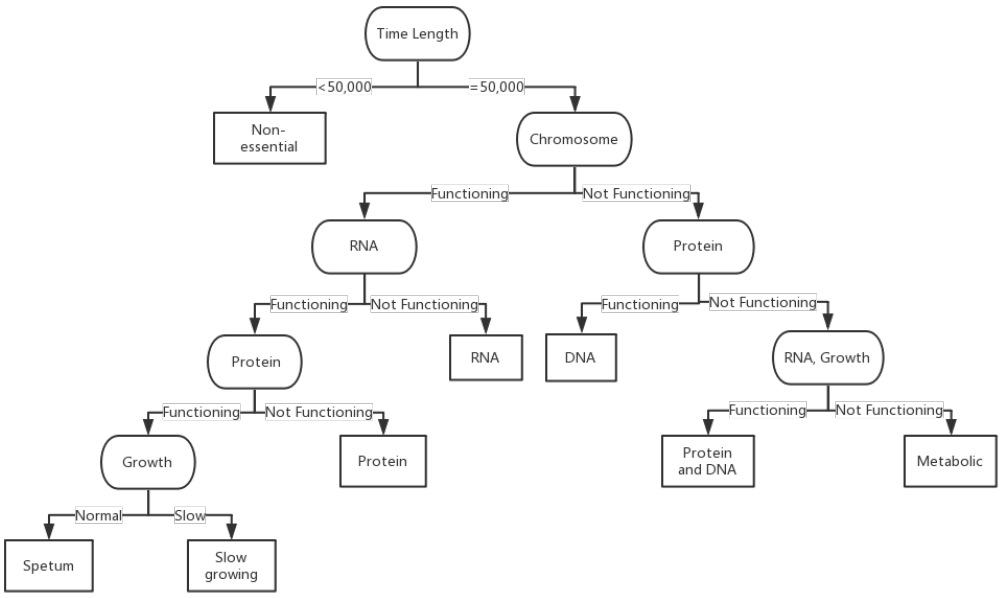


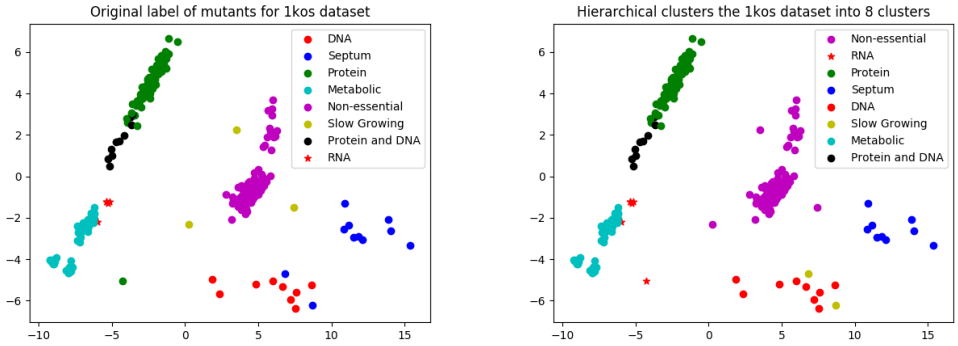
FIGURE 4.4: The decision tree of the types of the mutant. The ellipse is the time series applying to judgement. The square is the leaf node of the tree which is the type of the mutant.

Following the decision tree and the results so far, the weight of the last value of ploidy and gradient and last value of RNA weight was increased. According to the decision rule, the estimation on whether chromosome functioning or not was prior to RNA. Therefore, the weight of ploidy was more than the RNA weight.

In this experiment, the weight of the last value and gradient of the RNA weight was 3 times larger than others. And the weight of the last value of the DNA weight was 6 times more than others. The decision of the weight is

based on the empiricism.

The ARI score of the agglomerative clustering with average linkage was 0.9748. The weight of the feature enhanced the performance of the clustering models. The result is shown in figure 4.5.



(A) The ground truth of the labelling (B) Agglomerative clustering (average)

FIGURE 4.5: Visualisation of the clustering outcomes after weight the ploidy and RNA weight. The plots were applied principal component analysis to reduce the dimensions to 2. A is the ground truth labelling. B is the agglomerative clustering with average linkage.

It shows that every cluster had clear boundaries with others. The protein and protein and DNA mutant departed as expected. All data in the non-essential, metabolic, DNA, and RNA mutant were clustered. However, the slow growing was not categorised. Only 7 single gene knockouts were labelled incorrectly.

4.3 Analysis of the results

The results of the agglomerative clustering with average linkage in different data processing method were shown as follow.

	agglomerative clustering with average linkage	
	23 Time series unnormalised	23 Time series normalised
ARI	0.6893	0.8878
	5 Time series normalised	5 Time series normalised and feature weighted
ARI	0.7426	0.9748

TABLE 4.4: The comparison of during the improvement.

The accuracy of the clustering was enhanced during the improvements of the data processing methods. The final ARI score of the clustering models is 0.9748. Only 7 single gene knockouts were labelled incorrectly which had the

accuracy at 98%. The single gene knockouts which could not be recognised are listed in the table 4.5.

Coding Genes	Ground truth	Clustering labeled
MG_019	Septum	Slow growing
MG_201	Septum	Slow growing
MG_238	Slow growing	Non-essential
MG_254	Protein	RNA
MG_522	Protein and DNA	Protein
MG_400	Slow growing	Non-essential
MG_051	Slow growing	Non-essential

TABLE 4.5: The single gene knockouts which had the wrong label

As the table demonstrated, the all the slow growing mutant were recognised as non-essential because the range of wild type simulation was wide as it shown in figure 4.6. Even all of them were wild type simulation, the growth rate was various. As a result, the slow growing mutant was easily confused as wild type simulation.

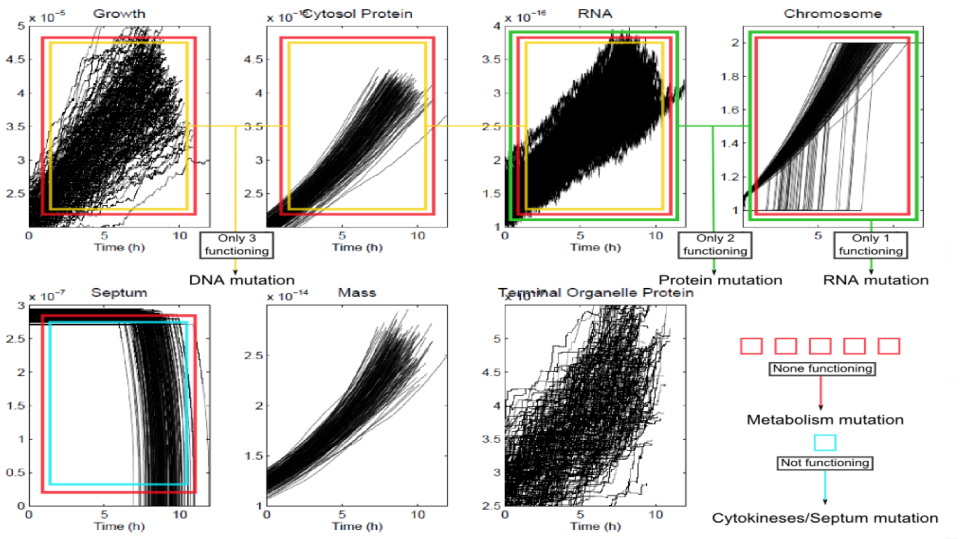


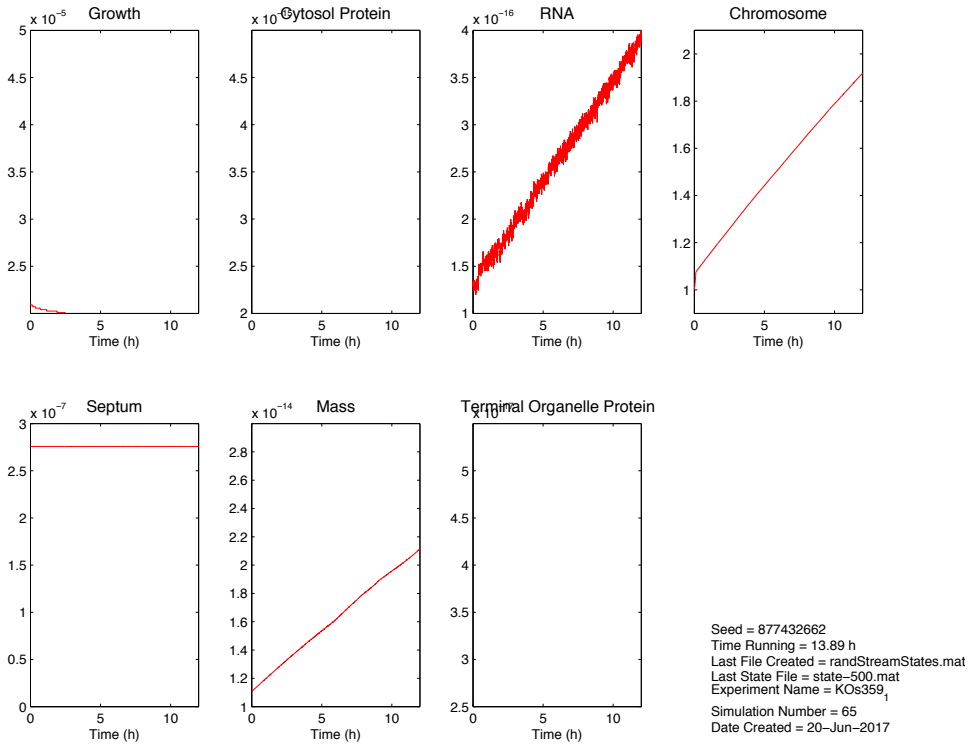
FIGURE 4.6: The raw data of 200 wild type simulations.

One of the protein mutants was recognised as RNA mutant because this was the outlier of the protein mutant. It behaved differently from other protein mutants. As usual, if the protein mutant occurred, only RNA weight and ploidy of the Chromosome are functioning in the 5 time series as wild type simulation. Nevertheless, in MG_254, RNA weight also behaved abnormal. This acted like RNA mutant which only ploidy functioning.

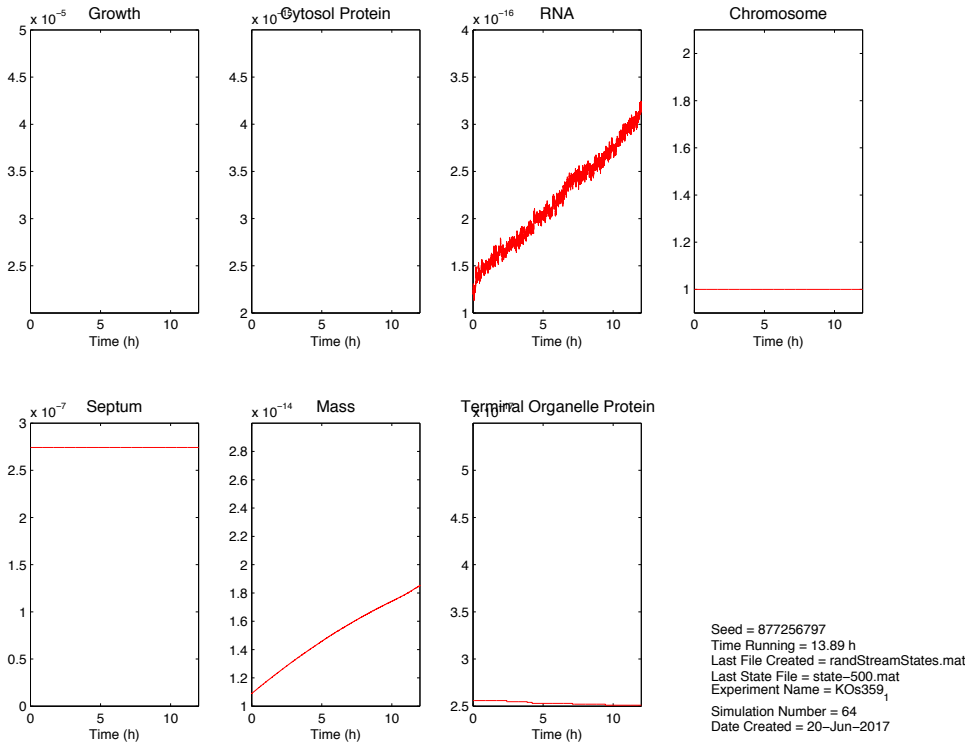
For the septum mutant which was wrong labelling, they were the outliers

of the septum mutant which could be noticed since the first time applied to the clustering methods. MG_019 and MG_201 were isolated from users septum simulation in figure 4.5 (A).

In conclusion, the clustering model had high accuracy in clustering the first run of single gene knockouts simulations. The accuracy of Metabolic, DNA, Protein and DNA, and septum mutant was 100%.



(A) Protein mutant



(B) Protein and DNA mutant

FIGURE 4.7: Comparison between protein and protein and DNA mutant.

Chapter 5

Prediction of the mutants

Due to the high accuracy of the clustering model for the first run of single gene knockouts simulations, the second run of the simulations was used to test the performance of the data. The model was applied to predict the mutants of the second run dataset. Due to the limitation of the dataset, there are only 256 simulations of the single gene knockouts. The distribution of the number of mutants is illustrated in figure 5.1.

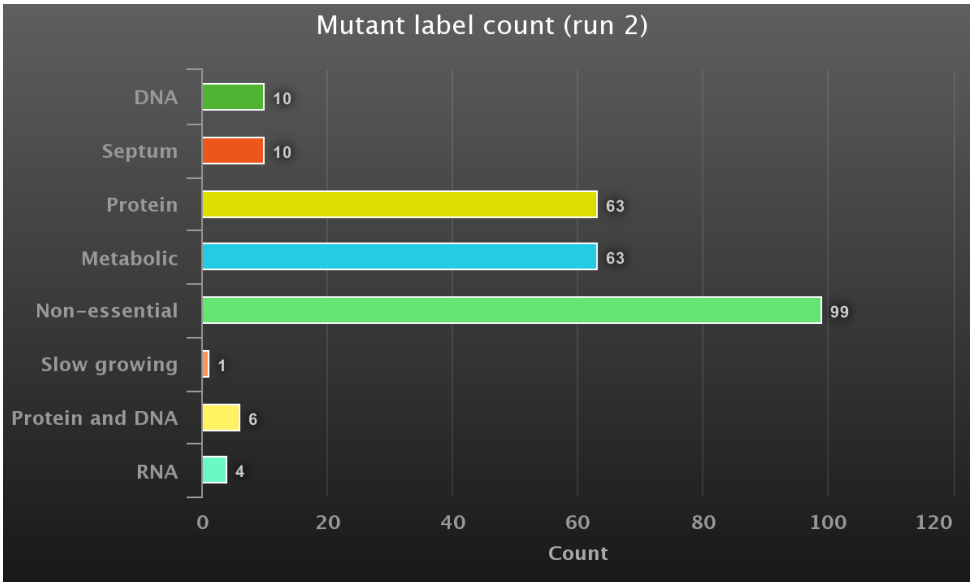


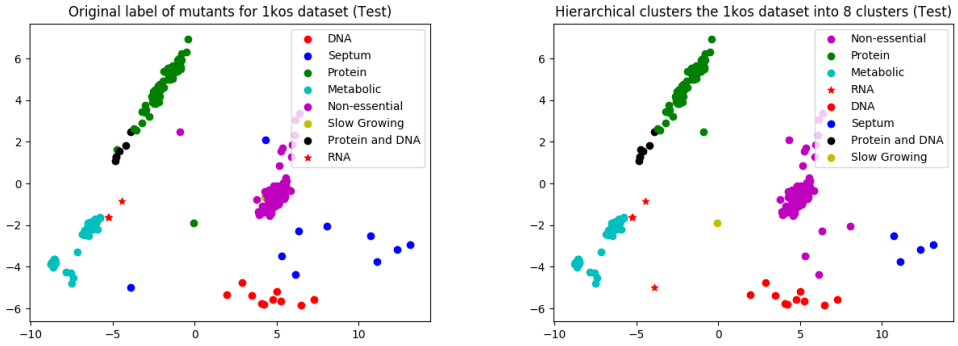
FIGURE 5.1: The count of mutants for the second run of single gene knockouts.

Most of the single gene knockouts were labelled as non-essential. The slow growing was the least among the mutants which only 1 of the knockouts was recognised. The dataset was mainly consisted by non-essential, protein, metabolic mutants. Comparing to the dataset of the first round, the number of non-essential, protein, metabolic mutants were decreased.

The data with 17 features for 5 time series was applied to the agglomerative clustering which had the highest accuracy for the first run dataset. All the features were normalised by 200 wild type simulations. The weight of last value of ploidy and last value and gradient of RNA weight was emphasised.

5.1 Result and analysis

The ARI score of the prediction is 0.9303 which is lower than the ARI score of the clustering model. The result of the clustering was visualisation as figure 5.2.



(A) The ground truth of the labelling (B) The prediction of the dataset

FIGURE 5.2: Visualisation of the prediction of the clustering models. A is the ground truth labelling which is manually labelled based on the behaviour of the cells. B is the result of prediction which produced by agglomerative clustering with average linkage.

The prediction of the metabolite, DNA, Protein mutant was accurate. The slow growing was predicted as non-essential which was the problem left when built the model. Part of the septum mutant was predicted as non-essential. This was because of the lack of the dataset. Part of the non-essential simulations was lacked. While calculating the average linkage of the clusters, every data points in the cluster will be count. Lacking the data influenced the distance calculation of non-essential with other clusters. Therefore, non-essential was merged with septum.

Comparing with non-essential, the pinched diameter of the septum was not functioning. The value of the pinched diameter was not changed during the simulation which meant the cell did not divide. Only one feature in the instance was different than wild type simulation so that it was hard to be predicted by the model.

There were 10 single gene knockouts predicted with wrong labels. The true positive rate of the prediction is calculated as follow.

$$tpr = \frac{TP}{TP + FP} \tag{5.1}$$

Whereas TP is true positive which means the positive instances correctly classified. FP is the false positive which means the positive instances with false category (Flach, 2012). The accuracy of the prediction was 96.1%. The gene knockouts that with false labels are list in the table5.1 below.

Coding Genes	Ground truth	Clustering labeled
MG_019	Septum	Non-essential
MG_201	Septum	Non-essential
MG_203	Septum	Non-essential
MG_238	Septum	Non-essential
MG_239	Slow Growing	Non-essential
MG_003	Septum	Non-essential
MG_254	Septum	RNA
MG_311	Protein	Protein and DNA
MG_049	Protein	Slow growing
MG_084	Non-essential	Protein

TABLE 5.1: The prediction of the second run of single gene knockouts which had the wrong label

Comparing to the result of the clustering model, it can be observed that MG_019, MG_201, MG_238, and MG_254 had the wrong labels in both of the results. MG_019 and MG_201 were the outliers of the septum mutant which were isolated from other septum simulations. MG_238 was the slow growing mutant which was unable to categorise in this model. The ground truth label of MG_238 was changed in the second run of the single gene knockouts simulation. However, the performance of the simulation was the same, only the ploidy of chromosome functioning in this simulation. As a result, MG_238 recognised as RNA mutant in both runs of gene knockouts.

MG_049 which predict the protein mutant as the slow growing was the outlier of protein mutant. It could be recognised from figure 4.5. The data point of MG_049 was far away from the group of protein mutant. The number of clusters was set as 8, and the clustering algorithm cannot give the exact label but clusters. Even the clustering methods could not recognise slow growing, it still determined MG_049 as one cluster.

The cluster worked well on predict the mutants of simulation. Nevertheless, the test is based on the single gene knockouts so that it might perform badly on the multiple gene knockouts. This might due to the limitation of the dataset for models. It will be discussed in the section of the limitation of the system.

Chapter 6

Evaluation

6.1 Advantage of the systems

For the clustering methods, several algorithms were developed to figure out the one most suitable for the dataset. K-Means as one of the most widely used clustering algorithms were used as the baseline to analyse the strengths and weakness of each algorithm. After comparing K-Means, agglomerative clustering, spectral clustering, and DBSCAN, it was found that agglomerative clustering with the average linkage was the best clustering methods for the simulation data. Because agglomerative clustering has a better performance all kinds of clusters and it is less sensitive on the Isolated point. While considering the distance between clustering average linkage took every data points in the cluster into account. This was more comprehensive while categorising the data.

The data processing methods applied in this project which included dimension reduction, feature selecting, and feature scaling was determined based on the simulation data. Dimension reduction and feature scaling reduced the redundant data which was the noise during clustering. And feature scaling helped fit the simulation data to the decision rule of the mutant. It emphasised the key features and weakened the less relevant features. The way to clean the data greatly improve the ARI score from 0.6893 to 0.9748.

Based on the validation and test of the clustering model, the accuracy reached to 98% and 96% respectively. The high accuracy shows that the clustering model can categorise most of the simulations into 8 types of mutants for single gene knockouts. This clustering model can be applied to the automatically classified the single gene knockouts mutants which save time on manual labelling. This model can also be used a basis to categorise more complicated genetic knockouts.

6.2 Limitation

However, the system also contained some limitation. The limitation of the system contains three main aspects, the calculation of the accuracy, the size of the dataset. And the modulation of the parameter and scale.

6.2.1 Accuracy calculating

During the prediction part of the project, the test set of the genetic knock out was not completed. It did not contain all single gene knockouts. This might influent the performance of the clustering. As a result, the accuracy of the prediction had some bias. While calculating the average linkage of the clusters, every data points in the cluster will be count. Lacking the data influenced the distance calculation between clusters and led to the bias of clusters gathering in the agglomerative clustering. To fix this problem, the completed dataset is needed.

6.2.2 Dataset

The dataset only contained the single gene knockouts. Therefore, the mutant classification might only suitable for single gene knockouts. When you try to find the minimal gene group of the cell, single knockouts just the most elementary step. During the research of minimal genomes, much more than one gene will be knocked out. This is a limitation of the application of the modelling. Thus, the performance of the clustering model might become worse with a larger size of the dataset. To make the clustering model has a wide application, a larger dataset contains more kinds of genetic knockouts will be required.

6.2.3 Parameter and scale modulation

To build a model with high accuracy, the parameter was tuning so that it can fix the desired output of the cluster. The different combinations of the parameters effected the result of the clusters. In this project, the parameter of the DBSCAN and the weight of the features were decided by empirical methods which mainly based on the ARI score of the clustering. The result of the parameter and weight modulation might not achieve the optimal performance of the clustering model. To optimise the parameter and weight tuning, some algorithm can be applied to automatically find the optimal parameters and weight for the clustering model. This can greatly improve the accuracy of the model.

Chapter 7

Conclusion

There were several achievements had been obtained according to the research of the project. During applying the whole cell model, the idea of the structure and algorithm of the model was obtained. The theory of clustering models such as K-Means, agglomerative clustering, DBSCAN, and spectral clustering was researched during the implementation of the models.

After applied the data processing method, the dimension of the dataset had been reduced from 60,000 time series with 50,000 features for each to 5 time series with 17 features overall.

The comparison between the clustering algorithm was made which showed that agglomerative clustering with the average linkage had the best performance on the dataset.

After improving the clustering model, the ARI score of the models had an improvement from 0.6893 to 0.9748. The accuracy of the prediction of the mutants was 98%.

The clustering model had a high accuracy on clustering the simulation data for single gene knockouts. However, the models still have some drawbacks such as parameter modulation and feature scaling. Some future works can be done to overcome the limitation of the system.

7.1 Future works

Due to the limitations of the parameter tuning and feature scaling which appeared in the project, an algorithm can be applied to automatically find the optimal parameters for the DBSCAN and the weight of the features. One of the algorithms which widely used to find the optimum is the genetic algorithm.

Genetic algorithm (Holland, 1992) is an optimization algorithm inspired by the natural selection and genetic mechanism of the genetics. The idea of the genetic algorithm is to simulate the creature and human evolution in order to solve the complex optimization (Grefenstette, 1986).

During the breeding process, the gene in the chromosome will have the crossover, mutation. And the individual with low fitness will be phased out. And the individuals of better adaptability will be increased. The evolution of creatures takes place in groups, and such a group is called a population. Individual means a single organism that makes up a population. After several generations of the natural selection, the individual preserved will have the high fitness, which is likely to contain the individual that produced with the highest fitness in the history. In each generation, the new individual inherits the gene from both sides of the parents, while there is a probability of genetic mutant.

The general steps of the genetic algorithm are summarised as follow.

Start the cycle until you find a satisfactory solution

1. Evaluate the fitness of the chromosome for each individual
2. Follow the selecting principle which is the higher the fitness, the greater the probability to be select. And select two individuals from the population as the parent to produce the offspring
3. Extract the chromosomes of both parents. Crossover and produce the offspring
4. Mutate the chromosomes of the offspring
5. Repeat steps 2, 3 and 4 until the new population is produced

We use the genetic algorithm to optimise the parameters for the DBSCAN and the weight for the features. Taking the weight of the features as an example. The implementation of the genetic algorithm is like the evolution of nature. We first look for a "digital" coding scheme for the potential solution to the problem in order to establish a mapping between phenotype and genotype. In this case, the distribution of the weights for each feature is the digital code. Then initialise population which is a group of distributions with a random number, and the individuals are these digital codes inside the population. After the appropriate decoding process, the fitness function is used to assess the fitness of each individual. For this project, the fitness function is the ARI score of the cluster. If the ARI score is high, the individual has a high fitness. Using some specific selection function to choose the parent for next generation. For example, 20 individuals with the highest fitness. Let the gene of the individual implement the crossover and mutation. And then produce the offspring. This means two distribution of the weight will cross to produce the new distribution. And some of the data points in the distribution will randomly change to another value. This process will keep repeating which is the evolution of the individuals. After we get the satisfied ARI score, the evolution will be stopped (Brindle, 1981).

Genetic algorithm does not guarantee that you can get the optimal solution

to the problem, sometimes it will be stuck in the local optimum. However, the biggest advantage of using genetic algorithms is that you do not have to understand and worry about how to find the optimal solution. Compared to modulate the parameters and weight manually, with the genetic algorithm, the accuracy of the clustering model will be automatically adjusted to the optimum.

Appendix A

Essential parts of the code

A.1 Data processing

```
# This is the data preprocessing methods with normalisation and feature

# normalised the data using (data-min)/(max-min)
def normalise_data(count, data, Max, Min):
    index = count
    if Max[index] != Min[index]:
        value = (data-Min[index])/(Max[index]-Min[index])
    else:
        value = data
    return value

# get the scale used to normalise the data
def get_normalisation_scale():
    Max, Min = normalise.normalisation()
    return Max, Min

# read the data by time series
def get_data(example_data, column_name):
    # example_data = pd.read_pickle(file_name)
    data = example_data[column_name]
    d = list(data)
    # print(d)
    return d

# calculate the standard deviation
def cal_std(data):
    # print('std')
    std = np.std(data)
    return std

# calculate the gradient of the linear regression
def cal_gradient(data):
    # print('gradient')
```

```
    # gradient = np.gradient(data)
    # print(gradient)
    x = range(1, len(data)+1)
    gradient, intercept, r_value, p_value, std_err = stats.linregres
    return gradient

# get the first and last value of the input data
def get_value(data):
    # print('first value')
    # print('last value')
    first_value = data[0]
    last_value = data[-1]
    return first_value, last_value

# get the time step the data start to increase
def get_changepoint_increase(data):
    # print('data changing point')
    d = pd.Series(data)
    change = d[d.pct_change() > 0].index.tolist()
    if not change:
        change_point = -1
    else:
        change_point = change[0]+1
    return change_point

# get the time step the data start to decrease
def get_changepoint_decrease(data):
    # print('initial data')
    # print('data changing point')
    d = pd.Series(data)
    initial_value = d[0]
    change = d[d.pct_change() < 0].index.tolist()
    if not change:
        change_point = -1
    else:
        change_point = change[0]+1
    return initial_value, change_point

# get the time step ploidy reach to 2, and the finaly value of ploid
def get_changepoint_ploidy(data):
    # print('data changing point')
    if data[-1] == 2:
        change_point = data.index(2)+1
    else:
        change_point = -1
    end_value = data[-1]
    return change_point, end_value
```

```
# get the length of the simulation
def get_end(data):
    # print('length of simulation')
    end_point = len(data)+1
    if end_point >= 50000:
        end_point = -1
    return end_point

# count the number of non-zero data
def count_nonzero(data):
    # print('number of non zero value')
    count = np.count_nonzero(data)
    return count

# count the number of values equal to 1,2,3,4 respectively
def count_values(data):
    # print('number of 0')
    # print('number of 1')
    # print('number of 2')
    # print('number of 3')
    count = []
    # unique = np.unique(data)
    for num in [0,1,2,3]:
        count.append(data.count(num))
    return count

# get the pinched value (1 or 0)
def pinched_value(data):
    # print('pinched value')
    value = data[-1]
    return value

# data processing
def data_preprocess(path):
    # path = '/Users/superxzz/Documents/project/panda_dataframe/wildtyp
    # column = ['mass_total', 'mass_cell', 'mass_cellDry', 'mass_media'
    column = ['metabolicReaction_growth', 'mass_proteinWt', 'mass_rnaWt
    x = []
    # y = []
    Max, Min = get_normalisation_scale()
    print(Max)
    print(Min)
    for filename in glob.glob(os.path.join(path, '*.pickle')):
        # print(filename)
        data_x = []
        count = 0
```

```
# scales = []
# file_path = path+'/'+filename
# print(file_path)
example_data = pd.read_pickle(filename)
for column_name in column:
    # print(count)
    # print(column_name+':')
    data = get_data(example_data, column_name)
    # data = pd.Series(d)
    # print(data[0])
    if column_name =='geometry_pinched':
        v = pinched_value(data)
        value = normalise_data(count,v,Max, Min)
        data_x.append(value)
        count +=1
    elif column_name == 'ribosome_nStalled':
        v = count_nonzero(data)
        # print(value)
        value = normalise_data(count, v, Max, Min)
        data_x.append(value)
        count +=1
    elif column_name == 'rnaPolymerase_nSpecificallyBound':
        value = count_values(data)
        # print(value)
        for v in value:
            val = normalise_data(count, v, Max, Min)
            data_x.append(val)
            count += 1
    elif column_name == 'chromosome_ploidy':
        v,e= get_changepoint_ploidy(data)
        value = normalise_data(count, v, Max, Min)
        data_x.append(value)
        count += 1
        end = 6*normalise_data(count,e, Max, Min)
        # end = normalise_data(count, e, Max, Min)
        data_x.append(end)
        # scales.append(value)
        count += 1
    elif column_name =='geometry_pinchedDiameter' or column_
        # print(data)
        initial_value, changepoint = get_changepoint_decreas
        initial = normalise_data(count, initial_value, Max,
        # value = scale_data(initial, scales[3], scales[0],
        data_x.append(initial)
        count += 1
        change = normalise_data(count, changepoint, Max, Min
        # value = scale_data(change, scales[3], scales[0], s
```

```
                data_x.append(change)
                count += 1
            elif column_name == 'chromosome_segregated' or column_name
                # print(data)
                v = get_changepoint_increase(data)
                value = normalise_data(count, v, Max, Min)
                data_x.append(value)
                count += 1
            elif column_name == 'mass_rnaWt':
                start, end = get_value(data)
                start_value = normalise_data(count, start, Max, Min)
                data_x.append(start_value)
                count += 1
                end_value = 3*normalise_data(count, end, Max, Min)
                # end_value = normalise_data(count, end, Max, Min)
                data_x.append(end_value)
                count += 1
                gradient = cal_gradient(data)
                value = 3*normalise_data(count, gradient, Max, Min)
                # value = normalise_data(count, gradient, Max, Min)
                data_x.append(value)
                count += 1
                # data_x.append(intercept)
                std = cal_std(data)
                value = normalise_data(count, std, Max, Min)
                data_x.append(value)
                # scales.append(value)
                count += 1
            elif column_name != 'mass_total' and column_name != 'mass_c
                start, end = get_value(data)
                start_value = normalise_data(count, start, Max, Min)
                data_x.append(start_value)
                count += 1
                end_value = normalise_data(count, end, Max, Min)
                data_x.append(end_value)
                count += 1
                gradient = cal_gradient(data)
                value = normalise_data(count, gradient, Max, Min)
                data_x.append(value)
                count += 1
                # data_x.append(intercept)
                std = cal_std(data)
                value = normalise_data(count, std, Max, Min)
                data_x.append(value)
                count += 1
        # print(len(data_x))
    timelength = get_end(data)
```

```
            value = normalise_data(count, timelength, Max, Min)
            data_x.append(value)
            # print(data_x)
            # print(count)
            # print(len(data_x))
            x.append(data_x)
        # print(x)
        return x
```

A.2 Clustering algorithms

```
cluster_pred = []

# get the data for clustering (run 1 of single gene knockouts)
cluster_data = data_pre_normal.data_preprocess('/Users/superxzz/Docu
# get the data for prediction (run 2 of single gene knockouts)
predict_data = data_pre_normal.data_preprocess('/Users/superxzz/Docu

# K-Means
print('K-Means')
kmeans = KMeans(n_clusters=8).fit(cluster_data)
cluster_pred = kmeans.labels_
accuracy1 = metrics.adjusted_rand_score(cluster_true, cluster_pred)
print(accuracy1)

# the visualisation of the result of clustering using PCA
pca = PCA(n_components=2).fit(cluster_data)
pca_2d = pca.transform(cluster_data)
for i in range(0, pca_2d.shape[0]):
    if cluster_pred[i] == 0:
        c1 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='c')
    elif cluster_pred[i] == 1:
        c2 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='m')
    elif cluster_pred[i] == 2:
        c3 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='g')
    elif cluster_pred[i] == 3:
        c4 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='r')
    elif cluster_pred[i] == 4:
        c5 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='y')
    elif cluster_pred[i] == 5:
        c6 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='k')
    elif cluster_pred[i] == 6:
        c7 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='b')
    elif cluster_pred[i] == 7:
        c8 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='r', marker='*

pl.legend([c1, c2, c3, c4, c5, c6, c7, c8], ['Cluster 1', 'Cluster 2
```

```
pl.title('K-means clusters the 1kos dataset into 8 clusters')
# pl.ylim((-10000,5000))
pl.savefig('plot_kmeans')

# plot the original label of dataset as the ground truth
for i in range(0, pca_2d.shape[0]):
    if cluster_true[i] == 0:
        c1 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='r')
    elif cluster_true[i] == 1:
        c2 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='b')
    elif cluster_true[i] == 2:
        c3 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='g')
    elif cluster_true[i] == 3:
        c4 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='c')
    elif cluster_true[i] == 4:
        c5 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='m')
    elif cluster_true[i] == 5:
        c6 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='y')
    elif cluster_true[i] == 6:
        c7 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='k')
    elif cluster_true[i] == 7:
        c8 = pl.scatter(pca_2d[i, 0], pca_2d[i, 1], c='r', marker='*')

pl.legend([c1, c2, c3, c4, c5, c6, c7, c8], ['DNA', 'Septum', 'Protein'
pl.title('Original label of mutants for 1kos dataset')
# pl.ylim((-10000,5000))
pl.savefig('plot_original')


# spectral clustering
print('spectral clustering')
spectral = cluster.SpectralClustering(n_clusters= 8 , eigen_solver='arp
cluster_pred = spectral.labels_
accuracy1 = metrics.adjusted_rand_score(cluster_true, cluster_pred)
print(accuracy1)

# agglomerative clustering with complete linkage
print('agglomerative clustering with complete linkage')
hierarchical = cluster.AgglomerativeClustering(n_clusters=8, linkage="c
cluster_pred = hierarchical.labels_
# print(cluster_pred)
accuracy1 = metrics.adjusted_rand_score(cluster_true, cluster_pred)
print(accuracy1)


# agglomerative clustering with ward linkage
print('agglomerative clustering with ward linkage')
```

```
hierarchical = cluster.AgglomerativeClustering(n_clusters=8, linkage
cluster_pred = hierarchical.labels_
# print(cluster_pred)
accuracy1 = metrics.adjusted_rand_score(cluster_true, cluster_pred)
print(accuracy1)


# agglomerative clustering with average linkage
hierarchical = cluster.AgglomerativeClustering(n_clusters=8, linkage
cluster_h = hierarchical.fit(cluster_data)
cluster_pred = cluster_h.labels_
# print(cluster_pred)
accuracy1 = metrics.adjusted_rand_score(cluster_true, cluster_pred)
print(accuracy1)

# prediction by agglomerative clustering with average linkage
hierarchical_label=[]
hierarchical_test = hierarchical.fit_predict(predict_data)
# cluster_result = hierarchical_test.labels_
print('result of prediction')
accuracy2 = metrics.adjusted_rand_score(ground_true, hierarchical_te
print(accuracy2)

# DBSCAN with eps =0.9 MinPt =10
dbscan = cluster.DBSCAN(eps=0.9, min_samples=10).fit(cluster_data)
cluster_pred = dbscan.labels_
accuracy1 = metrics.adjusted_rand_score(cluster_true, cluster_pred)
print(accuracy1)
```

Bibliography

Ailon, Nir and Bernard Chazelle (2010). "Faster dimension reduction". In: *Communications of the ACM* 53.2, pp. 97–104.

Aksoy, Selim and Robert M Haralick (2001). "Feature normalization and likelihood-based similarity measures for image retrieval". In: *Pattern recognition letters* 22.5, pp. 563–582.

Ansari, Sharaf et al. (2013). "An overview of clustering analysis techniques used in data mining". In: *International Journal of Emerging Technology and Advanced Engineering* 3.12, pp. 284–286.

Blackstone, Neil W (2001). "New Biological Books Reviews and Brief Notices The Cell: A Molecular Approach Geoffrey M. Cooper". In: *The Quarterly Review of Biology* 76.2.

Brindle, Anne (1981). "Genetic algorithms for function optimization". In:

Cadez, Igor V, Padhraic Smyth, and Heikki Mannila (2001). "Probabilistic modeling of transaction data with applications to profiling, visualization, and prediction". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 37–46.

Campello, Ricardo JGB (2007). "A fuzzy extension of the Rand index and other related indexes for clustering and classification assessment". In: *Pattern Recognition Letters* 28.7, pp. 833–841.

Carson, Chad et al. (2002). "Blobworld: Image segmentation using expectation-maximization and its application to image querying". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.8, pp. 1026–1038.

De Sa, JP Marques (2012). *Pattern recognition: concepts, methods and applications*. Springer Science & Business Media.

Donalek, Ciro (2011). "Supervised and Unsupervised learning". In: *Astronomy Colloquia. USA*.

Ester, Martin et al. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Vol. 96. 34, pp. 226–231.

Fiedler, Miroslav (1975). "A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory". In: *Czechoslovak Mathematical Journal* 25.4, pp. 619–633.

Flach, Peter (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.

Forman, George (2008). "BNS feature scaling: an improved representation over tf-idf for svm text classification". In: *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, pp. 263–270.

Fraser, Claire M et al. (1995a). "The minimal gene complement of Mycoplasma genitalium". In: *science*, pp. 397–403.

— (1995b). "The minimal gene complement of Mycoplasma genitalium". In: *science*, pp. 397–403.

Fred, Ana and José Leitão (2000). "Partitional vs hierarchical clustering using a minimum grammar complexity approach". In: *Advances in Pattern Recognition*, pp. 193–202.

French, Carl (1996). *Data Processing and Information Technology*. Cengage Learning EMEA.

Ghahramani, Zoubin (2004). "Unsupervised learning". In: *Advanced lectures on machine learning*. Springer, pp. 72–112.

Grefenstette, John J (1986). "Optimization of control parameters for genetic algorithms". In: *IEEE Transactions on systems, man, and cybernetics* 16.1, pp. 122–128.

Han, Jiawei, Jian Pei, and Micheline Kamber (2011). *Data mining: concepts and techniques*. Elsevier.

Holland, John H (1992). "Genetic algorithms". In: *Scientific american* 267.1, pp. 66–73.

Huang, Zhexue (1997). "A fast clustering algorithm to cluster very large categorical data sets in data mining." In: *DMKD* 3.8, pp. 34–39.

Hubert, Lawrence and Phipps Arabie (1985). "Comparing partitions". In: *Journal of classification* 2.1, pp. 193–218.

Jain, Anil K, M Narasimha Murty, and Patrick J Flynn (1999). "Data clustering: a review". In: *ACM computing surveys (CSUR)* 31.3, pp. 264–323.

Karr, Jonathan R et al. (2012). "A whole-cell computational model predicts phenotype from genotype". In: *Cell* 150.2, pp. 389–401.

Lee, Ruby, Jonathan R Karr, and Markus W Covert (2013). "WholeCellViz: data visualization for whole-cell models". In: *BMC bioinformatics* 14.1, p. 253.

López-Madrigal, Sergio et al. (2011). "Complete genome sequence of ?Candidatus Tremblaya princeps? strain PCVAL, an intriguing translational machine below the living-cell status". In: *Journal of bacteriology* 193.19, pp. 5587–5588.

MacQueen, James et al. (1967). "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA., pp. 281–297.

McCune, Jenny C (1998). "Data, data, everywhere". In: *Management Review* 87.10, p. 10.

McKinney, W (2014). "Pandas, Python Data Analysis Library. 2015". In: *Reference Source*.

Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar (2012). *Foundations of machine learning*. MIT press.

Pearson, Karl (1901). "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11, pp. 559–572.

Pedregosa, Fabian et al. (2011). "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.Oct, pp. 2825–2830.

Rand, William M (1971). "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical association* 66.336, pp. 846–850.

Razin, Shmuel and Leonard Hayflick (2010). "Highlights of mycoplasma research?an historical perspective". In: *Biologicals* 38.2, pp. 183–190.

Richardson, Mark (2009). "Principal component analysis". In: *URL: http://people. maths. ox. ac. uk/richardsonm/SignalProcPCA. pdf (last access: 3.5. 2013). Aleš Hladnik Dr., Ass. Prof., Chair of Information and Graphic Arts Technology, Faculty of Natural Sciences and Engineering, University of Ljubljana, Slovenia ales. hladnik@ ntf. uni-lj. si*.

Seal, Hilary L (1968). *The historical development of the Gauss linear model*. Yale University.

Shlens, Jonathon (2014). "A tutorial on principal component analysis". In: *arXiv preprint arXiv:1404.1100*.

Ştefan, Raluca-Mariana (2012). "A Comparison of Data Classification Methods". In: *Procedia Economics and Finance* 3, pp. 420–425.

Sung, Bong Hyun et al. (2016). "Construction of a minimal genome as a chassis for synthetic biology". In: *Essays in biochemistry* 60.4, pp. 337–346.

Tully, JosephG et al. (1981). "A newly discovered mycoplasma in the human urogenital tract". In: *The Lancet* 317.8233, pp. 1288–1291.

TutorVisa. *Cyanobacteria Mycoplasmas*. URL: http://www.tutorvista.com/content/biology/biology-iii/kingdoms-living-world/cyanobacteria.php (visited on 09/03/2017).

Von Luxburg, Ulrike (2007). "A tutorial on spectral clustering". In: *Statistics and computing* 17.4, pp. 395–416.

Wang, Jue D and Petra A Levin (2009). "Metabolism, cell growth and the bacterial cell cycle". In: *Nature reviews. Microbiology* 7.11, p. 822.

Wold, Svante, Kim Esbensen, and Paul Geladi (1987). "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3, pp. 37–52.

Wolpert, David H and William G Macready (1997). "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1, pp. 67–82.

Yeung, Ka Yee and Walter L Ruzzo (2001). "Details of the adjusted rand index and clustering algorithms, supplement to the paper an empirical study on principal component analysis for clustering gene expression data". In: *Bioinformatics* 17.9, pp. 763–774.