SUPPORT VECTOR MACHINE – RECURSIVE FEATURE ELIMINATION FOR FEATURE SELECTION OF MULTI-OMICS DATA OF LUNG CANCER

JI TONG LIN

UNIVERSITI TEKNOLOGI MALAYSIA

PSZ 19:16 (Pind. 1/13)

UNIVERSITI TEKNOLOGI MALAYSIA

DECLARATION OF THESIS / UNDERGRADUATE PROJECT REPORT AND COPYRIGHT

Author's full name : JI TONG LIN

Date of Birth : 29/1/1998

Title : SUPPORT VECTOR MACHINE – RECURSIVE FEATURE ELIMINATION FOR FEATURE SELECTION OF MULTI-OMICS DATA OF LUNG CANCER

Academic Session : 2021/2022

I declare that this thesis is classified as:

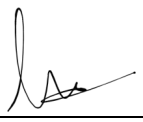
☐ **CONFIDENTIAL** (Contains confidential information under the Official Secret Act 1972)*

☐ **RESTRICTED** (Contains restricted information as specified by the organization where research was done)*

☑ **OPEN ACCESS** I agree that my thesis to be published as online open access (full text)

1. I acknowledged that Universiti Teknologi Malaysia reserves the right as follows:
2. The thesis is the property of Universiti Teknologi Malaysia
3. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
4. The Library has the right to make copies of the thesis for academic exchange.

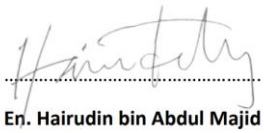
Certified by:



SIGNATURE OF STUDENT

A18CS0338

MATRIX NUMBER



HAIRUDIN BIN ABDUL MAJID
Pensyarah
Fakulti Sains Komputer & Sistem Maklumat
Universiti Teknologi Malaysia
81310 Skudai
Johor

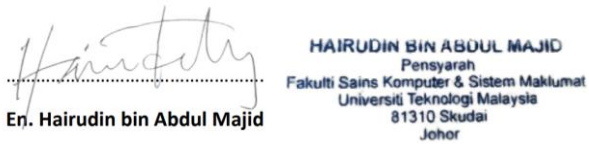
SIGNATURE OF SUPERVISOR

DR AZURAH BINTI A SAMAH

NAME OF SUPERVISOR

NOTES : If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organization with period and reasons for confidentiality or restriction

"I hereby declare that we have read this thesis and in my opinion this thesis is suffcient in term of scope and quality for the award of the degree of Bachelor of Computer Science (Bioinformatics)"



Signature : _____

Name of Supervisor : DR AZURAH BINTI A SAMAH

Date : 30 JUNE 2022

SUPPORT VECTOR MACHINE – RECURSIVE FEATURE ELIMINATION FOR FEATURE SELECTION OF MULTIOMICS DATA OF LUNG CANCER

JI TONG LIN

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Bachelor of Computer Science (Bioinformatics)

School of Computing
Faculty of Engineering
Universiti Teknologi Malaysia

JUNE 2022

DECLARATION

I declare that this thesis entitled *"Support Vector Machine – Recursive Feature Elimination for Feature Selection of Multiomics Data of Lung Cancer"* is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature : .....

Name : JI TONG LIN

Date : 29 JUNE 2022

DEDICATION

This thesis is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my mother, who taught me that even the largest task can be accomplished if it is done one step at a time.

ACKNOWLEDGEMENT

In preparing this thesis, I was in contact with many people, researchers, academicians, and practitioners. They have contributed towards my understanding and thoughts. In particular, I wish to express my sincere appreciation to my main thesis supervisor, Dr. Azurah binti A Samah, for encouragement, guidance, critics and friendship. I am also very thankful to Mr. Hairudin bin Abdul Majid for his guidance, advices and motivation. Without their continued support and interest, this thesis would not have been the same as presented here.

My fellow student should also be recognised for their support. My sincere appreciation also extends to all my colleagues and others who have aided me at various occasions. Their views and tips are useful indeed. Unfortunately, it is not possible to list all of them in this limited space. I am grateful to all my family member.

ABSTRACT

This study aims to apply Support Vector Machine – Recursive Feature Elimination (SVM-RFE) in feature selection on multi-omics data related to lung cancer. Lung cancer has been one of the deadliest cancers and is infecting individuals and causing deaths every year. Biological data especially multi-omics data, which has the properties of big data, has been growing exponentially every year due to the emergence of more advanced technologies. Multi-omics is a biological term which refers to the integration or combination of two or more omics. Multi-omics has become increasingly more popular every year due to its contribution in cancer diagnosis and prognosis. However, multi-omics is not a very straightforward field as it involves multiple omics. Each individual omics are already comprised of thousands of features, therefore making the dimensionality of multi-omics even higher and more difficult to perform analyses. This study realises the importance of efficient feature selection in multi-omics data, specifically in extracting informative feature subset that contribute the most to yield the most optimal classification result. Therefore, the study proposed a solution of performing RFE using SVM algorithm in feature selection phase on multi-omics data comprises of genomics, epigenomics and transcriptomics. Of the 20 selected feature subsets, the smallest 5 of them are used as input to two autoencoder algorithms, which are Stacked Denoising Autoencoder (SDAE) and Variational Autoencoder (VAE) for classification and evaluation. From the result, it can be seen that the VAE model is able to achieve AUC score of 1.0 for all feature subsets, while the SDAE model achieve AUC score of 1.0 for only the Feature Subset 1000 (FS 1000) and Feature Subset 4000 (FS 4000), while the rest of feature subsets resulted in 0.5 AUC score. Both the SDAE and VAE algorithms are able to correctly classify all the instances in the test set for FS 1000 and 4000. The study concludes that FS 1000 and 4000 are the two most optimal feature subsets selected by the SVM-RFE algorithm.

ABSTRAK

Kajian ini dilakukan bertujuan untuk mengaplikasikan algoritma Support Vector Machine – Recursive Feature Elimination (SVM-RFE) dalam proses pemilihan ciri atas data multi-omics yang berkaitan dengan kanser paru-paru. Kanser paru-paru merupakan antara kanser yang paling memudaratkan dan telah menjangkiti dan mengambil nyawa banyak individu pada setiap tahun. Data biologi terutamanya data multi-omics yang memiliki ciri-ciri data besar telah berkembang setiap tahun secara besar-besaran disebabkan oleh teknologi biologi yang semakin canggih. Multi-omics merupakan istilah biologi yang merujuk kepada kombinasi dua atau lebih jenis data omics. Multi-omics telah menjadi semakin popular kerana ia membawa sumbangan ketara dalam pengenalan dan ramalan penyakit. Namun begitu, multi-omics merupakan bidang yang berjenis terus terang disebabkan ia merupakan kombinasi beberapa jenis data omics. Setiap data omics mengandungi beribu-ribu ciri data, dan ini turut menyebabkan dimensi multi-omics lebih tinggi dan sukar untuk dijalankan pelbagai analisis. Kajian tersebut sedar akan kepentingan proses pemilihan ciri atas data multi-omics yang lebih kecap, terutamanya dalam mengeluarkan subset ciri yang bermakna dan mampu menghasilkan keputusan klasifikasi yang lebih optimum. Oleh itu, kajian tersebut mencadangkan penggunaan RFE yang menggunakan algoritma SVM dalam fasa pemilihan ciri atas data multi-omics yang terdiri daripada genomics, epigenomics dan transcriptomics. Daripada 20 subset ciri yang telah dipilih oleh SVM-RFE, 5 subset ciri yang paling kecil telah digunakan sebagai input kepada dua algoritma autoencoder, iaitu Stacked Denoising Autoencoder (SDAE) dan Variational Autoencoder (VAE) untuk tujuan klasifikasi. Keputusan klasifikasi menunjukkan bahawa model VAE dapat mencapai skor AUC sebanyak 1.0 untuk semua subset ciri, manakala model SDAE hanya dapat mencapai 1.0 skor AUC bagi Feature Subset 1000 (FS 1000) dan Feature Subset 4000 (FS 4000). Kedua-dua model SDAE dan VAE dapat mengklasifikasikan semua sampel dalam set test bagi FS 1000 dan 4000. Kesimpulan yang diperoleh daripada kajian tersebut ialah FS 1000 dan 4000 merupakan dua subset ciri yang paling optimum yang dipilih oleh algoritma SVM-RFE.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AE	-	Autoencoder
ANN	-	Artificial Neural Network
AUC	-	Area under curve
AUC ROC	-	Area under the ROC curve
BCE	-	Binary Cross Entropy
CNN	-	Convolutional Neural Network
DAE	-	Denoising Autoencoder
DBN	-	Deep Belief Network
DL	-	Deep Learning
DNN	-	Deep Neural Network
FN	-	False Negative
FP	-	False Positive
FS	-	Feature Subset
KL	-	Kullback-Leibler divergence
NaN	-	Not a Number
NN	-	Neural Network
ReLU	-	Rectified Linear Unit
RFE	-	Recursive Feature Elimination
RNN	-	Recurrent Neural Network
RO	-	Research Objective
ROC	-	Receiver Operating Characteristic
RQ	-	Research Question
SDAE	-	Stacked Denoising Autoencoder
SMOTE	-	Synthetic Minority Oversampling Technique
SVM	-	Support Vector Machine
SVM-RFE	-	Support Vector Machine – Recursive Feature Elimination
TN	-	True Negative
TP	-	True Positive
UTM	-	Universiti Teknologi Malaysia
VAE	-	Variational Autoencoder

VT - Variance Threshold

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Lung cancer is one of the deadliest cancers in the world. It accounts for 11.6% of total cancer cases in 2018 (Gao, Zhou, & Lyu, 2020), and is expected to cause an approximated annual death of 1.8 million (Xu et al., 2020), making it the leading cause of cancer death worldwide. Despite the fact that the evolution of technology has come so far into reality, the prognosis of lung cancer is still less optimistic even with the advanced treatments. On the other hand, the recent development of artificial intelligence technology in the multi-omics principle field, the analysis on multi-omics using machine learning and deep learning can bring improvement to the prognosis of lung cancer (Gao et al., 2020).

Big data has become a term which has been heard in an increasingly more frequent manner along with another term called "IR 4.0". According to Diebold in his journal article published in 2000, big data can be defined as data which are acquired from the explosion in the quantity and quality of recent information, where advanced data recording and storage technologies has to be implemented to accommodate them. has massive contributions in healthcare sector, especially in the diagnosis of cancer.

Going by that definition, we can say that biological data could also be a kind of big data. In biology, there is one field that goes by the term "omics". Omics can be defined as several field of studies in life sciences that focus on large amount of information to understand life (S. P. Yadav, 2007). There are several omics in biology, namely genomics, epigenomics, transcriptomics, proteomics, metabolomics and so on. When the knowledge of two or more of these disciplinary fields are applied in a study, the word "multi-omics" is termed.

Multi-omics allowed scientists to acquire more insights about biological data by combining multiple omics fields into a single big picture. By integrating multiple omics disciplinary fields together, scientists are allowed to study biological phenomenon in a more holistic way, which subsequently improved the prognosis and predictive accuracy of disease phenotype. With the improvement in prognosis of diseases, a better treatment and prevention can be facilitated. (Subramanian, Verma, Kumar, Jere, & Anamika, 2020)

1.2 Problem Background

Biological data are growing exponentially every year. Along with that, lately, the emergence of the multi-omics disciplinary field and its increasing popularity has made more scientists to work in this field, as it allows them to study more complex mechanisms across molecular layers (Ulfenborg, 2019). Despite the benefits of multi-omics, the way of implementing it is very challenging. For instance, multi-omics integration is very difficult as the analytical tools and experimental designs which are carefully developed are mostly specific to each individual omics discipline. This in turn does not permit appropriate comparison or intelligent integration across multiple omics discipline (Pinu et al., 2019).

Researchers performing data analysis in multi-omics have to address the three major challenges, which are: 1) large p small n problem or the curse of dimensionality, 2) sampling bias and noise present in each omics data with different scale, and 3) effectively performing multi-omics integration using the chosen omics types to provide insightful information (El-Manzalawy, Hsieh, Shivakumar, Kim, & Honavar, 2018).

Multi-omics data analysis belongs to one of the analysis involving the use of datasets with properties of small number (n) of samples with large number (p) of features, or more commonly known as "large p small n problem" (Huynh, Nguyen, & Do, 2020). Due to the nature of multi-omics data analysis requiring multiple observations from each omics types (i.e. the observation from the same individual or

patient has to be present in each of the omics types being studied), it is usually very difficult to acquire a large sample size. This in turn limits multi-omics analysis to be performed with small sample size (Taguchi & Turki, 2022). Besides handling different omics data with different scale and bias, it is almost challenging to extract complementary information from the omics data without the use of proper data integration method (El-Manzalawy et al., 2018).

In this research, feature selection is the main focus. Due to the very high number of features, insufficient or inappropriate dimensional/feature reduction can cause many machine learning and deep learning methods to fail at producing the optimal result. In addition, due to the same reason, many data scientists are struggling with the dimensional reduction step as they are unable to determine the significant/important features that should be kept, or those irrelevant features that should be disposed to produce an optimal result with their model. Therefore, the study aims to overcome the curse of dimensionality in multi-omics data by incorporating SVM-RFE as the feature selection algorithm.

1.3 Research Aim

To employ the feature selection of multi-omics data by selecting the highest rank of features that contribute to promising relationship with the output class, using SVM-RFE.

1.4 Research Question

The questions of the research are:

RQ1 - How does the algorithm for SVM-RFE, SDAE and VAE work?

RQ2 - What are the appropriate parameters for SVM-RFE, SDAE and VAE? Is there any specific fine-tuning method that can help improve the model

performance?

RQ3 - How to determine the quality of the selected features by SVM-RFE?

1.5 Research Objectives

The objectives of the research are:

RO1 - To study and understand the algorithm of SVM-RFE, SDAE and VAE models.

RO2 - To determine the suitable parameters to be used in developing the SVM-RFE, SDAE and VAE models and apply appropriate fine-tuning methods to them.

RO3 - To validate and verify the performance of the SVM-RFE feature selection model using SDAE and VAE models based on the selected performance measures.

1.6 Research Scope

Since multi-omics covers multiple omics fields, such as genomics, epigenomics, transcriptomics, proteomics and metabolomics, it will involve an unsightly massive amount of data if we were to take all of them into consideration in this research. Therefore, to reduce the scope of the study, the research only covered three (3) omics, which are genomics (gene expression), epigenomics (DNA methylation) and transcriptomic (miRNA expression).

The dataset for the three omics mentioned above are readily available at http://acgt.cs.tau.ac.il/multi_omic_benchmark/download.html. For performance

measure, confusion matrix is used to obtain measurements such as accuracy, precision, sensitivity and F1 score.

The tools used for this study involves the programming language Python. Besides that, a few handy libraries from Python such as Pandas and Numpy are used for data pre-processing phase. Besides that, ScikitLearn and Keras libraries are also used for feature selection and deep learning model development.

1.7 Research Contribution

This research proposed the use of a method called Support Vector Machine – Recursive Feature Elimination (SVM – RFE) for the feature selection procedure in researches related to multi-omics. An efficient and effective feature selection method will aid in developing any model for cancer classification.

1.8 Report Organization

This section outlines the composition of this report. There is a total of six (6) chapters as follows:

Chapter 1 introduced the big picture of the research. It consists of the introduction, problem background, research aim, research question, research objective, research scope and research contribution.

Chapter 2 comprised of literature review, whereby the domain of the research is discussed thoroughly. The literature review covered the fundamental of lung classification, omics, multi-omics, feature selection using SVM-FRE and several deep learning methods focusing on autoencoder.

Chapter 3 highlighted about the research methodology, where the research framework is discussed. Besides that, the overall flow of the research is also depicted in this section.

Chapter 4 covered the design and implementation of the research, where the proposed solution is discussed in detail. Besides that, the experimental design workflow and the process involved in it are also a part of this chapter. There are four (4) phases in this chapter, which are the pre-processing, multi-omics integration, feature selection and deep learning classification.

Chapter 5 discussed the overall findings of the research. The results obtained are thoroughly analysed. The quality of the selected feature subsets using SVM-RFE is assessed using the SDAE and VAE models developed using the same feature subsets. A comparative analysis between the two deep learning models is also carried out.

Chapter 6 is the conclusion of the research, where the results obtained from the research are summarized. Besides that, the achievement of the project objectives is also revisited along with the suggestions for improvement and future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter aims to provide the research's literature review. It presents the current knowledge related to this research which are obtained from the findings by previous researchers. First of all, the lung cancer classification is discussed, followed by the introduction to omics, which will briefly introduce a few selected omics disciplinary fields and multi-omics. The subsequent subchapter covers the review of machine learning method for feature selection, and will focus on RFE method using SVM, which is belongs to the wrapper method category. The next subchapter is the introduction of deep learning, whereby the autoencoder artificial neural network (ANN) is discussed. The two autoencoder techniques, Stacked Denoising Autoencoder (SDAE) and Variational Autoencoder (VAE) are the main focuses for this subchapter. Finally, the chapter summary is provided at the end of this chapter. The taxonomy for this chapter is provided in Figure 2.1.

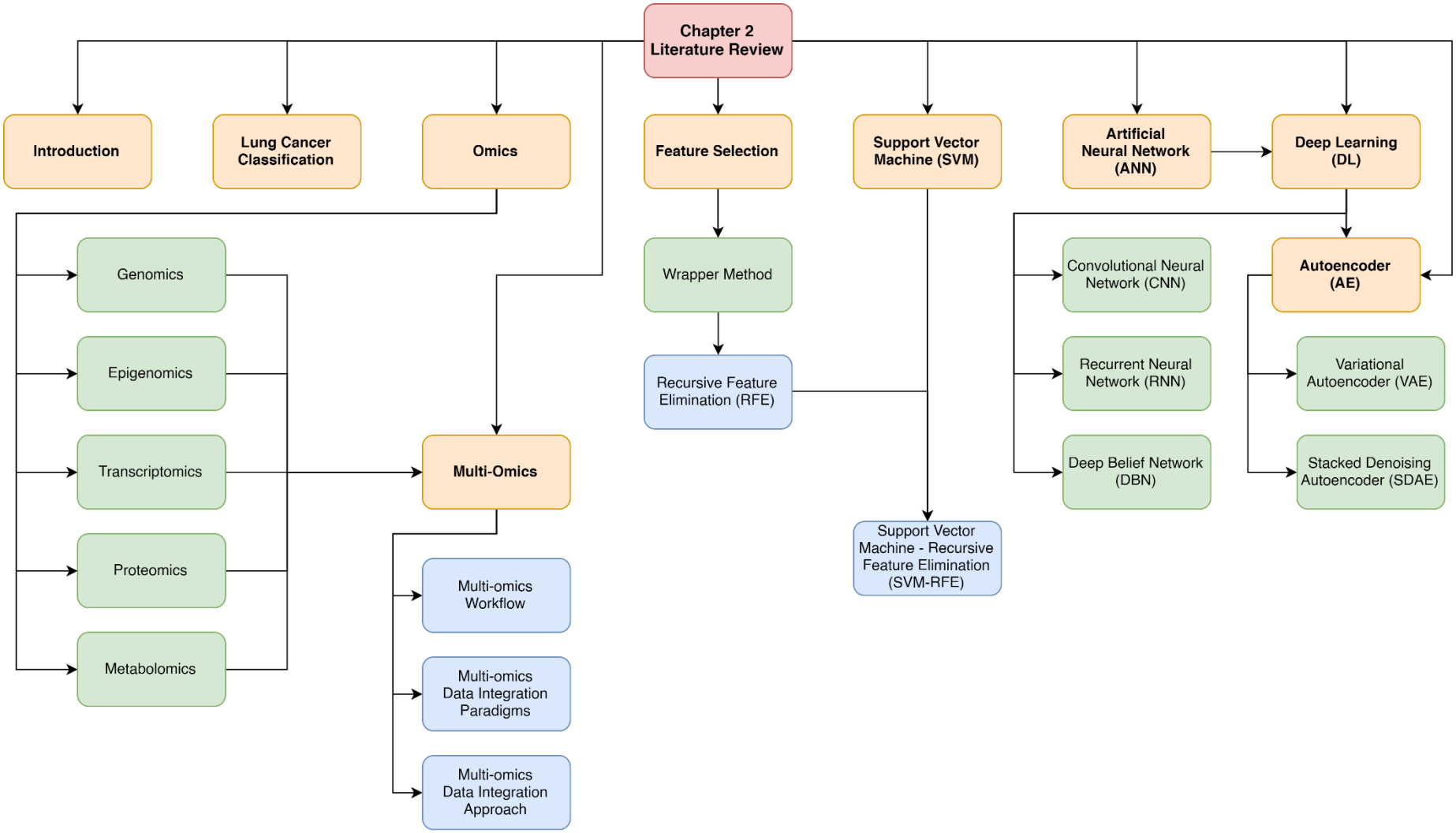


Figure 2.1 Chapter 2 Taxonomy

2.2 Lung Cancer Classification

Lung cancer can be categorized into two categories, which are Non-Small Cell Lung Cancer (NSCLC) and Small Cell Lung Cancer (SCLC). NSCLC alone accounts for approximately 80% of all primary lung cancers. NSCLC can be further divided into three different major histological types, which are adenocarcinoma (AC), squamous cell carcinoma (SCC) and large cell carcinoma (LCC). AC is the most common NSCLC, which now represents the dominant histological subtype of all lung cancers. SCC on the other hand constitutes approximately 33% of the lung cancer cases worldwide, which LCC only accounts for 3% of all lung cancers (Davidson, Gazdar, & Clarke, 2013). In addition, AC stems from small airway epithelia, while SCC stems from large airway epithelia (Chaunzwa et al., 2021).

There are several attempts of application of deep learning into the effort of studying lung cancer, specifically using the convolutional neural network (CNN) architecture with histopathology images as input. Coudray et al. (2018) has developed an InceptionV3 CNN model to classify lung tumor subtypes into adenocarcinoma (AC), squamous cell carcinoma (SCC) and normal lung, and at the same time trained the model to predict the ten most commonly mutated genes in AD. In a more recent study, K. H. Yu et al. (2020) has developed the first fully automated machine learning methods that classifies the NSCLC transcriptomics subtypes, which does not rely on prior knowledge in pathology. Kriegsmann et al. (2020) has also developed multiple CNN models including VGG16, InceptionV3 and InceptionResNetV2 in their study to classify the lung cancer subtypes of AC, SCC and small cell lung cancer (SCLC).

Another recent attempt of developing a deep learning based lung cancer classifier was done by Yang et al. (2021). In their research, the first deep learning model which could classify six types of lung cancer from digital whole slide image (WSI) was developed. The six types of lung cancers are lung adenocarcinoma (AC), lung squamous cell carcinoma (SCC), small cell lung carcinoma (SCLC), pulmonary tuberculosis (PTB), organizing pneumonia (OP), and normal lung. The deep learning classifier or specifically, convolutional neural network (CNN) classifier used was

EfficientNet-B5- and ResNet-50- based models. In their research, EfficientNet-B5-based model was able to outperform ResNet-50, and was chosen as the backbone for their classifier. With this setup, the model was able to achieve an area under the curve (AUC) of 0.970, 0.918, 0.963 and 0.978 respectively for four different cohorts of dataset from different medical centers, averaging at an UAC of 0.9573. This impressive result is undoubtedly one of the most significant research in history, as the result from this research was comparable to the current state-of-art models (Yang et al., 2021).

2.3 Omics

Ever since the successful attempt of mapping and sequencing the human genome in the Human Genome Project (HGP) in year 2003, several new technologies had emerged to allow human to obtain numerous amounts of molecular measurements from a cell or tissue. Subsequently, these advanced technologies could be applied in a certain interested biological system to obtain a snapshot of the underlying biology at a resolution that has never before been possibly obtainable from older technologies. Ultimately, the term 'omics' has been coined to describe the scientific fields that are specifically focused on measuring such biological molecules in a high throughput way (Committee on the Review of Omics-Based Tests for Predicting Patient Outcomes in Clinical, Board on Health Care, Board on Health Sciences, & Institute of, 2012).

There are multiple research areas that are part of omics. For instance, genomics, epigenomics, transciptomics, proteomics and metabolomics, which corresponds to the focused analysis of genes, modified histone proteins in chromosomes, ribonucleic acids (RNA), proteins, and metabolites, respectively. There are multiple motivations behind omics researches which made them so popular even until today. One of the common reasons is to acquire the thorough understanding of the biological system under certain specific and focused study. For example, genomics study could be carried out on several samples consisting of human genetic data to determine their genetic factor which contributes to the

development of common diseases such as high blood pressure. Besides that, omics studies allow scientists to associate clinical outcomes such as response to therapy and risk of breast cancer reoccurrence, to the omics-based molecular measurements, in an effort to develop a more advanced predictive or prognostic model for certain disease or cancer. (Committee on the Review of Omics-Based Tests for Predicting Patient Outcomes in Clinical et al., 2012).

2.3.1 Genomics

Genomics is the first omics discipline, which is a focused study on entire genomes. Genomics is different compared to "genetics" in the sense that "genetics" primarily focus on the study of individual variants or single genes. It had massive contribution in providing scientists a useful framework for mapping and studying specific genetic variants which can cause mendelian and complex diseases. For example, genome-wide association study (GWAS) is an approach which is widely used in multiple human populations to identify thousands of genetic variants associated with complex diseases. In GWAS, millions of genetic markers are genotyped from thousands of individuals. These genetic markers are then used in tests which could determine significant statistical differences in minor allele frequencies between cases, which ultimately served as the evidence of association (Hasin, Seldin, & Lusis, 2017).

2.3.2 Epigenomics

Epigenomics is one of the "omics" approach, which refers to the study of chemical modifications of DNA that play a vital role in the regulation of gene activity and expression (Martin-Sanchez, Lopez-Campos, & Gray, 2014). These chemical modifications involve DNA methylations, histone modifications and their relations to non-coding RNA. Epigenomics gives scientists important insights regarding the mechanisms and functions of gene regulation across many genes or in a cell or organism (Kalavacharla, Subramani, Ayyappan, Dworkin, & Hayford, 2017).

Epigenomic regulation is affected by an individual's environmental factors such as diet and amount of exercises. Therefore, it is a process in the body which is described as highly dynamic. The chemical modifications involved in epigenomics could be inherited, and they do not alter the genomic sequence but instead, it can affect how the body interprets a DNA sequence, such as turning a gene "on" or "off" (Martin-Sanchez et al., 2014).

2.3.3 Transcriptomics

Transcriptomics refers to a comprehensive analysis of whole sets of transcripts for a particular cell, tissue, organ or whole organism (D. Yadav, Tanveer, Malviya, & Yadav, 2018). It encompasses everything related to RNAs, such as their transcription and expression levels, functions, locations, trafficking and degradation, and it covers all types of transcripts such as messenger RNAs (mRNAs), microRNAs (miRNAs) and variants of long non-coding RNAs (lncRNAs) (Milward et al., 2016). Transcriptomics examines RNA qualitatively and quantitatively. Qualitative transcriptomics examinations involves determining the presence of transcripts, identification of novel splice site and RNA editing sites, while quantitative transcriptomics examinations involves determining the amount of expression for each transcript (Hasin et al., 2017). Transciptomics has several contributions in biology. These include the functional annotation of the genome, understanding posttranslational modifications (PTM), relating transcriptomes to diseases and so on (D. Yadav et al., 2018).

2.3.4 Proteomics

Proteomics is the analysis of the entire protein composition of a cell, tissue or organism under a specific and defined set of conditions (L.-R. Yu, Stewart, & Veenstra, 2010). These analyses include the large-scale study of proteins themselves, and their structure and function (Kaczor-Urbanowicz & Wong, 2020). Proteomics is a core technology in understanding molecular mechanisms underlying normal and

disease phenotypes and identifying critical diagnostic and prognostic biomarkers. Besides that, it also assesses the protein abundance and PTMs (Hasin et al., 2017), protein localization, isoforms and molecular interactions (Coorssen, 2013). Proteomics is dependent on three basic technologies, which are (1) a method to fractionate complex protein or peptide mixture, (2) Mass Spectrometry (MS) to gather data necessary for individual protein identification, and (3) bioinformatics to analyze and assemble the information obtained from MS (L.-R. Yu et al., 2010).

2.3.5 Metabolomics

Metabolomics refers to the comprehensive qualitative and quantitative analysis of every small molecule in a system. The small molecules could be some cell samples, body fluids (sweat, saliva etc.), cell tissues or an entire organism (Milne & Morrow, 2009). It is a powerful study which involves obtaining a metabolic profile using analytical technologies and analyzing the data generated by these technologies to acquire insights into the system's metabolic state (Dayalan, Xia, Spicer, Salek, & Roessner, 2019). These insights help scientists in developing patient-specific treatment strategies based on the biomarker discovery and disease subtyping (Uppal, 2021). Metabolomics can be divided into two categories or approaches, which are targeted and untargeted metabolomics. Targeted metabolomics involves determining the quantified intensities of a specific compound of interest in samples (Hoang, Udupa, & Le, 2019). Untargeted metabolomics on the other hand is applied to obtain a list of metabolites present in a sample (Pereira Braga & Adamec, 2019).

2.4 Multi-omics

Multi-omics is an integrated disciplinary field which involves the integration of one or more omics disciplinary field. It allows researchers to have a deeper understanding regarding omics, such as from the original cause of disease (whether they are genetic, environmental or developmental) to the functional consequences or

relevant interactions (Hasin et al., 2017). It provides a more promising approach towards a more detailed molecular understanding of health and disease, which also include the chain of cause and effect, which in turn help doctors to develop novel therapies (Wörheide, Krumsiek, Kastenmüller, & Arnold, 2021).

Single omics (genomics, transcriptomics, proteomics etc.) have proven to be useful in science, but their limitation is that they face difficulty in explaining the complexity of molecular processes (Zhou, Varol, & Efferth, 2021). This is where multi-omics comes into rescue, where it provides a more holistic view to study biological phenomenon, which overcomes the limitation of single omics analyses (Subramanian et al., 2020). Although the adoption of multi-omics approach does not automatically provide the answer to any study or research, it has been proven useful to provide more reliable results and has made the mitigation of the risk of false positive results possible, through the combination of multiple layers of evidence in multi-omics (Wörheide et al., 2021).

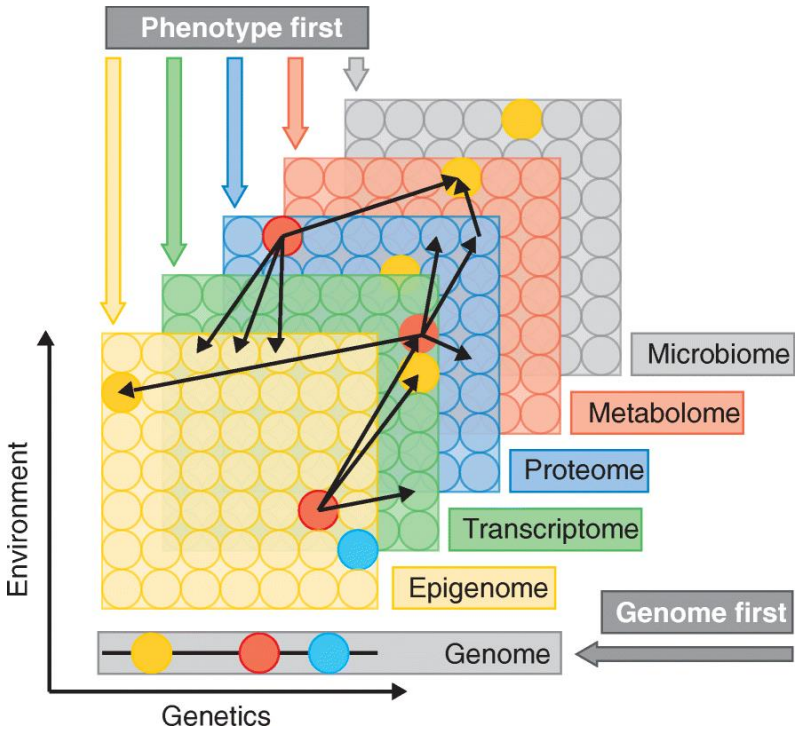


Figure 2.2 Multi-omics data types and approaches to disease research (Hasin et al., 2017)

Figure 2.2 depicts the data types in multi-omics approach. There are several data layers and each of them represents a single omics field, such as genomics, transcriptomics and proteomics. Omics data are collected on the entire pool of molecules represented as circles. Each data layer (except for genome) reflects both genetic regulation and environment which may affect each individual molecule to different extent. Referring to the figure, the thin black arrows from each red circle indicate the potential interactions or correlations present between molecules and different data layers. For instance, the red protein could have certain correlations to multiple methylated DNAs (the red circle in the blue proteomics layer showing several arrows pointing towards the green epigenomics layer). Interactions within each distinct layers are present but not depicted in the figure. (Hasin et al., 2017)

The thicker, coloured arrows show various potential starting points or conceptual frameworks for integrating multiple omics data to study a disease. The genome-first approach indicates that the multi-omics study starts from associated locus, while the phenotype-first approach implies the starting point of studies using any layers besides the genome layer. The environment first approach, which is not shown in the figure, examines environmental perturbations. (Hasin et al., 2017).

2.4.1 Multi-Omics Workflow

Before any multi-omics study and analysis can be performed, the desired omics data of interests have to be integrated by means of concatenating or combining the said omics data using several omics data integration methods. The choice of appropriate integration strategy is not straightforward and is heavily dependent on the available data and a particular study's objective, which is further complicated by the data dimensionality, heterogeneity and the lack of universal protocols to these approaches (Wörheide et al., 2021).

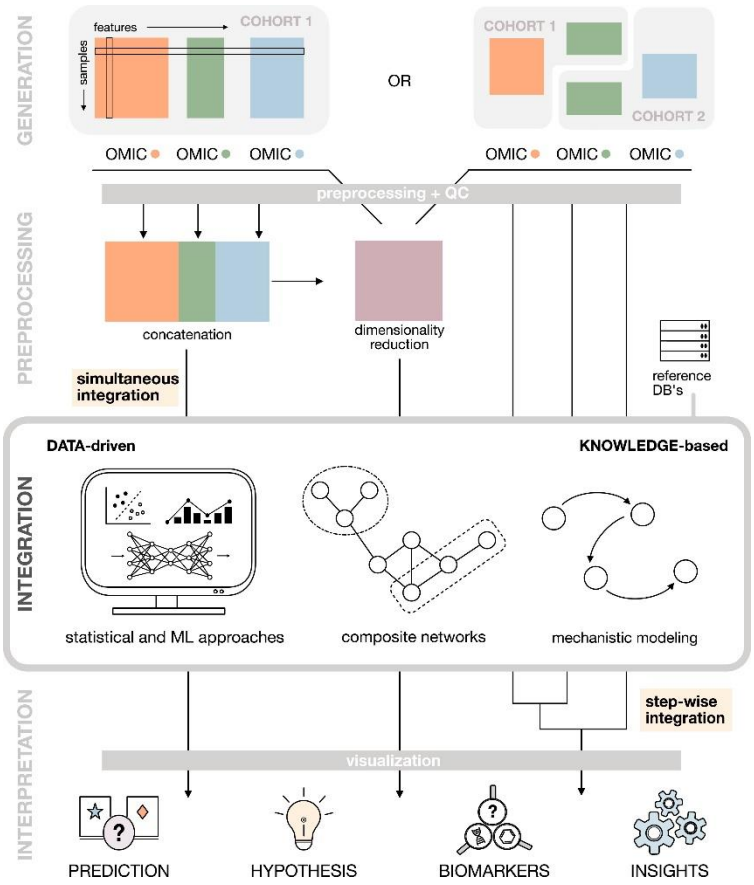


Figure 2.3 Multi-omics workflow (Wörheide et al., 2021)

Figure 2.3 depicts the general idea and workflow for multi-omics researches. The workflow starts with the data generation phase, where the sample preparation is done. Any subsequent data which are required for the studies are also acquired during this stage. Acquired omics data or samples (genomics, transcriptomics, proteomics) can either from the same cohort or different cohorts. In Figure 2.3 under "Generation" section, the figure on the left shows the samples collected from a single cohort. Each omics type is depicted in different colours (orange, green, blue). On the other hand, the figure on the right shows the samples collected from two different cohorts, where the cohort 1 contains the orange and part of the green omics, while the cohort 2 contains part of the green omics and blue omics.

The process continues with data preprocessing phase, where various kind of data cleaning processes are done. Ultimately, the cleaned datasets are concatenated

into one single large dataset. The subsequent step is to perform dimensionality reduction on this single large dataset.

The next phase is the multi-omics data integration step. There are several data integration techniques to be used, such as statistical and Machine Learning (ML) approaches, composite network and mechanistic modelling. In multi-omics data integration phase, there are two different strategies, which are the data-driven and knowledge-based approaches.

In the last phase of the multi-omics workflow is the data interpretation, which the results obtained are further applied in various predictions, forming hypothesis, identifying biomarkers and gaining insights regarding a particular topic.

2.4.2 Multi-Omics Data Integration Paradigms

According to Wörheide et al. (2021), there are two major data integration paradigms in general, namely the simultaneous integration and step-wise integration.

In simultaneous integration, all omics data which are collected in the data generation step are used at the same time to perform analysis in a single modelling step. In this approach, all the information in each omics layers are complementary to one another, and the correlation between these layers are considered. However, the restriction for simultaneous integration is the requirement to use only the data acquired from the same biological samples or individuals (Wörheide et al., 2021).

In step-wise integration, omics datasets are analyzed in isolations, or in other words, the integration is performed in a subsequent step, with a specific combination from different omics data type. The advantage for step-wise integration is that it facilitates the integration of data and statistical results from different sources, such as from different studies or knowledge bases. This subsequently allows the large scale analysis of heterogeneous data in the absence of omics measurements for the same samples. (Wörheide et al., 2021)

In this study, simultaneous integration is used since the omics datasets obtained are from the similar patients.

2.4.3 Multi-Omics Data Integration Approach

There are several data integration approaches for multi-omics dataset. In 2017, (Lin & Lane, 2017) have surveyed the Machine Learning and System Genomics (MLSG) software framework and stated that there are three integration approaches, which are (1) Model-based Integration (MBI), (2) Concatenation-based Integration (CBI), and (3) Transformation-based Integration (TBI).

2.4.3.1 Model-based Integration (MBI)

In MBI approach, several models will be generated from each multi-omics data type (e.g. genomics, epigenomics, transcriptomics), which acts as the training set in the training phase. Among these generated models, a final model is generated by combining the generated models (Lin & Lane, 2017). One of the advantages of MBI approach includes the ability to combine predictive models from different multi-omics data types, in which these data types can be collected from different set of patients with the same phenotype. Figure 2.4 shows the flowchart of MBI approach.

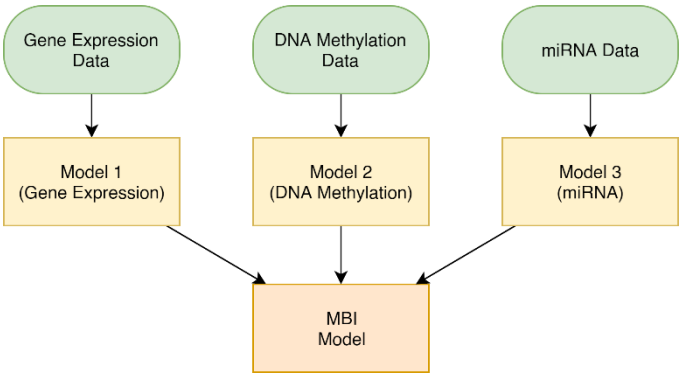


Figure 2.4 Model-based Integration (MBI) framework based on Lin and Lane (2017)

2.4.3.2 Concatenation-based Integration (CBI)

CBI approach is the simplest multi-omics integration method. It involves combining the multi-omics datasets (e.g. genomics, epigenomics, transcriptomics) into a large matrix. This large matrix of input will then be used to construct models for classification purposes (Lin & Lane, 2017). In CBI approach, the number of samples remains the same while the number of features increases depending on the number and type of multi-omics data types (Picard, Scott-Boyer, Bodein, Périn, & Droit, 2021).

One of the advantages of CBI approach is its simplicity and ease of implementation. It is relatively simple to apply various machine learning methods to the combined large matrix of input for continuous or categorical data analyses (Picard et al., 2021).

However, the drawbacks of CBI approach are fairly obvious, such as producing a large matrix with more complexity and noise which makes the learning process by machine learning models more challenging. Besides that, the learning model also tends to focus on learning the features on the omics type with more features instead of learning the entire input as a whole (Picard et al., 2021).

In this study, the CBI approach is used due to its simplicity and ease of implementation nature. Figure 2.5 depicts the framework of the CBI approach.

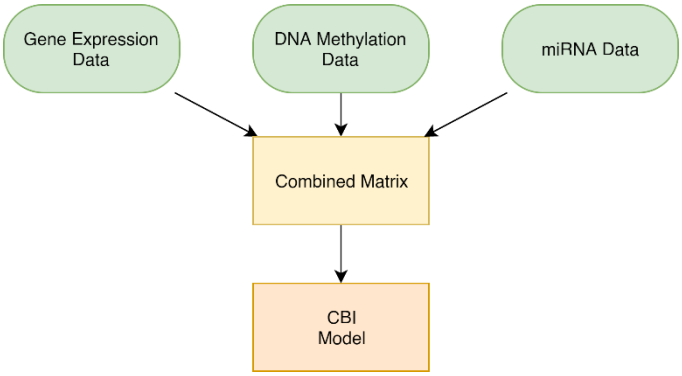


Figure 2.5 Concatenation-based Integration (CBI) framework based on Lin and Lane (2017)

2.4.3.3 Transformation-based Integration (TBI)

In TBI approach, the multi-omics data types are transformed into several individual intermediate forms. These intermediate forms could be in form of graphs or kernels. The intermediate forms are then combined into a final intermediate form in form of a large matrix. The final intermediate form is then used for model building (Lin & Lane, 2017).

The advantage of the TBI approach includes its ability to integrate data which contain unique variables such as the ID of a patient for multi-omics data types linking (Lin & Lane, 2017). Figure 2.6 shows the framework of the TBI approach.

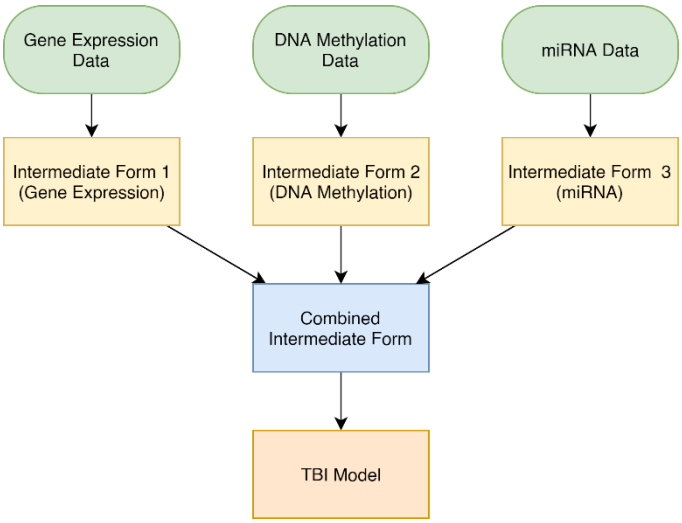


Figure 2.6 Transformation-based Integration (TBI) framework based on Lin and Lane (2017)

2.5 Feature Selection

Feature selection is one of the phases involved in any research involving moderate to large number of input variables or features (also known as attributes, fields or columns). The idea behind feature selection is fairly simple, where it involves the process of selecting a subset of input variables from a large pool of attributes, by the process of eliminating features with little to no predictive information to the desired output (also known as outcome) class (Samb et al., 2012).

The term feature selection can be confusing sometimes, as it has several interchangeable synonyms, but they are quite intuitive to tell as they ultimately refer to the same thing which is related to the features of the dataset. These synonyms includes variable selection, feature reduction, attribute selection and variable subset selection, just to name a few (Samb et al., 2012).

There are numerous benefits from applying feature selection in one's study. These benefits involve facilitating data visualization and data comprehensibility, reduction in storage requirement, reduction in model training time, escaping from the

curse of dimensionality (Guyon & Elisseeff, 2003), having better predictive performance and higher computational efficiency from working with fewer inputs (Samb et al., 2012).

In biology or healthcare informatics, feature selection often involves prior knowledge or biological hypotheses to help guide the model to appropriately select a subset from the biological dataset such as omics datasets (Wörheide et al., 2021), which is thought to be biologically relevant based on the said knowledge (Committee on the Review of Omics-Based Tests for Predicting Patient Outcomes in Clinical et al., 2012). For instance, there are several popular approaches of biological feature selections such as limiting the analyses to either genes, proteins or metabolites of interest, or selectively investigate only the entities which have certain associated traits in previous studies (Wörheide et al., 2021). Biological data such as omics dataset usually have only somewhere between a hundred to less than a thousand samples (or rows) but with thousands of input variables. It is even more complex when it comes to multi-omics analyses, where multiple single-omics datasets are combined. In such cases, it is not even exaggerating to say that these kinds of data consist over a ten thousand input variables. In such cases, effective and low-cost feature selection algorithms have to be developed to facilitate these researches.

There are three types of basic feature selection methods, namely the filter methods, wrapper methods and embedded methods as shown in Figure 2.7.

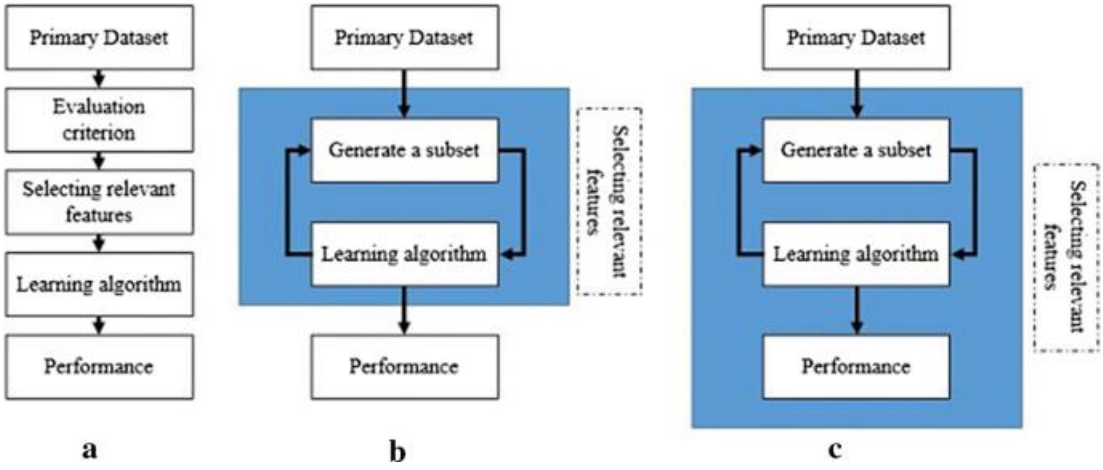


Figure 2.7 An overview of the three basic feature selection methods, which are the filter method (a), wrapper method (b) and the embedded method (c) (Tadist, Najah, Nikolov, Mrabti, & Zahi, 2019)

In filter method, the filter model only considers the intrinsic information of the data (Saeys, Inza, & Larranaga, 2007). It selects a subset of input variables as pre-processing step independently of a classifying algorithm (Guyon & Elisseeff, 2003). An evaluator criterion or a scoring method is usually deployed to assess the degree of relevance of each input variables to the output class or target variable (Tadist et al., 2019). Filter method is advantageous at selecting relevant input variables as it can easily scale from low to high dimensional datasets. Besides that, it does not involve any classifying algorithm, where the wrapper and embedded methods both does. Due to the absence of classifying algorithm in filter method, it is very computationally cheap, therefore making it a fast method (Saeys et al., 2007). The downsides of filter method are directly related to the drawback from its advantages. These disadvantages include the absence of interaction between the data and the classifier, which subsequently ignores the feature dependencies in the dataset (Saeys et al., 2007).

The next feature selection method is the wrapper method. In wrapper method, there exists a feature subset selection algorithm which wraps around the classifier algorithm (also known as induction algorithm). The feature subset selection algorithm is responsible for generating a good subset, and incorporate the use of the said classifier algorithm as the function for feature subset evaluation (Kohavi & John, 1997). The evaluation of the goodness of the generated subset is assessed by

the accuracy of a predictive model generated by the classifier (Tadist et al., 2019). The "wrapped" classifying algorithm acts as a black box, where it is run on the dataset. In each iteration, the dataset is usually partitioned into internal training and holdout sets (Kohavi & John, 1997). Only after the final optimal feature set is selected from several iterations of the feature subset selection algorithm, it is fed to the "external" classifying algorithm as shown in Figure 2.8. Due to the exponential growth of the feature subsets with increasing number of features, heuristic search methods are incorporated for the optimal subset generation (Saeys et al., 2007). Besides all the benefits of using the wrapper method, it is prone to high risks of overfitting and is the most computationally intensive method among the three basic feature selection methods (Saeys et al., 2007).

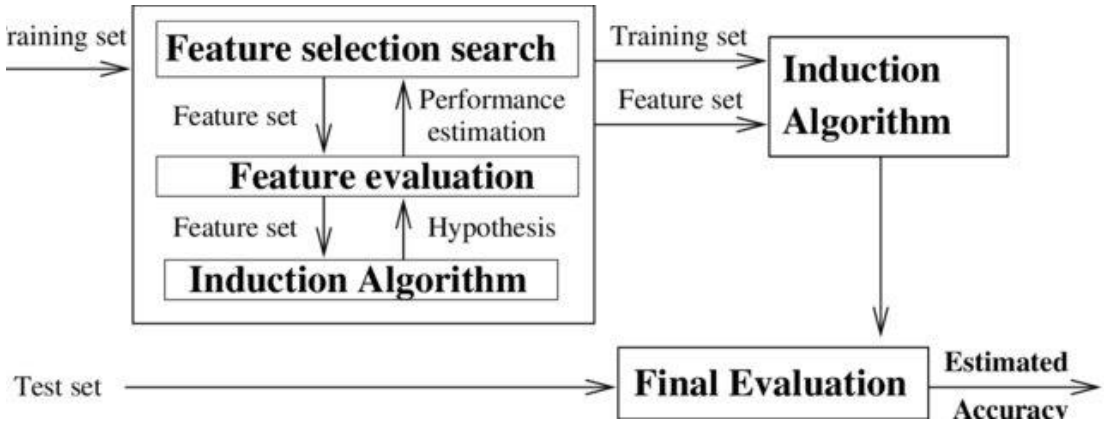


Figure 2.8 The wrapper approach for feature selection (Kohavi & John, 1997)

The last method out of the three basic type of feature selection methods is the embedded method. Embedded method is different from the filter and wrapper feature selection methods specifically in the way of the interaction between the feature selection algorithm and the classification algorithm. In embedded method, the feature selection part and the machine learning part is inseparable, in contrast to the wrapper method where these two processes are distinct parts (Lal, Chapelle, Weston, & Elisseeff, 2006). In embedded method, available data can be made use in a more efficient way as it does not necessarily have to be split into the training dataset and validation dataset. This allows embedded methods to obtain outputs quickly without having to retrain a predictor from scratch for every variable subset in the iterations (Guyon & Elisseeff, 2003). Embedded method is a hybrid between filter and wrapper

methods. Therefore, it inherits their strengths. For example, embedded method overcomes the weakness in filter method where it allows the interaction between the data and the classifying algorithm (Saeys et al., 2007). Besides that, embedded method is significantly less computationally intensive (therefore faster) compared to wrapper method (Lal et al., 2006; Saeys et al., 2007), while also overcoming the risks of overfitting (Tadist et al., 2019).

To sum up this section, every feature selection method has their own advantages and downsides. A quick recap for the three feature selection methods is summarized in Table 2.1.

Table 2.1 Summary of three feature selection methods with their advantages and disadvantages

Feature Selection Method	**Advantages**	**Disadvantages**
Filter	1. Easily scalable to high dimensional datasets 2. Computationally simple 3. Fast and independent of the classifying algorithm	1. No interaction between data and the classifying algorithm 2. Ignores feature dependencies 3. Least effective among the three basic methods
Wrapper	1. Involvement of a classifying algorithm 2. Taking feature dependencies into account 3. Highest effectiveness among the three basic methods	1. Risk of overfitting 2. Computationally costly
Embedded	1. Combines the advantages of filter and wrapper methods 2. Less computationally costly and chances of overfitting than wrapper method	1. Less effective than wrapper method

In this study, wrapper method will be the focus. There are three main types of feature selection methods in wrapper method, namely the forward selection, backward elimination, and the stepwise selection.

Forward selection approach starts with an empty set of variables. In each step, one variable which decrease the error of the classifier the most is added into the set of variables. This process continues until the accuracy of the classifier does not improve significantly (i.e. the subsequently added features does not significantly reduce the error of the classifier (Samb et al., 2012).

Backward elimination is the opposite of the forward selection approach. It starts with a set of all input variables and the variables are removed or excluded one by one in each step. At each step, a variable which contribute the least to the decrease of error of the classifier is removed, and the process continues until any subsequent removal of variable no longer produce a better result than before (Samb et al., 2012).

Stepwise selection approach is the hybrid of both forward selection and backward elimination approaches. Due to this, it is also called the bidirectional elimination approach. At each step, features are added into the set of variables according to their significance to the output class, and at the same time, the already existing variables in the set of selected variables are checked to detect any insignificant variables and eliminates them (Vikashrajluhaniwal, 2020).

Recursive Feature Elimination (RFE) is one of the instances under backward elimination approach. It is a wrapper method which uses backward elimination approach to obtain the ranking of features. Similar to backward elimination, a classifier is necessary to be used in RFE. In contrary, in RFE, a specific number of features to be kept (or the size of subset) is defined by the user, which is distinct from forward selection and backward elimination approaches where the final number of features to be obtained is not specified.

By default, RFE is done by removing only one feature in each iteration. However, due to computational complexity problems, it is more reasonable and efficient to eliminate more than one feature in each iteration. Although this could potential cause classification performance degradation issue, it has significantly reduced the computational intensity of the algorithm (Guyon, Weston, Barnhill, & Vapnik, 2002). Considering the case of feature selection in cancer classification using multi-omics data, it will be too computationally intensive to remove one feature at a time given multi-omics datasets contain ten thousand of features.

RFE is initially designed for two-class problems, where the output of the classifier only involves binary classes such as whether a patient has lung cancer or not. However, a multi-class version of RFE can be easily designed using a one-for-the-rest approach (Lal et al., 2006).

2.6 Support Vector Machine – Recursive Feature Elimination (SVM-RFE)

Machine learning has been gaining its popularity recently. Machine learning can be classified into three major categories, which are supervised learning, unsupervised learning and reinforcement learning. Besides that, a subfield of machine learning known as deep learning is also one of the most popular study field recently, which will be discussed later in the next section. Supervised learning works on labelled data (known output class) while unsupervised learning works on unlabeled data (unknown output class). Under supervised learning, it can be further divided into classification and regression, while unsupervised learning is used for clustering. There are many types of classification algorithm, such as k-Nearest Neighbour, decision tree, naïve Bayes method and many more, including Support Vector Machine (SVM).

SVM is one of the popular supervised machine learning method used for classification. It belongs to the generalized linear classifier family, with the objective to simultaneously minimize the empirical classification error and maximize the geometric margin. Due to the way it works, SVM is also known as the maximum

margin classifier (Samb et al., 2012). It is a powerful method to analyze data with a number of features almost equal to or greater than the number of instances in a dataset (Sanz, Valim, Vegas, Oller, & Reverter, 2018). The dataset with n samples and p attributes used for the linear SVM is defined as

$$D = \{(x_i, c_i) \mid x_i \epsilon R^p , c_i \epsilon \{-1, +1\}, i = 1, \dots n\}, \quad (2.1)$$

where c_i is either +1 or –1, depending on which class the instance belongs.

Fundamentally, the SVM's approach is to construct a hyperplane,

$$w \cdot x + b = 0, \quad (2.2)$$

which maximizes the distance between itself and the nearest instances in each output class. The hyperplane is also known as the decision boundary. The equation of the planes representing the two classes are defined as

$$x_i \cdot w + b \geq +1 - \varepsilon_i \quad for\ c_i = +1 \quad (2.3)$$

$$x_i \cdot w + b \leq -1 - \varepsilon_i \quad for\ c_i = -1 \quad (2.4)$$

SVM can also exist in the form of non-linear version. Non-linear SVM is deployed for datasets which cannot be linearly separated into two distinct classes. A conversion function is used in linear SVM to convert the original 2-dimensional data into a new high-dimensional feature space for linear separable problems. This technique to perform non-linear classification in SVM is known as kernel trick (Huang, Hung, Lee, Li, & Jiang, 2014). Figure 2.9 depicts the difference between linear and non-linear SVM.

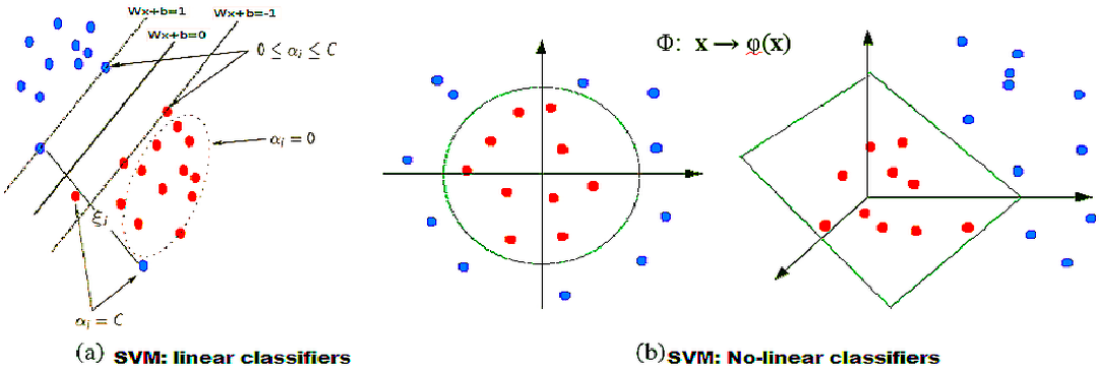


Figure 2.9 Linear (left) vs Non-linear SVM (right) (Samb et al., 2012)

Over the past few years, there have been several feature selection strategies deployed on studies related to gene selection. Perez-Riverol, Kuhn, Vizcaino, Hitz, and Audain (2017) has implemented both Correlation Matrix (CM) and Principle Component Analysis (PCA) in their study for feature selection. CM is said to be efficient at removing feature redundancy, capable of keeping original features and at the same time very easy to implement (Perez-Riverol et al., 2017). PCA on the other hand is able to reduce the dimensionality of the data by representing the original data using fewer new variables (Perez-Riverol et al., 2017). However, these filter methods have their caveats which have been discussed in Chapter 2.5. Regardless of that, according to Perez-Riverol et al. (2017) also, these filter methods can be used in early stage of researches but wrapper method should be used to decide the optimal feature subsets as the last step of feature selection, which leads to the next discussion of Support Vector machine – Recursive Feature Elimination (SVM – RFE).

SVM-RFE is one of the wrapper feature selection method which utilizes backward feature elimination approach to obtain the ranking of features. When used together with SVM, another characteristic that differentiates RFE with the regular backward elimination is that the most significant predictors (top ranked features which are disposed last) are a subset of feature which is the most relevant to the model instead of their individual significance (Sanz et al., 2018).

RFE is initially proposed by Guyon et al. (2002) in their study of gene selection for cancer classification using SVM. In their cancer classification study, the idea of SVM – RFE is to eliminate yield better and more compact gene subsets by removing redundant genes from the dataset. It has proven to show good performance in gene selection for microarray data, where it managed to remove half of the features from the feature subset in each iteration when the dataset contains over thousands of features (Guyon et al., 2002).

SVM –RFE works in four simple steps, which are (1) Training a SVM classifier using the training dataset, (2) Ranking the features using their weights obtained from the classification result, (3) Eliminating features with the smallest weight (i.e. contribute the least to the model accuracy), and (4) Repeating step 1 to 3 with the remaining variables using the same training dataset (Samb et al., 2012).

After having the size of the final subset of features specified, SVM – RFE performs the feature selection using a greedy approach, where the subset of features obtained after eliminating one or more least contributing features in each iteration are trained iteratively. The objective of the algorithm is to remove features which decrease the margin the least until the final feature subset converges to the specified size.

According to Lal et al. (2006), RFE can be easily generalized to other domains such as different classification algorithm or regression, instead of just SVM. In a study by Zhu and Hastie (2004), it is shown that RFE can be adapted to penalized logistic regression and still give the similar performance when compared to SVM. In addition, this model gives superior performance in comparison to SVM-RFE as not only it excels in univariate correlation score feature selecting, but also able to output the probability of correctness instead of than just prediction.

Due to SVM-RFE's greedy nature when used in feature selection, it does not necessarily return the most optimal result. A study done by Samb et al. (2012) involves the proposed solution to counter the limitation of SVM-RFE by incorporating local search. Their algorithm involves two steps, which are (1)

Obtaining the initial solution (final variable subset) from the application of SVM-RFE and (2) Introducing local search procedure to improve the performance of the initial result. Ultimately, their study had justified that SVM-RFE does not necessarily produce the most optimal result due to its greedy approach, when the proposed method outperformed it.

2.7 Artificial Neural Network (ANN)

Artificial Neural Network (ANN) is one of the popular fields in machine learning. It is inspired by one of the most sophisticated organs in humans and animals, which is the brain. ANN works by mimicking the way human brain works. It consists of an interconnected group of nodes (Mhatre, Siddiqui, Dongre, & Thakur), similar to how the neurons in brain are interconnected with one another, forming a huge mesh of neural network.

ANN is widely applied in researches due to their ability to model highly non-linear systems (Dinesh, Ashiq, & Joselin, 2018), in which traditional machine learning methods cannot achieve. Some remarkable applications of ANN include pattern recognition, data classification (Mhatre et al.) and data mining such as forensic and economics (Sharma, 2017).

ANN's architecture consists of three layers, which are the input layer, hidden layer and the output layer. A neural network can have more than just three layers, but in that case, it becomes more complicated and eventually, a new subfield of neural network called "deep learning" is produced (further discussed in the next section). Figure 2.10 illustrates the architecture of an ANN model. It consists of a number of nodes in each layer, which are then interconnected to one another across layers through directed graph manner. Due to the characteristic of directed acyclic graph (Weng, 2017), information in ANN is passed from one layer to another in one direction, i.e. from input layer to output layer.

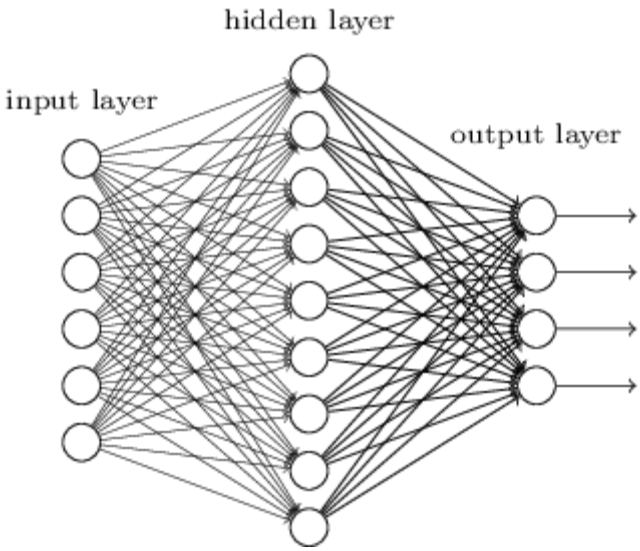


Figure 2.10 Typical architecture of an ANN model containing only 1 hidden layer (Nielsen, 2019)

Each node resembles the neurons in nervous system. They receive inputs from several other neurons, and each of them have their own assigned weight. The inputs are multiplied by these weights and added together, and the sum is then passed to one or more nodes (Dickson, 2019). There are three layers in ANN as mentioned before. Each of them plays a distinct role in the architecture.

The input layer serves the purpose of receiving the values from each explanatory variable for each instance. There are usually equal number of input nodes to the number of explanatory features. These input nodes can communicate with one or more hidden or output nodes. Input nodes do not manipulate the data they receive, hence also known as passive nodes. The way input nodes work is that they duplicate the values they received as input and send them to all the hidden nodes (Sharma, 2017).

The hidden layer is where all the operations are performed. Each of the hidden node are given certain transformations and assigned weights, which are then applied on the input values. Each hidden node have inbound edges either from input nodes or other hidden nodes, and outbound edges either to other hidden nodes or output nodes. Every time an input is given to a hidden node, the input will be

multiplied by the weight, and the weighted inputs will be added together to produce a single number (Sharma, 2017).

The output layer is the last layer in the ANN architecture. They receive inputs either directly from input nodes, or from the hidden nodes. The output that n output node produces corresponds to the prediction of the response attribute. Depending on the number of output classes determined by the researcher in their study, an ANN model will have different number of output nodes, and is usually equal to the number of output class (Sharma, 2017).

2.8 Deep Learning (DL)

Deep learning is a subfield under neural network (NN), When a NN consists of more than one hidden layer, it is considered as a deep neural network (DNN). In simple words, deep learning models are large and deep ANNs. Deep learning is one of the most popular field in artificial intelligence. Its increasingly high popularity is driven by two simple facts, (1) In this modern era, we are dealing with big data and (2) technologies have become so advance to the point that we have significantly more powerful computers compared to years ago (Weng, 2017).

Figure 2.11 illustrates the relationship between each model's performance and the size of data. It can be seen that traditional machine learning algorithms and statistically learnings can no longer keep up with the growing data size. Small and medium NN still perform better than those methods but it is still barely near satisfactory level. This is where large NN such as DNN comes into play to produce the highest performance among all the models.

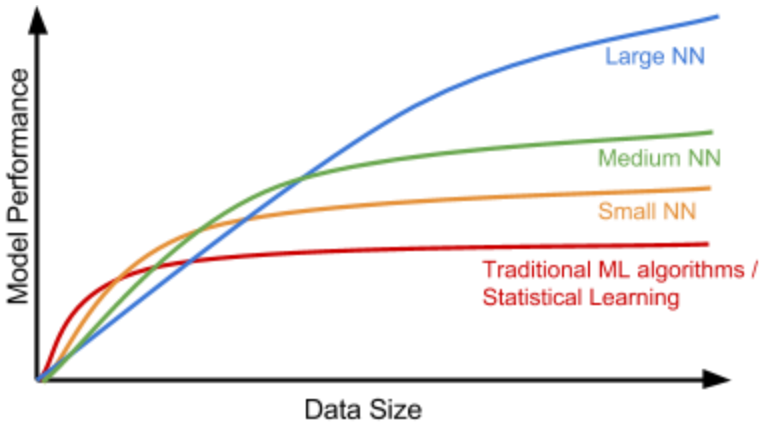


Figure 2.11 The relationship between the size of data and the performance of different models (Weng, 2017)

A DNN is essentially an ANN, except with more than just three layers. In DNN, the model still retains the input and output layers. The only difference between ANN and DNN is the number of hidden layers in between of the input and output layers, which is also referred as the depth of the DNN model. A DNN model can be defined as illustrated in Figure 2.12.

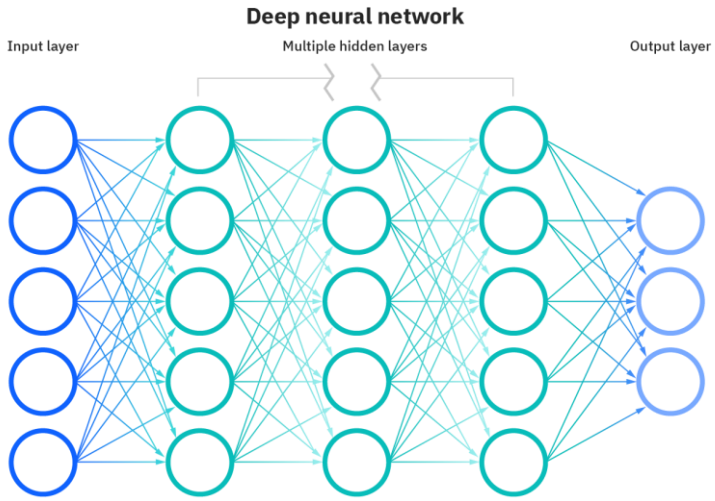


Figure 2.12 Deep Neural Network (DNN) Architecture (Kavlakoglu, 2020)

Most DNN are feed-forward, in which the inputs flow from input nodes to output nodes in only one direct. However, backpropagation can also be implemented in DNN models. Backpropagation is a technique used in deep learning to allow the

inputs to flow in the opposite direction as compared to feed-forward manner, which is from the output nodes back to the input nodes. DNN models with backpropagation implemented has several advantages, which include allowing the model to calculate and attribute the error associated with each node (neuron). This further allowed us to make adjustment to fir the model in a more appropriate manner (Kavlakoglu, 2020).

There are several types of deep learning models. Some of the most basic ones include Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Deep Belief Network (DBN) and AutoEncoder (AE), just to name a few.

2.8.1 Convolutional Neural Network (CNN)

A CNN is a special kind of feedforward NN which consists of several different hidden layers, including the convolution layer, ReLU layer and pooling layer. Each convolution layer and pooling layer in CNN comes in pair, which is illustrated in Figure 2.13 (Emmert-Streib, Yang, Feng, Tripathi, & Dehmer, 2020).

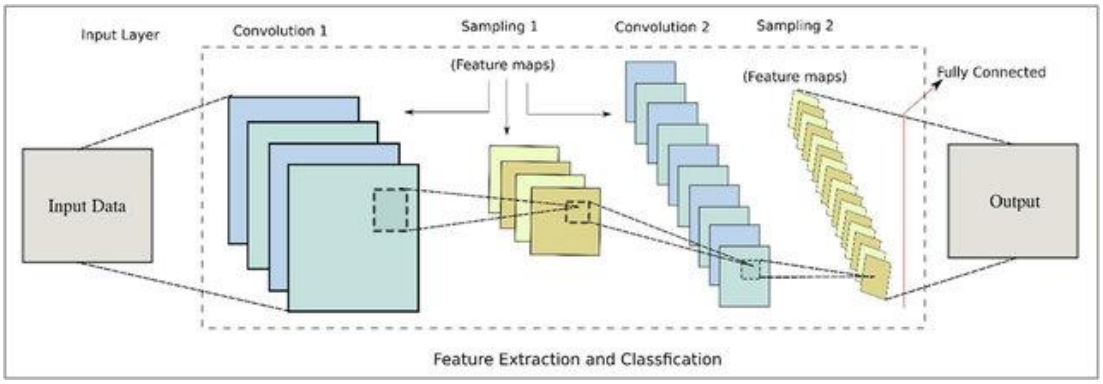


Figure 2.13 Typical Model of a CNN, showing each convolution and pooling layers come in pair (Jan et al., 2017)

In a typical ANN, each neuron (node) in a layer is usually connected to every single node in the next layer, which is also known as fully connected layer. Each of these connections are parameters in the network. With such huge number of parameters, it can be difficult to make appropriate adjustment to the model. In CNN,

a neuron is only connected to its nearby nodes in the next layer. This connection technique is known as local connectivity as shown in Figure 2.14. With this technique, CNN has greatly reduced the number of parameters in the deep learning model (Emmert-Streib et al., 2020).

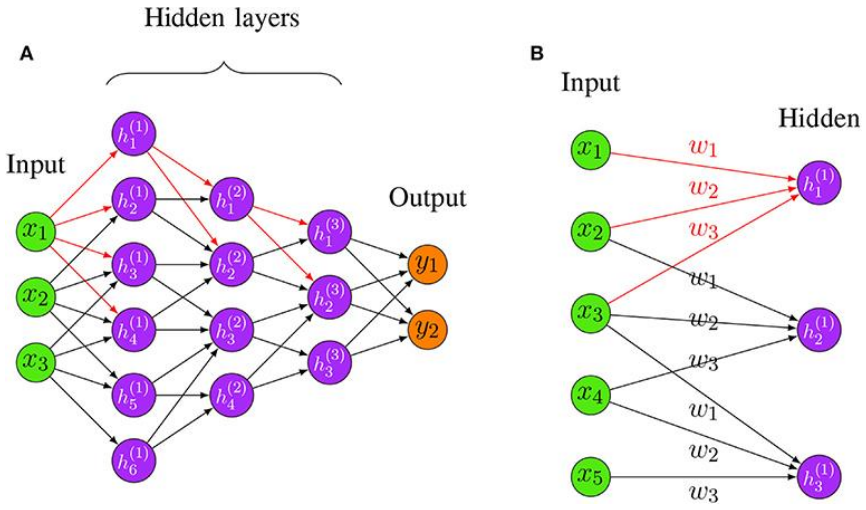


Figure 2.14 Local Connectivity of CNN Model

One of the main advantage of using a CNN model is that it has an automatic feature extraction design, which eliminates the tedious manual feature extraction design process (Khan & Yairi, 2018). Besides that, CNN is used extensively in image processing tasks. This makes the convolution and pooling layers conceptually a two-dimensional plane operation (Khan & Yairi, 2018).

2.8.2 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a strict superset of the feedforward neural network (Lipton, 2015). It is distinct from other deep learning models as it has the inclusion of recurrent edges in each hidden layer that span adjacent timestamps. This feature of RNN has introduced a notion of time to the model (Lipton, 2015). RNN model also has the ability to update current state of a neuron depending on the information of past states and current input data (Y. Yu, Si, Hu, & Zhang, 2019). Figure 2.15 illustrates the difference in architecture between RNN and a regular NN.

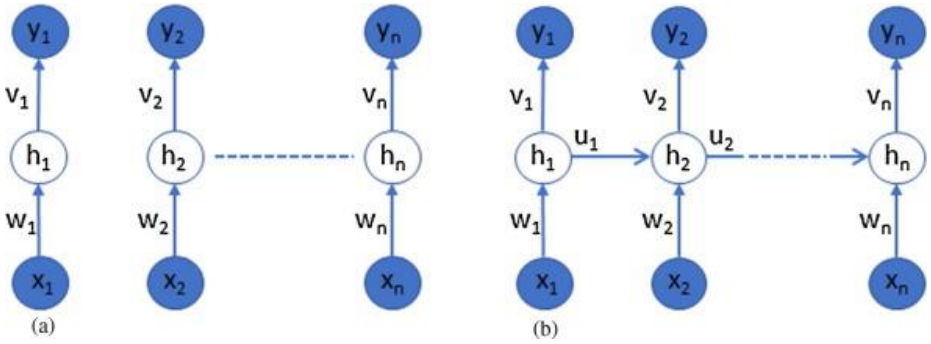


Figure 2.15 Comparison between a typical NN architecture (left) and the RNN architecture (right) (Khan & Yairi, 2018)

RNN is a useful framework which works well on sequential data. Compared to simple NN, RNN is able to overcome the limitation of utilizing the past network result to produce better output. This feature has made it an ideal candidate for health management systems due to their time series nature (Khan & Yairi, 2018).

Despite being powerful, RNN still have certain shortcomings such as the vanishing or explosion gradient problem (Khan & Yairi, 2018), and the long-term dependencies problem (Y. Yu et al., 2019). Fortunately, these problems have been addressed by having certain variations of RNN, such as incorporating Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) (Khan & Yairi, 2018). These variations of RNN, for example LSTM, has achieved exciting results due to their immense learning capacity. LSTM has been applied widely in various tasks, such as speech recognition, trajectory prediction, sentence embedding and many more (Y. Yu et al., 2019).

2.8.3 Deep Belief Network (DBN)

A Deep Belief Network (DBN) is a deep learning model which incorporates different types of NN to produce a new NN model. In DBN, the specific NN used are Restricted Boltzmann Machines (RBMs) and Deep Feedforward Neural Network (D-FFNN). The input layer consists of the RBMs (unsupervised learning) while the D-FFNN (supervised learning) forms the output layer (Emmert-Streib et al., 2020).

DBN is considered as one of the DNN as it consists of multiple RBM which are stacked onto each other in a sequential manner as shown in Figure 2.16. This gives the DBN model its depth (Emmert-Streib et al., 2020).

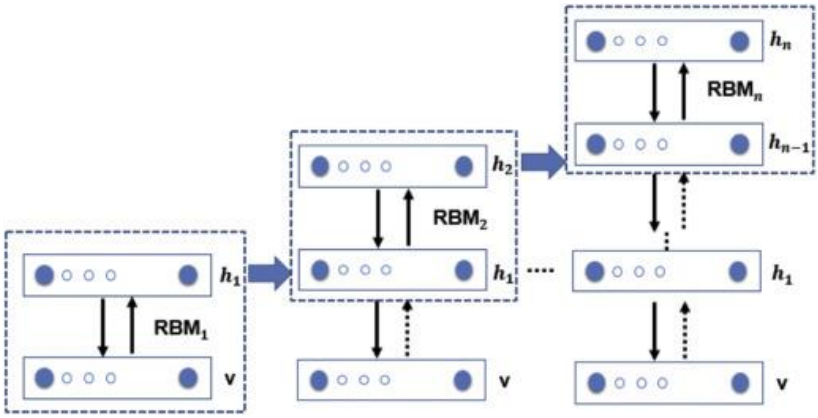


Figure 2.16 Architecture of DBN illustrating the RBMs stacked on top of each other to give the model its depth (Morabito, Campolo, Ieracitano, & Mammone, 2019)

In each RBM, there exists two layers, which consists of the visible layer v and the hidden layer h_n. The first RBM (RBM_1) is trained using the input data from the visible unit. Similarly, the second RMB (RBM_2) consists of the similar components, except it takes the output of the previous RBM (RBM_1), which is the hidden layer h_1 as the input, while having hidden layer h_2 as its output unit. This process repeats itself and ultimately, the entire DBN is fine-tuned with standard backpropagation algorithm (Morabito et al., 2019).

DBN was invented to counter the limitations found in training a regular DNN, where the problems such as slow learning, poor parameter selection which leads to becoming stuck at local minima, and the requirement of large amount of datasets can be solved (Al-jabery, Obafemi-Ajayi, Olbricht, & Wunsch Ii, 2020).

2.9 Autoencoder (AE)

An Autoencoder (AE) is an unsupervised NN model, which is used in representation learning such as feature selection or dimensionality reduction (Emmert-Streib et al., 2020). The fundamental of AE is to train a model to copy its input to its output (Khan & Yairi, 2018) with as little error as possible. The design of AE is similar to DBNs where it consists of a pre-training phase involving the use of several RMBs, and followed by an unrolling phase and finally the fine-tuning phase as shown in Figure 2.17 (Emmert-Streib et al., 2020).

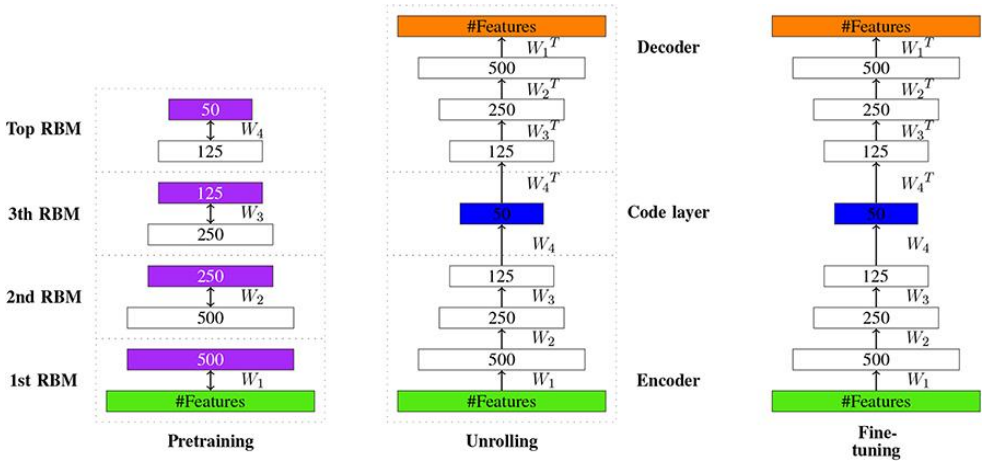


Figure 2.17 The overview of an AE, showing different phases consists of the pre-training, unrolling and fine-tuning phase (Emmert-Streib et al., 2020).

According to Jordan (2018), AE can be understood as a NN architecture such that a bottleneck is imposed in the network (Figure 2.18) which in turn forces a compressed knowledge representation of the original input. The ultimate goal of AE is to reconstruct the original input by going through a series of process which attempts to minimize the reconstruction error of the original input (Jordan, 2018).

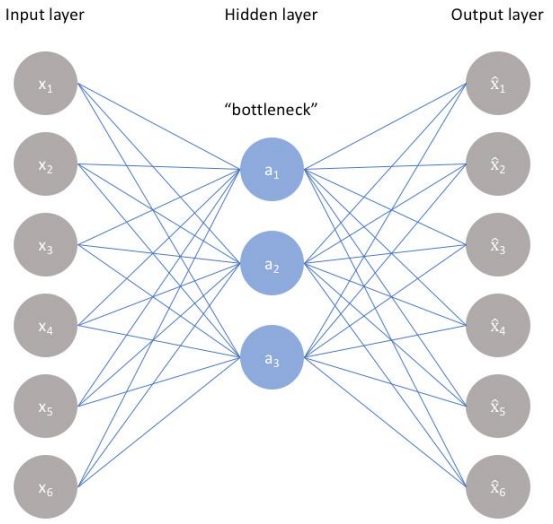


Figure 2.18 The idea of imposing a bottleneck which represents the hidden layer into the NN by Jordan (2018)

The typical architecture of an AE and its operation are illustrated in Figure 2.19. Similar to other ANN, AE consists of the input layer, hidden layer and the output layer. However, the hidden layer in AE can be further divided into encoders and decoders. In AE, a single layer is used to search for the initial parameters for the first hidden layer. This allows the layer to learn intrinsic information about the data without the help of an output class. The subsequent layers then utilize the output from the first hidden layer to find initial weights for the second layer. This process is repeated for every layer of hidden layer (Khan & Yairi, 2018).

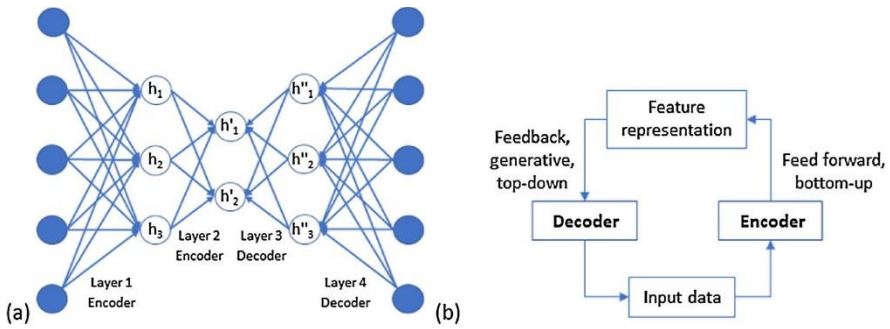


Figure 2.19 The typical architecture of an AE (left) and its operation (right) (Khan & Yairi, 2018)

There are several types of AE such as Stacked Denoising Autoencoder (SDAE) and Variational Autoencoder (VAE) which will be discussed further in the following subsection.

2.9.1 Stacked Denoising Autoencoder (SDAE)

A Denoising Autoencoder (DAE) one of the variations of the classic AE, specifically the stochastic extension to it. In DAE, random noises are introduced to the original input data. This forces the model to learn to reconstruct the original input data based on the noisy version of it (Gondara, 2016). When DAEs are stacked on top of one another, i.e. feeding the output of one DAE to another DAE's input layer adjacent to it, will result in the formation of SDAE as illustrated in Figure 2.20.

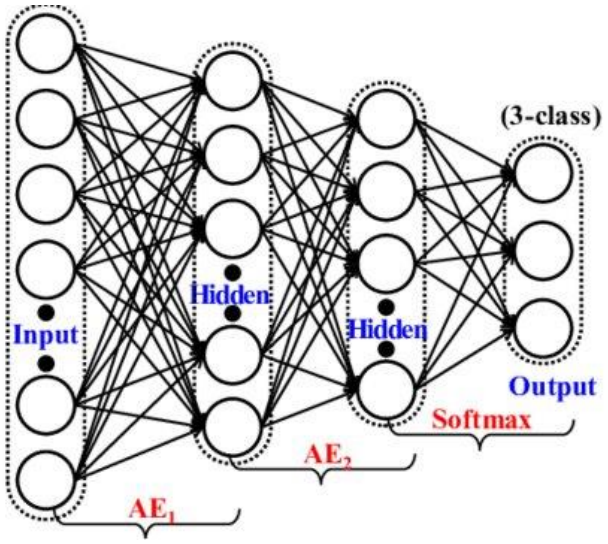


Figure 2.20 Architecture of a SDAE model, showing two hidden layers (Dong, Liao, Liu, & Kuang, 2018)

The process of feeding the output from the first DAE to the input layer of the subsequent DAEs is known as unsupervised greedy layer-wise training (Chandra & Sharma, 2016). At the last layer of the SDAE which is the output layer, a phase of training called fine tuning is performed, where the entire network is trained by

minimizing the classification error by implementing backpropagation (Chandra & Sharma, 2016).

There are two types of noises or corrupted data that can be introduced to the original dataset, which are binary noise and Gaussian noise (Dong et al., 2018). In binary noise implementation, a random number of raw data are forcefully assigned a zero to their value. In general, a percentage of raw data to have their values assigned zero will be determined, such as 30% or 50%, depending on the number of instances in the dataset and the number of input nodes present in the SDAE model (Monn, 2017). In Gaussian noise implementation, a number of Gaussian random values are generated and are added together with the original values in the dataset (Dong et al., 2018). Figure 2.21 illustrates the binary noise implementation in action, where two out of five input nodes have their values set to zero or missing.

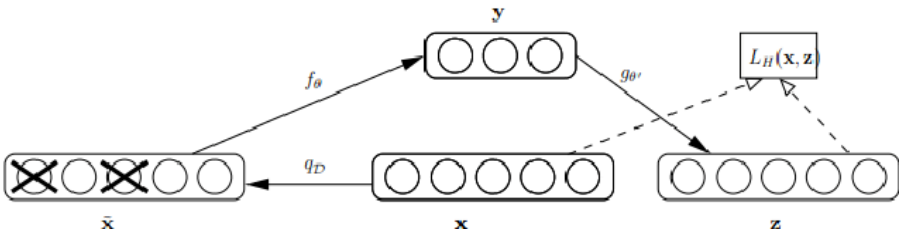


Figure 2.21 Basic architecture of DAE, where certain inputs are deliberately set to zero or missing (Gondara, 2016).

2.9.2 Variational Autoencoder (VAE)

A Variational Autoencoder (VAE) is one of the subtypes of an AE. However, it is not precisely the same as a typical AE model, as it is able to generate new data from the original input data, which is also termed as a generative model (Dong et al., 2018). The generated data is, obviously, not present in the original input data. A generic AE works in a way that it is trained to reproduce the input data with the minimum amount of reconstruction loss. Therefore, it cannot possibly generate a new

data which the model itself has never seen it before. Hence, VAE is the solution to the inability of regular AE to generate new data.

According to Rocca (2019), VAE can be defined as a type of AE which is trained in a way that regularization is done well to prevent overfitting, and at the same time it is made sure that the latent space produced by the VAE has good properties to become a generative model. It has the similar architecture as a typical AE, except it has a special component called the "sampler" which facilitates the new data generation process. The architecture of a VAE is depicted in Figure 2.22.

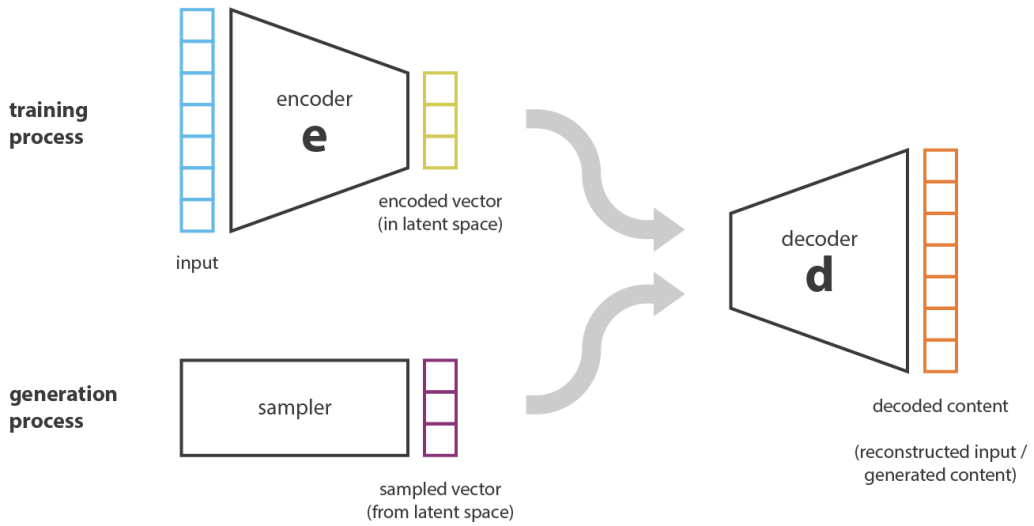


Figure 2.22 Architecture of a VAE model, consisting the similar component as a regular AE except with a sampler (Rocca, 2019)

Regularization refers to the process of tuning the latent space to make generative process possible. It involves two main properties which are continuity and completeness. Continuity is referred as two points which are close to each other in the latent space should not cause the decoder to produce a completely different content after the decoding process. Completeness on the other hand is referred as "meaningful" content should be produced after decoding a point sampled from the latent space (Rocca, 2019). Figure 2.23 shows the comparison between two VAEs with and without regularization.

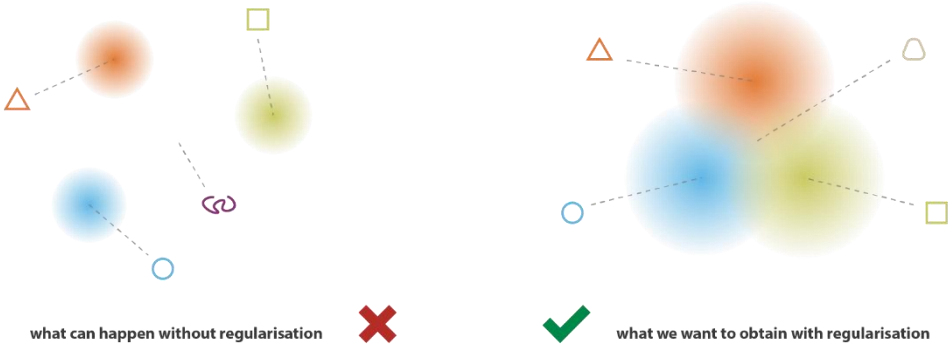


Figure 2.23 Comparison between two VAE. The latent space produced on the left is by the VAE without regularization, while the one produced on the right is by a VAE with regularization (Rocca, 2019)

The training of the VAE can be generalized into four steps. Firstly, the input is encoded as distribution over the latent space by the encoder. The encoding of the input as distribution instead of a single point is one of the process in regularizing the latent space. Next, from the distribution of latent space, a point is sampled and subsequently decoded by the decoder to compute the reconstruction error. Finally, backpropagation of the reconstruction error in the network is performed to regulate the model (Rocca, 2019). Figure 2.24 shows the difference between a regular AE and a VAE in term of mode training.

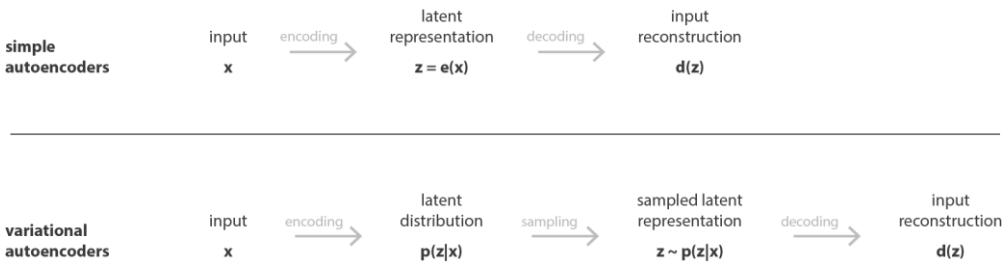


Figure 2.24 Difference between a simple AE and a VAE in the training phase (Rocca, 2019)

2.10 Chapter Summary

As a summary, this chapter reviewed several topics which are related to this research. In the beginning, we have gone through several previous researches done by great researchers related to lung cancer classification. Following that, omics is discussed, including five of its main subfields and their integration which is multi-omics. In multi-omics, we have gone through typical multi-omics workflow and the techniques to integrate omics data efficiently.

Feature selection is another main focus of this research. Here, several feature selection methods are discussed. The focus of this section is on the wrapper method, which is where the RFE method belongs to. Next, SVM is introduced and further discussed the feature selection method using RFE with SVM as its dedicated classifier.

Deep learning algorithms are almost inevitable to be included in any cancer-related researches. Firstly, the fundamental of deep learning, which is ANN is reviewed. Next, several deep learning models such as CNN, RNN, DBN and AE are discussed. Autoencoder is further explored by reviewing two of its variations which are SDAE and VAE.

In the next section of the research, the experimental methodologies related to RFE-SVM, SDAE and VAE will be discussed in details.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

In the previous chapter, we have reviewed several components which are involved in this research, such as lung cancer, topics related to omics and multi-omics, feature selection and so on. Besides that, machine learning algorithm SVM was also reviewed, which will be applied in the feature selection phase in this research. Besides machine learning, several variants of deep learning models are also reviewed. Specifically, the SDAE and VAE autoencoder variants are used to build the lung cancer classification model.

In this chapter, the methodology of this research is covered. This includes the research framework, the dataset which are obtained to be used in this research, and the performance measure for the classification model.

3.2 Research Framework

The research framework encompasses 6 phases or steps. The first phase is basically the initial study of the research including the literature review, identifying the methodology for development, and acquiring the dataset. The second phase is the data pre-processing step, in which the data is modified or cleaned thoroughly to become suitable for the model development. This is done to all three omics data and one clinical data. The next step is to integrate the three omics data into a single, multi-omics dataset. Feature selection is performed in Phase 4 using SVM-RFE on the multi-omics dataset. In phase 5, both the SDAE and VAE models are built to compute the result of the classification. Finally, in phase 5, the result obtained is further studied and analysed to acquire findings.

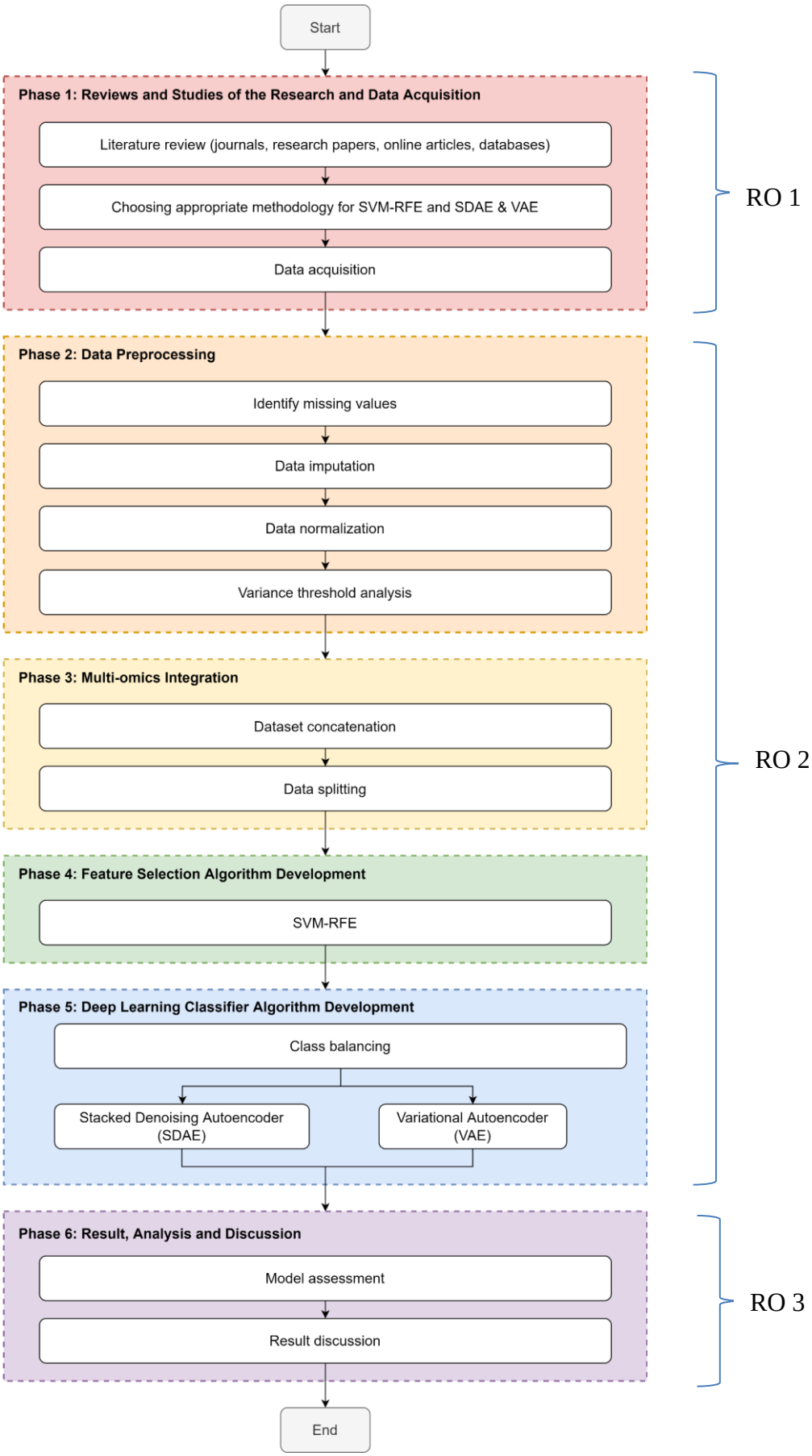


Figure 3.1 Research Framework

3.2.1 Phase 1: Reviews and Studies of the Research and Data Acquisition

In phase 1, extensive literature review is done to obtain substantial understanding regarding the components involved in the research. The literature review was done on various sources, such as the journals, research papers, online articles and databases. The contents which are reviewed are the existing lung cancer classification researches, omics and multi-omics definitions and its integration, feature selection strategies, SVM, deep learning architectures involving the two chosen autoencoders, which are the SDAE and VAE.

After reviewing SVM, SDAE and VAE, the most appropriate methodology for these algorithms are chosen to be applied in the research in the next chapter. The dataset acquisition was also involved in this phase. The dataset was obtained from an open source website which is http://acgt.cs.tau.ac.il/multi_omic_benchmark/download.html.

At the end of this phase, RQ1 has been answered and RO1 has been achieved.

3.2.2 Phase 2: Data Preprocessing

In this phase, the dataset which are obtained from Phase 1 has undergone pre-processing. In data pre-processing, duplicated rows are identified and removed. Then, the missing values are identified. Missing values can come in different form depending on the source of dataset. The most common type of missing data can be represented in symbol such as "?", or "N/A" (not available or NaN in python) or simply left as a blank space.

After the missing values have been identified and located, data imputation is performed. Data imputation refers to the process of dealing with the missing values. There are various strategies to deal with missing values. One of the most common method is to replace the cells with missing values with the mean of the column. Other than that, missing values can also be replaced with the median value of the

column. Excessive missing values can affect the accuracy of the model dramatically. Therefore, removing rows or columns involving majority missing values is another way to impute data.

Lastly, when the data is imputed and no more missing values are left, the data will undergo data normalization. Data normalization is a process of modifying or rescaling the values of data in each column so that they range between 0 and 1. After normalization, variance threshold (VT) analysis is done on each omics dataset to check for zero variance predictors. At the end of Phase 2, the cleaned datasets are used as input in Phase 3.

3.2.3 Phase 3: Multi-omics integration

Phase 3 involves the multi-omics integration. The pre-processed omics datasets obtained from Phase 2 are concatenated together to produce a single dataset, which is now known as a multi-omics dataset. The involved omics datasets are genomics (gene expression), epigenomics (DNA methylation) and transcriptomics (miRNA expression). The clinical dataset which contains the outcome of the patient is also integrated together with the three omics datasets to represent the output class of the multi-omics dataset.

At the end of this phase, the integrated multi-omics data is provided as input for Phase 4.

3.2.4 Phase 4: Feature Selection Algorithm Development

In phase 4, the algorithm of SVM-RFE is developed using the identified methodology. The SVM-RFE is the used on the integrated multi-omics obtained from Phase 3 to effectively select the most relevant attributes which have the highest

correlation with the output class. The feature selection process is only performed after the integration of omics data.

In general, SVM-RFE covers four basic steps. Firstly, the algorithm trains an SVM classifier using the training dataset to obtain the feature weight. Next, it ranks the features using the weights of each feature which are obtained from the SVM classification. Then, the features which hold the lowest weight will be removed as they contribute the least to the model accuracy. The last step involves repeating the SVM classifier training with the remaining features until a stopping criterion is reached. At the end of Phase 4, the feature selected multi-omics dataset will be used in Phase 5 for classification.

3.2.5 Phase 5: Deep Learning Algorithm Development

In phase 5, deep learning algorithms are developed. Specifically, the SDAE and VAE algorithms. The identified methodologies for these two deep learning architectures are applied to the feature selected multi-omics dataset obtained from Phase 4 to build their respective model for lung cancer classification. The results obtained from the two autoencoders are compared to each other in the next phase. The classification result obtained from both classifiers will be used in Phase 6 for further analysis.

Due to our integrated dataset being imbalance in output class, class balancing is performed using SMOTE to balance out instances in the output class in the dataset. SMOTE works wonder in a way that it performs oversampling on the minority class to make the number equal to the originally majority class. Therefore, in the end of the process, both the classes will be balanced with equal number of instances.

At the end of Phase 2 to Phase 5, RQ2 has been answered and RO2 has been achieved.

3.2.6 Phase 6: Result, Analysis and Discussion

In phase 6, the results obtained from Phase 4 & 5 are analyzed and discussed thoroughly. Both the SVM-RFE and the autoencoder models are assessed along with the quality of the selected feature subsets suing SVM-RFE. The confusion matrix is used to assess the model's performance. Metrics such as accuracy, AUC score, sensitivity, precision, and F1 score can be used to decide which model performed better. Further details related to performance measure are mentioned in Chapter 3.4.

At the end of Phase 6, RQ3 has been answered and RO3 has been achieved.

3.3 Dataset

The lung cancer dataset is obtained from http://acgt.cs.tau.ac.il/multi_omic_benchmark/download.html, which is an open source website. The website also contains other omics dataset for other cancers, such as breast cancer, prostate cancer and ovarian cancer. Despite being labeled as pre-processed, the datasets still contain missing values which require further data-preprocessing.

The dataset came in 4 files, which contains 3 omics datasets and 1 clinical dataset. The 3 omics datasets include gene expression, DNA methylation and miRNA expression. Each of the datasets contain different number of instances as well as their features/attributes. The number of instances and features for each dataset are summarized in Table 3.1. Note that this summary in Table 3.1 is refers to the raw dataset obtained from the source before data transposition. The need of data transposition for omics datasets will be addressed in Chapter 4.3.1 later.

Table 3.1 Summary of the LUSC multi-omics benchmark dataset obtained from http://acgt.cs.tau.ac.il/multi_omic_benchmark/download.html

Data / omics type	Number of instances	Number of features
Gene expression (genomic)	20531	553
DNA methylation (epigenomics)	5000	413
miRNA (Transcriptomics)	1046	388
Clinical data	626	127

The target class "sample_type", which is the attribute which indicates whether a patient is healthy or has lung cancer is extracted from the clinical data. The summary of the "sample_type" class is shown in Table 3.2.

Table 3.2 Summary of the distribution of the classes for "sample_type" attribute

	Class	
	Primary Tumour	**Solid Tissue Normal**
Number of instances	506	120

3.4 Performance Measurements

For the purpose of training and testing, the integrated multi-omics data is split into train and test dataset with a percentage of 70% and 30%. For feature selection, the train data is used in a 2-fold cross validation. For deep learning unsupervised learning, the train data is split into another 70% of train data and 30% of validation data. This results in a 49% train data, 21% validation data and 30% test data. The train data is used for model training while the validation data will be used for fine tuning. The test data will be used to obtain the final classification result. The data distributions for both implementations are summarized in Figure 3.2.

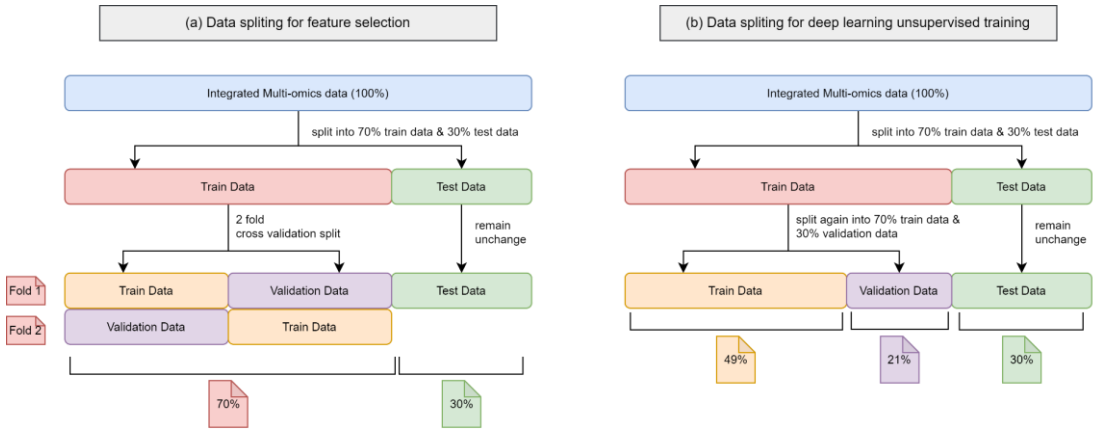


Figure 3.2 Data distribution for the integrated multi-omics data.
(a) Data splitting for feature selection.
(b) Data splitting for deep learning unsupervised training

When both the SDAE and VAE models have been developed with train data and tested with test data, the next step is to assesses both model's performances. The performance can be measured using the predicted outcome by the models. In this research, both models are required to classify the inputs provided to them into their respective output class, which is "sample_type". Since the output class is a binary class, the classifiers will predict the outcome of the inputs into either Primary Tumour or Solid Tissue Normal.

A confusion matrix is deployed to study the outcome of both classification models. There are several evaluation metrics that are contained in a confusion matrix, such as the accuracy, AUC ROC, precision, recall and F1 score.

In this research, "Primary Tumour" is assigned as the positive class, while "Solid Tissue Normal" is assigned as the negative class. There is a total of 4 possible outcomes in the confusion matrix, which are the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP refers to the number of patients predicted with "Primary Tumour" when they indeed have the cancer; TN refers to the number of patients predicted with "Solid Tissue Normal" when they indeed healthy; FP refers to the number of patients predicted as "Primary Tumour" when the patient is actually healthy; and lastly, FN refers to the number of patients predicted as "Solid Tissue Normal" when the patient actually has lung cancer. The confusion matrix is represented in Table 3.3.

Table 3.3 Confusion Matrix

Predicted / Actual	**Primary Tumour**	**Solid Tissue Normal**
Primary Tumour	TP	FN
Solid Tissue Normal	FP	TN

With these 4 possible outcomes in the confusion matrix, we can calculate the evaluation metrics, which are measured in percentage or probability (decimals). Accuracy refers to the total number of correctly predicted classes over all instances; Precision refers to the number of TP over the sum of TP and FP; Recall refers to the number of TP over the sum of TP and FN; and F1 score is a metric which measure the relationship between the Precision and the Recall, or the harmonic mean of the Precision and the Recall.

Besides, Receiver Operator Characteristic Area Under Curve (ROC AUC) is also used to measure the classification capability of the classifiers. It takes involves the relationship between the true positive proportion and the false positive proportion, whereby a graph is plotted with the true positive proportion as the x-axis and the false positive proportion as the y-axis. ROC AUC ranges between 0.5 to 1.0, whereby 0.5 represents a 50-50 guess by the classifier and 1.0 represents 100% accurate classification (Bowers & Zhou, 2019).

Table 3.4 summarizes the equations involved in calculating these metrics.

Table 3.4 Equations for each metrics

Measurements	**Equation**
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$
Precision	$\frac{TP}{TP + FP}$
Recall	$\frac{TP}{TP + TN}$
F1 score	$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

AUC ROC	Relationship between the true positive proportion (y-axis) and the false positive proportion (x-axis)

3.5 Chapter Summary

In this chapter, we have discussed about the research framework in details. In Phase 1, preliminary studies regarding the research is done to gain better understanding of the topics involved in the research. Dataset to be used for classification is also obtained during this phase. Phase 2 refers to the data pre-processing step, in which we thoroughly "clean" the dataset to become usable for the model. Phase 3 is where we integrate the three omics dataset into a single dataset, which is now called a multi-omics dataset. Data splitting is also carried out in this phase.

Phase 4 is the starting point of the algorithm development, where the algorithm for SVM-RFE for feature selection is developed. The next phase, Phase 5 involves class balancing and the development of the classifying algorithms, which are the SDAE and VAE autoencoder classifier. At the end of Phase 5, results of the classification should be obtained from both autoencoder classifiers. In Phase 6, the result from Phase 5 is discussed in detail, such as the results obtained from SVM-RFE, and the comparison between the performance of SDAE and VAE.

CHAPTER 4

RESEARCH DESIGN AND IMPLEMENTATION

4.1 Introduction

In this chapter, the methods to be used in this study will be discussed. The steps of pre-processing and the algorithm of SVM-RFE, SDAE and VAE will be discussed in detail in their respective subchapter. In this study, SVM-RFE is used to perform feature selection on the multi-omics data to obtain a subset of features which are good predictor for the output class. As for the classification models, SDAE and VAE are used. Two deep learning models are deployed to compare their performance on the same dataset.

4.2 Experiment Design

To achieve the research objectives, several tools are used as mentioned in Chapter 1.6. The Python programming language is used for all the programming parts in this study. As mentioned previously, several libraries from Python are used. The Pandas and Numpy libraries are used in data pre-processing, while the scikit-learn library is used for feature selection. Finally, Keras library is used for the model development for the SDAE and VAE autoencoders for classification. Figure 4.1 shows the experimental workflow.

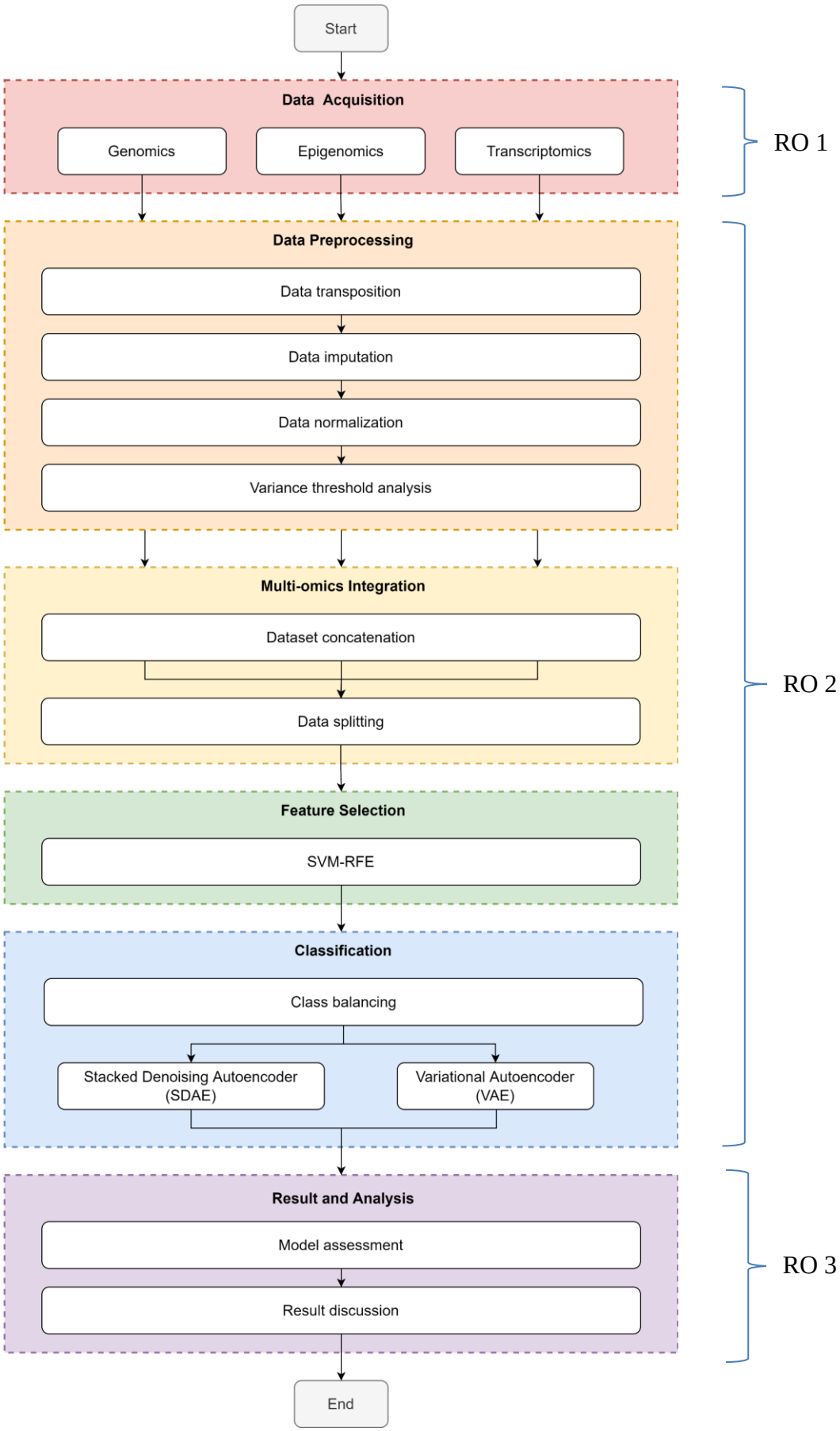


Figure 4.1 Experimental Design of the study

First of all, the workflow starts with data identification or data acquisition, where the necessary data for the study are procured. There is a total of 4 datasets for this study, which consists of three omics data and one clinical data. The three omics data are the genomics data, epigenomics data and the transcriptomics data. These are the main dataset which make up the features/columns of the final dataset. The clinical data on the other hand consists of the outcome of the patient, whether the cancer has "Solid Tissue Normal" or "Primary Tumor". These outcomes are necessary for the study as the patient's condition has to be known for supervised learning.

With the obtained datasets, the study proceeds with data preprocessing. Despite being labelled as preprocessed data, the datasets still contain certain caveats, which will be discussed in details in the subsequent subchapter. In short, the omics datasets should undergo data transposition, as the rows and columns of the raw dataset is not suitable for direct usage. Then, the transposed raw dataset undergoes data imputation by checking for duplicated rows and missing values. When the data is cleaned, data normalization is performed on each omics dataset.

The next step is to combine the omics datasets to produce a single dataset known as multi-omics dataset. This process is also known as multi-omics data integration. In this step, column concatenation method is used for integration. The patientID for each dataset are used as the index of the table and the concatenation process proceed with patientID as the reference.

With the new multi-omics dataset, the study proceeds with the feature selection process. But before that, as mentioned in Chapter 3.2.3, the integrated multi-omics dataset will be split into train, validation and test dataset depending on the experiments. Next, feature selection is performed to select the subset of features which have the highest predictability for the output class for the study. SVM-RFE is used as the feature selection method in this study.

After performing feature selection, the next step is to develop the classification algorithm for the study. Two deep learning methods, which are the

SDAE and VAE methods are used. The same multi-omics dataset will be fed to both classifiers and the end result will be compared.

Lastly, the study is concluded with the result and analysis step. The result obtained from the classification is compiled and further analysis and discussion regarding the results will be discussed in details.

4.3 Design and Implementation

In this subchapter, the steps involved in the study for the multi-omics analysis is discussed in detail. This includes the data pre-processing step, exploratory data analysis (EDA), multi-omics data integration, feature selection using SVM-RFE, and classification model development using Stacked Denoising Autoencoder (SDAE) and Variational Autoencoder (VAE).

4.3.1 Data Preprocessing

Despite being labelled as pre-processed, the obtained datasets still contain certain flaws and requires further pre-processing steps. The omics dataset and the clinical dataset are pre-processed separately.

In the downloaded omics datasets, it is observed that the rows and columns are inverted, as in the rows are representing the omics expressions instead of the samples, and the columns are representing the samples instead of the omics expressions. Therefore, data transposition is required to make better sense of the dataset by transposing the rows and columns. After data transposition, the rows now represent the samples, and the columns now represent the omics expressions. Next, data imputation is performed to check for rows with duplicates and cells with NaN values or missing values. The duplicated rows will be handled by removing them, while the NaN values or missing values will be imputed with value zero as they can

reduce the statistical power of the study and produces biased estimations (Kang, 2013). Next, normalization is performed on each cleaned omics dataset to prepare the data for variance threshold analysis. Variance threshold analysis is performed to remove columns with zero predictive power in order to speed up SVM-RFE by a little bit. With the cleaned omics datasets, the header of the column "geneID" is renamed into "patientID" for each omics dataset, as that particular column now represents the ID of the patient after data transposition.

For the clinical data, it is pre-processed by extracting the necessary columns which are the column which represents the ID of the patients and the column which indicates the condition of the patients, whether they have lung cancer or not. Besides that, the class labels "Primary Tumour" and "Solid Tissue Normal" are renamed into "1" and "0" respectively for classification.

4.3.1.1 Data Transposition

The raw omics data obtained from the source is a little bit misleading as the rows and columns are inverted. As shown in Figure 4.2, the patient id, namely `patientID` which are supposed to be represented in the rows of the dataframe are instead located as columns. On the other hand, the features of the omics data are represented in the rows.

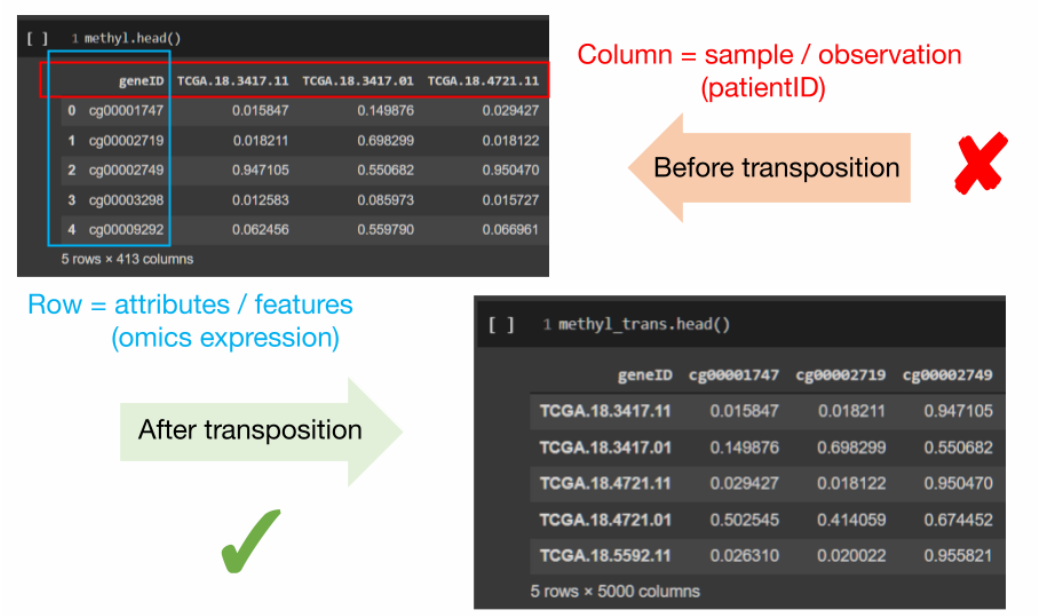


Figure 4.2 Dataset structure before and after data transposition

Therefore, there is a need to perform data transposition to ensure that the `patientID` are represented as rows and the features of each omics dataset represented in columns. Figure 4.3 shows the code snippet which performs the data transposition on each omics dataset.

Figure 4.3 Code snippet for data transposition

After the data transposition, the dimensions of the dataframe for each omics dataset are displayed again for checking. The code snippet for this is shown in Figure 4.4. From the output, it can be seen clearly that the dataframes have been successfully transposed, whereby the rows are transposed into columns and vice versa. Now, the dataframes have the correct format (patient ID as rows and omics information as columns) for further analysis.

After transposition

```
[12]  1 print('gene expression: ', geneExp_trans.shape)
      2 print('DNA methylation: ', methyl_trans.shape)
      3 print('miRNA: ', miRNA_trans.shape)
```

```
gene expression:  (552, 20531)
DNA methylation:  (412, 5000)
miRNA:  (387, 1046)
```

Figure 4.4 Code snippet to show the dimension of the dataframe after data transposition

Figure 4.5, 4.6 and 4.7 shows the code snippet to display the first 5 rows of each omics dataset after transposition. It can be seen that the first column still has "geneID" as its header in each omics dataset while the rows are actually the ID of the patients. This will be addressed in the Chapter 4.3.1.5.

```
[12]  1 geneExp_trans.head()
```

geneID	?|100130426	?|100133144	?|100134869	?|10357	?|10431	?|136542	?|155060	?|26823	?|280660	?|317712	?|340602	?|388795	?|390284	
TCGA.18.3406.01	0.0	75.3622	25.4252	279.9764	1510.2362	0.0	23.6220	0.0000	0.0	0.0	0.0000	4.7244	10.2362	
TCGA.18.3407.01	0.0	14.4148	6.9933	198.2540	968.6013	0.0	301.6175	0.0000	0.0	0.0	0.0000	0.9515	3.8059	
TCGA.18.3408.01	0.0	8.4888	3.3754	254.5929	1474.1213	0.0	49.2362	1.7796	0.0	0.0	0.0000	0.0000	7.7117	
TCGA.18.3409.01	0.0	2.5897	7.1538	164.3744	944.6154	0.0	93.3333	1.0256	0.0	0.0	2.0513	0.0000	7.6923	
TCGA.18.3410.01	0.0	16.6653	19.5722	200.8033	1156.0568	0.0	120.9232	0.0000	0.0	0.0	2.3633	7.8777	3.1511	

5 rows × 20531 columns

Figure 4.5 Code snippet for displaying the first 5 rows of the transposed gene expression dataset

```
[13]  1 methyl_trans.head()
```

geneID	cg00001747	cg00002719	cg00002749	cg00003298	cg00009292	cg00011616	cg00012529	cg00018128	cg00025439	cg00028211	cg0003407
TCGA.18.3417.11	0.015847	0.018211	0.947105	0.012583	0.062456	0.516875	0.977849	0.064575	0.384277	0.247132	0.07568
TCGA.18.3417.01	0.149876	0.698299	0.550682	0.085973	0.559790	0.701763	0.807980	0.382961	0.513249	0.715990	0.51685
TCGA.18.4721.11	0.029427	0.018122	0.950470	0.015727	0.066961	0.442266	0.978429	0.060206	0.661084	0.299236	0.06937
TCGA.18.4721.01	0.502545	0.414059	0.674452	0.536764	0.351668	0.851836	0.966762	0.465739	0.954354	0.655749	0.08398
TCGA.18.5592.11	0.026310	0.020022	0.955821	0.017323	0.069340	0.350202	0.976545	0.102285	0.128341	0.426997	0.0642(

5 rows × 5000 columns

Figure 4.6 Code snippet for displaying the first 5 rows of the transposed DNA methylation dataset

```
[14]  1 miRNA_trans.head()
```

geneID	hsa-let-7a-1	hsa-let-7a-2	hsa-let-7a-3	hsa-let-7b	hsa-let-7c	hsa-let-7d	hsa-let-7e	hsa-let-7f-1	hsa-let-7f-2	hsa-let-7g	hs
TCGA.18.5592.01	5998.824696	11667.530081	5913.311862	16781.396385	403.700120	1748.041408	553.844746	20.881041	6035.615101	857.117004	811.
TCGA.18.5595.01	5348.668096	10618.397887	5498.172459	24295.057027	3577.340401	1496.239666	682.935931	16.744489	5235.044780	770.246479	590.
TCGA.21.5782.01	3829.339596	7731.024727	3852.399735	13874.291042	1686.103124	824.739098	552.539023	14.016947	4173.433047	601.372259	697.
TCGA.21.5783.01	4114.228785	8111.916613	4126.215855	12566.112021	1656.879446	732.543163	1310.253340	12.653018	3687.022931	283.361015	612.
TCGA.21.5784.01	5035.557137	10008.799220	5005.512379	23486.098875	2102.854892	623.428736	692.698595	16.969725	4379.301721	765.028569	785.

5 rows × 1046 columns

Figure 4.7 Code snippet for displaying the first 5 rows of the transposed miRNA dataset

At this end of this subchapter, the three omics datasets are successfully transposed into a proper dataframe format, whereby the rows now represent the patientID (observation), while the columns represent the omics expression (feature/attribute). The summary of the status of each dataset after this step is shown in Table 4.1. The changes are shown in bold.

Table 4.1 Summary of Data Transposition

Dataset	**Data Shape (rows, columns)**	
	Raw dataset	**After Data Transposition**
Gene expression	(20531, 552)	(**552**, **20531**)
DNA Methylation	(5000, 412)	(**412**, **5000**)
miRNA	(1046, 387)	(**387**, **1046**)
Clinical Data	(626, 127)	(626, 127)

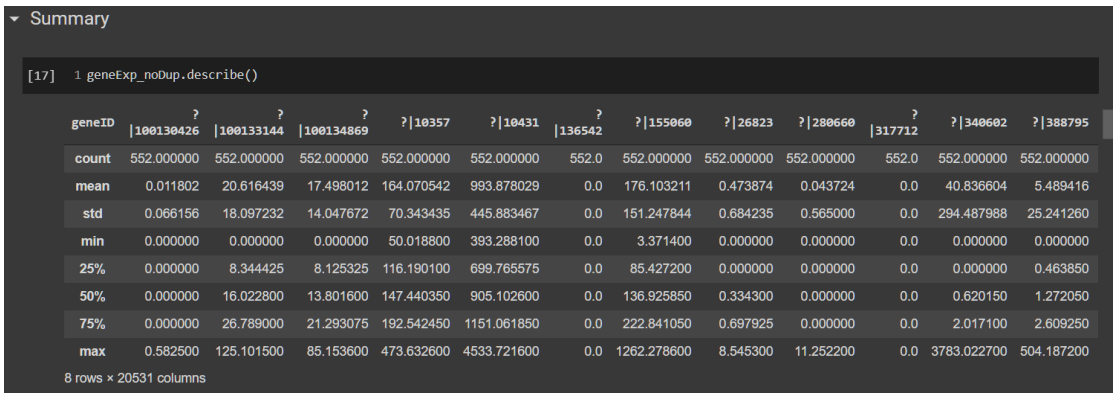
4.3.1.2 Data Imputation

The first step in data imputation involves removing the duplicated rows in all 4 datasets. Duplicated rows in dataset could be catastrophic when it is being fed into a model for classification. When the dataset is split into training and testing dataset, the duplicated row of data could be present in both training and testing dataset. This can cause issues such as biases in the classification result. Figure 4.8 shows the code snippet for the removal of duplicated rows.

Figure 4.8 Code snippet to remove duplicates in all dataset

Figure 4.9 Code snippet to check the dimension of all dataset after removing duplicates

The result of the deletion of duplicates is displayed as shown in Figure 4.9. Fortunately, there is no duplicated rows in the dataset obtained from the source.



geneID	?|100130426	?|100133144	?|100134869	?|10357	?|10431	?|136542	?|155060	?|26823	?|280660	?|317712	?|340602	?|388795
count	552.000000	552.000000	552.000000	552.000000	552.000000	552.0	552.000000	552.000000	552.000000	552.0	552.000000	552.000000
mean	0.011802	20.616439	17.498012	164.070542	993.878029	0.0	176.103211	0.473874	0.043724	0.0	40.836604	5.489416
std	0.066156	18.097232	14.047672	70.343435	445.883467	0.0	151.247844	0.684235	0.565000	0.0	294.487988	25.241260
min	0.000000	0.000000	0.000000	50.018800	393.288100	0.0	3.371400	0.000000	0.000000	0.0	0.000000	0.000000
25%	0.000000	8.344425	8.125325	116.190100	699.765575	0.0	85.427200	0.000000	0.000000	0.0	0.000000	0.463850
50%	0.000000	16.022800	13.801600	147.440350	905.102600	0.0	136.925850	0.334300	0.000000	0.0	0.620150	1.272050
75%	0.000000	26.789000	21.293075	192.542450	1151.061850	0.0	222.841050	0.697925	0.000000	0.0	2.017100	2.609250
max	0.582500	125.101500	85.153600	473.632600	4533.721600	0.0	1262.278600	8.545300	11.252200	0.0	3783.022700	504.187200

Figure 4.10 Code snippet to show the summary of the partially cleaned gene expression dataset

After some data cleaning is done, the summary of the partially cleaned data is displayed for further investigation. Figure 4.10 shows the summary of the partially cleaned gene expression dataset. From the summary of the omics dataset, it can be seen that the dataframe contains several columns or features which has their `mean`, `min` and `max` as 0. This means that these columns only have zero for all observations. These columns or features that has only a single value are zero-variance predictor which do not provide any predictability to the output class. Therefore, they are recommended to be removed from the dataframe (M. Kuhn, & Johnson, K., 2019).

Before handling the zero-variance predictors in the next subchapter, the original missing values (NaN values) are checked in each omics dataset. Figure 4.11 shows the code snippet for checking missing values in term of NaN. It can be seen that the raw omics data obtained from source do not contain any missing values.

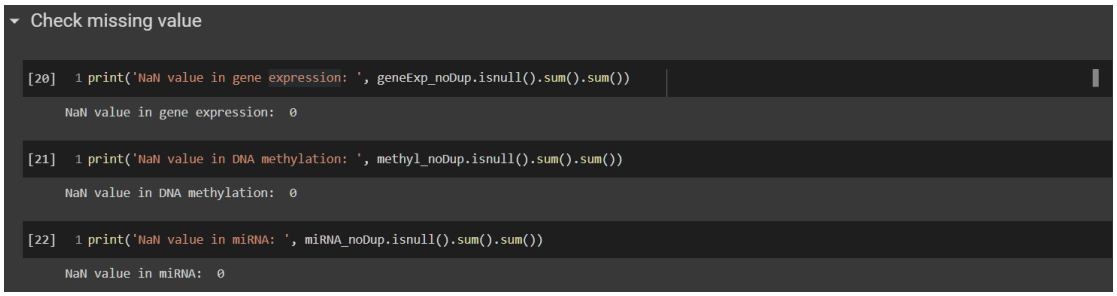


Figure 4.11 Code snippet to check for missing values in term of NaN in each omics dataset

4.3.1.3 Data Normalization

After data imputation, the omics datasets are normalized. This is done by separating the patient ID and the omics data. Then, normalization is performed on the omics data. After the normalization process, the values of the omics data are now in the range of 0 to 1 to improve the data quality and the quality of the machine learning algorithms (Singh & Singh, 2020). Figure 4.12 to 4.14 shows the normalized omics data.

```
[ ]  1 geneExp_patientID = pd.DataFrame(geneExp_index.iloc[:, 0])
     2 geneExp_data = pd.DataFrame(geneExp_index.iloc[:, 1:])

[ ]  1 geneExp_data_normalized = pd.DataFrame(scaler.fit_transform(geneExp_data), columns=geneExp_data.columns)

[ ]  1 geneExp_data_normalized
```

geneID	?|100130426	?|100133144	?|100134869	?|10357	?|10431	?|136542	?|155060	?|26823	?|280660	?|317712	...	ZXDA|7789	ZX
0	0.0	0.602408	0.298580	0.542847	0.269766	0.0	0.016086	0.000000	0.0	0.0	...	0.446393	
1	0.0	0.115225	0.082126	0.349930	0.138950	0.0	0.236909	0.000000	0.0	0.0	...	0.241421	
2	0.0	0.067855	0.039639	0.482926	0.261043	0.0	0.036432	0.208255	0.0	0.0	...	0.292701	
3	0.0	0.020701	0.084011	0.269952	0.133157	0.0	0.071460	0.120019	0.0	0.0	...	0.515664	
4	0.0	0.133214	0.229846	0.355948	0.184224	0.0	0.093376	0.000000	0.0	0.0	...	0.227373	
...	...	...	...	...	...	...	...	...	...	...	...	...	
547	0.0	0.159473	0.553149	0.299091	0.192018	0.0	0.177495	0.193744	0.0	0.0	...	0.382644	
548	0.0	0.240502	0.179625	0.327427	0.206438	0.0	0.114483	0.049173	0.0	0.0	...	0.304776	
549	0.0	0.428926	0.392123	0.496477	0.095660	0.0	0.675989	0.000000	0.0	0.0	...	0.255934	
550	0.0	0.580062	0.495378	0.260009	0.016321	0.0	0.449478	0.155782	0.0	0.0	...	0.544769	
551	0.0	0.357710	0.277503	0.062864	0.099667	0.0	0.244860	0.061086	0.0	0.0	...	0.437211	

552 rows × 20531 columns

Figure 4.12 Code snippet for the normalization of the gene expression data

```
[ ]  1 methyl_patientID = pd.DataFrame(methyl_index.iloc[:, 0])
     2 methyl_data = pd.DataFrame(methyl_index.iloc[:, 1:])

[ ]  1 methyl_data_normalized = pd.DataFrame(scaler.fit_transform(methyl_data), columns=methyl_data.columns)

[ ]  1 methyl_data_normalized
```

geneID	cg00001747	cg00002719	cg00002749	cg00003298	cg00009292	cg00011616	cg00012529	cg00018128	cg00025439	cg00028211
0	0.002847	0.004810	0.984372	0.000272	0.030059	0.506335	0.997398	0.037146	0.364027	0.232569
1	0.147377	0.729302	0.540043	0.082095	0.593481	0.701831	0.813255	0.397239	0.503883	0.735485
2	0.017490	0.004715	0.988143	0.003778	0.035162	0.427445	0.998027	0.032204	0.664193	0.288458
3	0.527676	0.426504	0.678770	0.584681	0.357702	0.860514	0.985380	0.490861	0.982212	0.670867
4	0.014129	0.006739	0.994141	0.005557	0.037857	0.330100	0.995986	0.079795	0.086494	0.425499
...	...	...	...	...	...	...	...	...	...	...
407	0.044311	0.843443	0.283627	0.025448	0.074708	0.956824	0.579403	0.691209	1.000000	0.906708
408	0.330334	0.024993	0.630227	0.407363	0.365163	0.700148	0.516686	0.563022	0.436764	0.633140
409	0.274211	0.508431	0.618242	0.023181	0.666571	0.375160	0.491338	0.647238	0.520454	0.260979
410	0.005917	0.006892	0.934445	0.046530	0.906145	0.797681	0.237945	0.963307	0.971044	0.984584
411	0.206518	0.367592	0.573508	0.297008	0.199033	0.797856	0.540062	0.099249	0.862625	0.391290

412 rows × 5000 columns

Figure 4.13 Code snippet for the normalization of the DNA methylation data

```
[ ]  1 miRNA_patientID = pd.DataFrame(miRNA_index.iloc[:, 0])
     2 miRNA_data = pd.DataFrame(miRNA_index.iloc[:, 1:])

[ ]  1 miRNA_data_normalized = pd.DataFrame(scaler.fit_transform(miRNA_data), columns=miRNA_data.columns)

[ ]  1 miRNA_data_normalized
```

geneID	hsa-let-7a-1	hsa-let-7a-2	hsa-let-7a-3	hsa-let-7b	hsa-let-7c	hsa-let-7d	hsa-let-7e	hsa-let-7f-1	hsa-let-7f-2	hsa-let-7g	...	hsa-mir-941-3	hsa-mir-941-4	hsa mir-94
0	0.312092	0.297506	0.302226	0.196355	0.010394	0.715232	0.122674	0.314912	0.191348	0.344950	...	0.0	0.0	0.10745
1	0.271110	0.264816	0.276391	0.297879	0.119368	0.598595	0.156246	0.240188	0.160535	0.299305	...	0.0	0.0	0.78334
2	0.175341	0.174849	0.173972	0.157075	0.054428	0.287550	0.122335	0.190917	0.119675	0.210574	...	0.0	0.0	0.08243
3	0.193299	0.186717	0.191012	0.139399	0.053424	0.244844	0.319387	0.166278	0.100954	0.043482	...	0.0	0.0	0.08750
4	0.251374	0.245821	0.245732	0.286948	0.068738	0.194302	0.158785	0.244257	0.127598	0.296564	...	0.0	0.0	0.07348
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
382	0.214582	0.212244	0.209253	0.059291	0.007054	0.382183	0.138507	0.399701	0.267913	0.238114	...	0.0	0.0	0.15205
383	0.257395	0.252072	0.256485	0.069747	0.027039	0.038851	0.239184	0.262358	0.288853	0.148430	...	0.0	0.0	0.07866
384	0.195985	0.195351	0.198617	0.064819	0.073602	0.080927	0.104938	0.245702	0.147021	0.159881	...	0.0	0.0	0.10353
385	0.269465	0.262616	0.270333	0.215586	0.242241	0.147675	0.279501	0.370982	0.214933	0.080044	...	0.0	0.0	0.10897
386	0.270217	0.264854	0.265534	0.123776	0.044856	0.227641	0.155666	0.120249	0.178090	0.202422	...	0.0	0.0	0.05331

387 rows × 1046 columns

Figure 4.14 Code snippet for the normalization of the mIRNA expression data

4.3.1.4 Variance Threshold Filter

Next, the variance threshold (VT) analysis is carried out on the normalized dataset. The VT analysis primarily focus on checking for zero variance predictors. A feature with only a single unique value is referred as a zero-variance feature. These features are uninformative and provide very little predictive power to the classification (M. Kuhn & Johnson, 2013). Therefore, the VT analysis is carried out before feature selection to filter out these features. First of all, the variance threshold is set to 0 as shown in Figure 4.15.

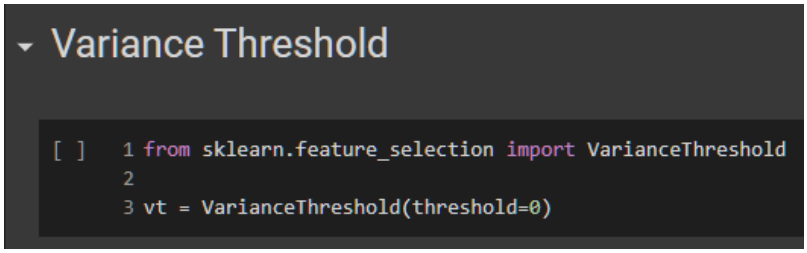


Figure 4.15 Initializing the variance threshold function

Then, for each omics dataset, the data are fit onto the variance threshold function. The number of features which has zero variance are displayed and the features with no zero variance are extracted. The final dimensions of each omics dataset are also displayed. The implementations of VT analysis for each omics dataset are shown in Figure 4.16 to 4.18.

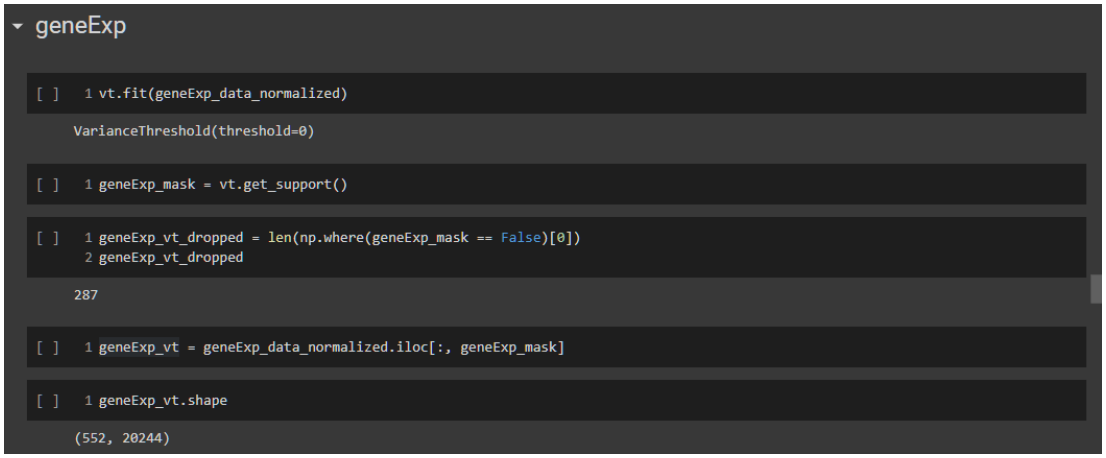


Figure 4.16 Implementation of VT analysis on gene expression data

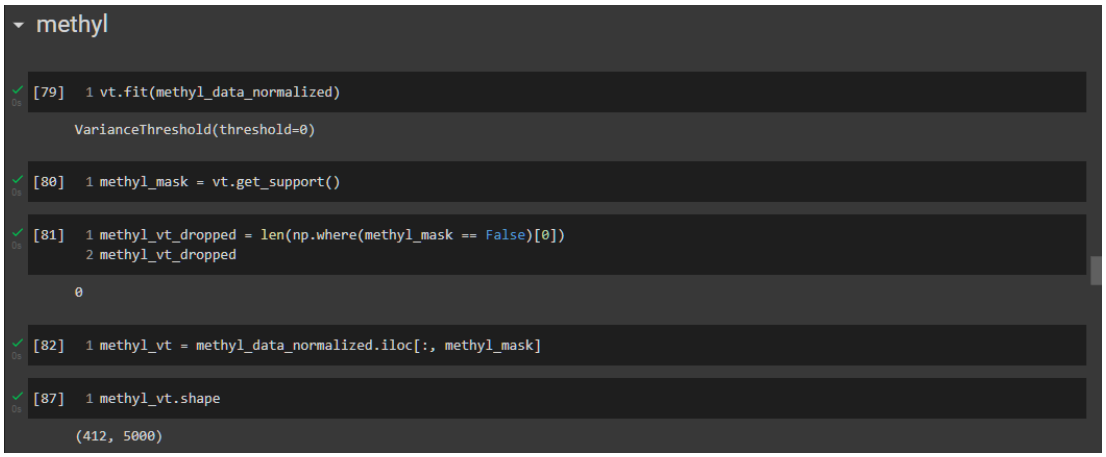


Figure 4.17 Implementation of VT analysis on DNA methylation data

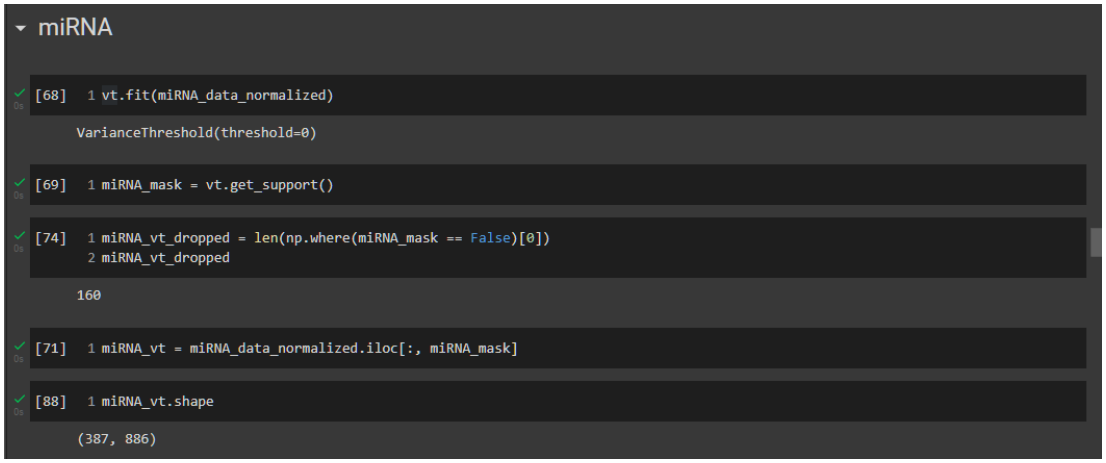


Figure 4.18 Implementation of VT analysis on miRNA expression data

The summary of the VT analysis on the three omics datasets are shown in Figure 4.19. According to the result, only the gene expression data and the miRNA expression data contains zero variance feature. It was found out that the gene expression data has 287 zero variance features while the miRNA expression data contains 160 zero variance features.

```
[91]  1 omics_names = ['Gene Exp.', 'DNA Methylation', 'miRNA Expression']
      2 omics_vt_dropped = [geneExp_vt_dropped, methyl_vt_dropped, miRNA_vt_dropped]
      3 fig = plt.figure()
      4
      5 plt.bar(omics_names, omics_vt_dropped)
      6
      7 plt.text(0, omics_vt_dropped[0] + 2, s=omics_vt_dropped[0])
      8 plt.text(1, omics_vt_dropped[1] + 2, s=omics_vt_dropped[1])
      9 plt.text(2, omics_vt_dropped[2] + 2, s=omics_vt_dropped[2])
     10
     11 plt.xlabel("Omics Types")
     12 plt.ylabel("Number of features")
     13 plt.title("Zero Varience Features in each Omics Data")
     14 plt.show()
```

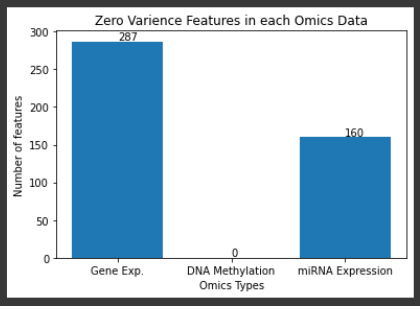


Figure 4.19 Summary of the VT analysis on each omics dataset

The dimensions of each omics dataset are tabulated in Table 4.2. It can be seen that the dimension of gene expression data is reduced from 20531 to 20244 after removing the 287 zero variance features. The dimension of the miRNA expression data is down from 1046 to 886 after the removal of 160 zero variance features.

Table 4.2 Summary of Omics Dimensions before and after VT Analysis

Dataset	**Data Shape (rows, columns)**	
	Before VT	**After VT**
Gene expression	(552, 20531)	(552, **20244**)
DNA Methylation	(412, 5000)	(412, 5000)
miRNA	(387, 1046)	(387, **886**)
Clinical Data	(626, 127)	(626, 127)

4.3.1.5 Clinical Data cleaning

The data cleaning step for clinical data is a little bit different. The raw clinical dataset consists of various information regarding the patient. However, there is only two columns which are important for the study, which is the `sampleID` and `sample_type`. `sampleID` refers to the patient's ID while `sample_type` refers to the study's target class, which indicate whether a patient has lung cancer. "Primary Tumor" refers to patients with lung cancer while "Solid Tissue Normal" refers to the opposite. Therefore, these two columns are extracted from the raw clinical dataset as shown in Figure 4.20.

Figure 4.20 Extraction of important columns from clinical dataset

After the extraction, the columns are renamed appropriately in the first cell in Figure 4.21. Besides that, since the target class consists of string values, the values are modified into nominal values such as '0' and '1', representing the negative (Solid Tissue Normal) and positive (Primary Tumor) output respectively. The values in `patientID` column are also modified so that the format of patient ID are similar across all 4 datasets.

```
[37]  1 clinical_extract = clinical_extract.rename(columns={'sampleID': 'patientID', 'sample_type': 'class'})

[38]  1 clinical_extract['class'] = clinical_extract['class'].replace('Primary Tumor', '1')

[39]  1 clinical_extract['class'] = clinical_extract['class'].replace('Solid Tissue Normal', '0')

[40]  1 clinical_extract['patientID'] = clinical_extract['patientID'].str.replace('\-', '.')

[41]  1 clinical_extract.head()
```

	patientID	class
0	TCGA.18.3406.01	1
1	TCGA.18.3406.11	0
2	TCGA.18.3407.01	1
3	TCGA.18.3407.11	0
4	TCGA.18.3408.01	1

Figure 4.21 Further data cleaning for clinical data

Table 4.3 shows the status of each dataset and change are shown in bold. Note that all four datasets have `patientID` set as the index of the dataframes for data concatenation purpose in Chapter 4.3.3. Therefore, the `patientID` column is not counted in total number of columns for all four datasets, which is why clinical data now only has one column (`class`) instead of two.

Table 4.3 Summary of Clinical Data Pre-processing

Dataset	**Data Shape (rows, columns)**	
	Before Data Cleaning	**After Data Cleaning**
Clinical Data	(626, 127)	(626, **1**)

4.3.1.6 Omics Data & Clinical Data Combination

After cleaning both the omics data and the clinical data, the omics data need to be combined with the clinical data to obtain a complete omics data with class label. In Chapter 4.3.1.1, the issue with incorrect header name was mentioned, whereby the column holding the ID of the patients is not named correctly after data transposition. In this subchapter, the column header will be renamed appropriately. This is important for the multi-omics integration which will be addressed in the next section.

It is necessary to ensure that the column holding the ID of the patients has consistent header naming across all datasets as it will be used as the identifier to concatenate the omics data with the class label from the clinical data. Besides that, the multi-omics integration also relies on this ID column to concatenate the three omics data. Therefore, all 4 datasets are necessary to have the similar column header, which is `patientID`. In Figure 4.22, the previously incorrectly labelled header for patient ID is now renamed from "geneID" into "patientID".

```
[30] 1 geneExp_clean = geneExp_clean.reset_index()
     2 geneExp_clean = geneExp_clean.rename(columns={'index': 'patientID'})
     3 geneExp_clean.head()
```

geneID	patientID	?|10357	?|10431	?|155060	?|57714	?|653553	?|8225	A1BG|1	A2LD1|87769	A2M|2	A4GALT|53947	AAAS|8086	AA
0	TCGA.18.3406.01	279.9764	1510.2362	23.6220	149.6063	785.0394	1621.2598	741.6929	170.2362	9074.6772	181.8898	567.7165	
1	TCGA.18.3407.01	198.2540	968.6013	301.6175	386.7745	666.0324	353.4729	46.7127	118.4063	11310.1713	1318.7441	500.0000	
2	TCGA.18.3408.01	254.5929	1474.1213	49.2362	287.1126	8.3049	43.3042	1.1864	148.3316	14678.8492	190.4197	426.5164	
3	TCGA.18.3409.01	164.3744	944.6154	93.3333	295.8974	216.9231	417.4359	162.6462	68.7795	25126.0923	266.1538	788.7179	
4	TCGA.18.3410.01	200.8033	1156.0568	120.9232	342.6812	150.4646	616.0385	273.8811	172.2190	4292.9610	1039.0726	935.8743	

5 rows × 13435 columns

Figure 4.22 Code snippet to rename column with header geneID into patientID using gene expression dataset as an example

Next, for each omics dataset, an identifier is attached to each of the column as shown in Figure 4.23. For example, a tag of "[geneExp]" is attached in front of each feature in the gene expression dataset.

```
[ ]  1 renamed_columns = list()
     2 for column in geneExp_vt.columns:
     3   renamed_columns.append('[geneExp]' + column)

[ ]  1 geneExp_vt.columns = renamed_columns
```

Figure 4.23 Attaching a tag to each feature in each omics dataset (using gene expression data as an example)

Following that, the patient ID which was detached in Chapter 4.3.1.3 Data Normalization, is reattached to each omics dataset as shown in Figure 4.24.

```
[ ]  1 geneExp_concat_vt_id = [geneExp_patientID, geneExp_vt]

[ ]  1 geneExp_vt_id = pd.concat(geneExp_concat_vt_id, axis=1, join='inner')
```

Figure 4.24 Reattaching the patient ID in each omics dataset (using gene expression data as an example)

Finally, the pre-processing phase ends by attaching the class label from the clinical data to each omics dataset according to the patient's ID. Figure 4.25 shows the process of attaching the class label to each omics dataset and displays the final dataframe of the omics dataset using gene expression data as an example.

```
[ ]  1 geneExp_index_id = geneExp_vt_id.set_index('patientID')
     2 clinical_index_id = clinical_extract.set_index('patientID')

[ ]  1 geneExp_concat_vt_id_class = [geneExp_index_id, clinical_index_id]

[ ]  1 geneExp_vt_id_class = pd.concat(geneExp_concat_vt_id_class, axis=1, join='inner')

[ ]  1 geneExp_vt_id_class = geneExp_vt_id_class.reset_index()
     2 geneExp_vt_id_class = geneExp_vt_id_class.rename(columns={'index': 'patientID'})

[ ]  1 geneExp_vt_id_class
```

	patientID	[geneExp]?|100130426	[geneExp]?|100133144	[geneExp]?|100134869	[geneExp]?|10357	[geneExp]?|10431	[geneExp]?|155060	[geneExp]?|26823	[geneExp]?|280660	[geneExp]?|340602	...	[geneExp]ZXI
0	TCGA.18.3406.01	0.0	0.602408	0.298580	0.542847	0.269766	0.016086	0.000000	0.0	0.000000	...	
1	TCGA.18.3407.01	0.0	0.115225	0.082126	0.349930	0.138950	0.236909	0.000000	0.0	0.000000	...	
2	TCGA.18.3408.01	0.0	0.067855	0.039639	0.482926	0.261043	0.036432	0.208255	0.0	0.000000	...	
3	TCGA.18.3409.01	0.0	0.020701	0.084011	0.269952	0.133157	0.071460	0.120019	0.0	0.000542	...	
4	TCGA.18.3410.01	0.0	0.133214	0.229846	0.355948	0.184224	0.093376	0.000000	0.0	0.000625	...	
...	...	...	...	...	...	...	...	...	...	...	...	
547	TCGA.O2.A52S.01	0.0	0.159473	0.553149	0.299091	0.192018	0.177495	0.193744	0.0	0.000219	...	
548	TCGA.O2.A52V.01	0.0	0.240502	0.179625	0.327427	0.206438	0.114483	0.049173	0.0	0.001111	...	
549	TCGA.O2.A52W.01	0.0	0.428926	0.392123	0.496477	0.095660	0.675989	0.000000	0.0	0.000142	...	
550	TCGA.O2.A5IB.01	0.0	0.580062	0.495378	0.260009	0.016321	0.449478	0.155782	0.0	0.000211	...	
551	TCGA.XC.AA0X.01	0.0	0.357710	0.277503	0.062864	0.099667	0.244860	0.061086	0.0	0.001518	...	

552 rows × 20246 columns

Figure 4.25 Attaching the class label for each omics dataset (using gene expression data as an example)

4.3.2 Exploratory Data Analysis

```
[1]  1 from google.colab import drive
     2 drive.mount("/content/gdrive")

     Mounted at /content/gdrive

[2]  1 import numpy as np
     2 import pandas as pd

[3]  1 location = "/content/gdrive/My Drive/Colab Notebooks/Project/dataset/"

[4]  1 geneExp = pd.read_csv(location + 'exp.csv')
     2 methyl = pd.read_csv(location + 'methyl.csv')
     3 miRNA = pd.read_csv(location + 'mirna.csv')
     4 clinical = pd.read_csv(location + 'clinical.csv')
```

Figure 4.26 Google Colab environment setup, importing libraries and loading datasets

Due to the large size of dataset and the high demand for computational power, Google Colab is used. The very first step is to mount the google drive into Colab which is shown in Cell 1. Next, required libraries such as numpy and pandas are imported as shown in Cell 2. In Cell 3, the location of the datasets is stored into a variable called 'location' for the ease of use. In Cell 4, all 4 datasets are loaded into their respective variables. The program took a while to load all 4 datasets due to their relatively large file size.

```
▾ Check loaded data

[5]  1 print('gene expression: ', geneExp.shape)
     2 print('DNA methylation: ', methyl.shape)
     3 print('miRNA: ', miRNA.shape)
     4 print('clinical data: ', clinical.shape)

     gene expression:  (20531, 553)
     DNA methylation:  (5000, 413)
     miRNA:  (1046, 388)
     clinical data:  (626, 127)
```

Figure 4.27 Code snippet for checking dataset rows and columns

After loading the datasets, their number of rows and columns are briefly checked using the `shape` attribute in pandas. From the output, it can be seen that the gene expression contains 20531 rows and 553 columns. This is the largest dataset file among the 4 datasets. Its size is 87MB, which is significantly larger than the other 3 datasets due to its large number of rows and columns.

The DNA methylation dataset consists of 5000 rows and 413 columns. This is the second largest file among the 4 datasets. The last omics dataset is the miRNA dataset, which is also the smallest dataset among the 4 datasets. Though small, it still consists of 1046 rows and 388 columns. Clinical dataset is another necessary dataset to be used in the study. It consists of 626 rows and 127 features. Despite having large dimension, the clinical dataset is only important for one feature, which is the class of the patients, whether they have lung cancer or not.

```
[6]  1 geneExp.head()
```

	geneID	TCGA.18.3406.01	TCGA.18.3407.01	TCGA.18.3408.01	TCGA.18.3409.01	TCGA.18.3410.01	TCGA.18.3411.01	TCGA.18.3412.01	TCGA.18.3414
0	?|100130426	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0
1	?|100133144	75.3622	14.4148	8.4888	2.5897	16.6653	12.9418	47.5782	12.3
2	?|100134869	25.4252	6.9933	3.3754	7.1538	19.5722	14.4142	20.8881	25.7
3	?|10357	279.9764	198.2540	254.5929	164.3744	200.8033	277.6667	394.3623	205.8
4	?|10431	1510.2362	968.6013	1474.1213	944.6154	1156.0568	1119.5816	861.5023	1030.4

5 rows × 553 columns

Figure 4.28 Code snippet for displaying the first 5 rows of the raw gene expression dataset

```
[7]  1 methyl.head()
```

	geneID	TCGA.18.3417.11	TCGA.18.3417.01	TCGA.18.4721.11	TCGA.18.4721.01	TCGA.18.5592.11	TCGA.18.5592.01	TCGA.18.5595.11	TCGA.18.559
0	cg00001747	0.015847	0.149876	0.029427	0.502545	0.026310	0.707292	0.020024	0.81
1	cg00002719	0.018211	0.698299	0.018122	0.414059	0.020022	0.365875	0.019219	0.76
2	cg00002749	0.947105	0.550682	0.950470	0.674452	0.955821	0.133416	0.949246	0.76
3	cg00003298	0.012583	0.085973	0.015727	0.536764	0.017323	0.735382	0.016739	0.34
4	cg00009292	0.062456	0.559790	0.066961	0.351668	0.069340	0.632683	0.068826	0.66

5 rows × 413 columns

Figure 4.29 Code snippet for displaying the first 5 rows of the raw DNA methylation dataset

```
[8]  1 miRNA.head()
```

	geneID	TCGA.18.5592.01	TCGA.18.5595.01	TCGA.21.5782.01	TCGA.21.5783.01	TCGA.21.5784.01	TCGA.21.5786.01	TCGA.21.5787.01	TCGA.21.A5DI.01
0	hsa-let-7a-1	5998.824696	5348.668096	3829.339596	4114.228785	5035.557137	4015.422861	4258.855065	4928.943949
1	hsa-let-7a-2	11667.530081	10618.397887	7731.024727	8111.916613	10008.799220	7841.828348	8660.778360	9710.772056
2	hsa-let-7a-3	5913.311862	5498.172459	3852.399735	4126.215855	5005.512379	3954.410924	4345.840549	4951.473175
3	hsa-let-7b	16781.396385	24295.057027	13874.291042	12566.112021	23486.098875	8205.507344	7951.159940	17202.190111
4	hsa-let-7c	403.700120	3577.340401	1686.103124	1656.879446	2102.854892	829.642709	371.977398	1283.264683

5 rows × 388 columns

Figure 4.30 Code snippet for displaying the first 5 rows of the raw miRNA dataset

	sampleID	_EVENT	_INTEGRATION	_OS	_OS_IND	_OS_UNIT	_PANCAN_CNA_PANCAN_K8	_PANCAN_Cluster_Cluster_PANCAN	_PANCAN_DNAMethyl_LUSC	_PANCAN_DNAMethyl_PANCAN	_PANCAN_RPPA_PANCAN_K8	_PANCAN_U
0	TCGA-18-3406-01	1.0	TCGA-18-3406-01	371.0	1.0	days	Squamous	C2-Squamous-like	cluster 1	Cluster 13 (HNSC-LUSC)	NaN	
1	TCGA-18-3406-11	1.0	TCGA-18-3406-11	371.0	1.0	days	NaN	C2-Squamous-like	NaN	NaN	NaN	
2	TCGA-18-3407-01	1.0	TCGA-18-3407-01	136.0	1.0	days	High	C2-Squamous-like	cluster 2	Cluster 14 (HNSC)	NaN	
3	TCGA-18-3407-11	1.0	TCGA-18-3407-11	136.0	1.0	days	NaN	C2-Squamous-like	NaN	NaN	NaN	
4	TCGA-18-3408-01	1.0	TCGA-18-3408-01	2304.0	1.0	days	Squamous	C2-Squamous-like	cluster 3	Cluster 13 (HNSC-LUSC)	NaN	

5 rows × 127 columns

Figure 4.31 Code snippet for displaying the first 5 rows of the raw clinical dataset

The first 5 rows of each dataset are displayed as shown above to show a quick preview of their content.

The class balance of each cleaned omics dataset is also observed and depicted in Figure 4.31. The summary of the class label distribution for each omics along with their percentages is tabulated in Table 4.4. From the result, it can be seen that each omics dataset has about 90% of samples with label "Primary Tumour" and 10% of samples with label "Solid Tissue Normal".

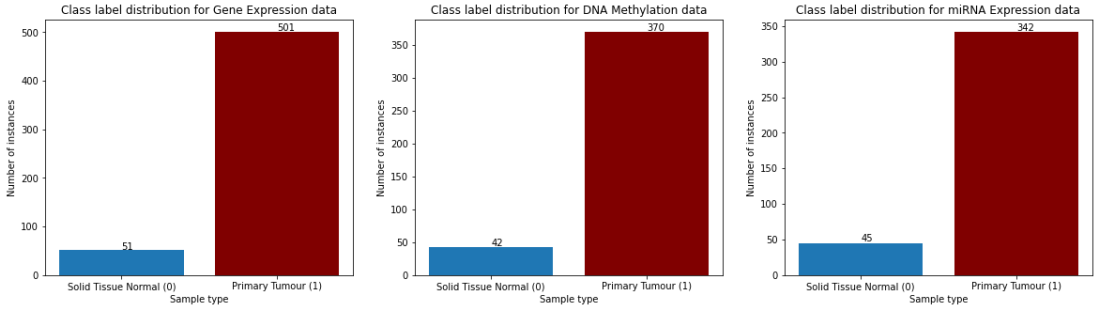


Figure 4.32 Class label distribution for each omics dataset

Table 4.4 Class label distribution with percentage for each omics dataset

Omics	Primary Tumour (1)	Solid Tissue Normal (0)	Total Sample
Gene expression	501 (90.8%)	51 (9.2%)	552
DNA methylation	370 (89.8%)	42 (10.2%)	412
miRNA expression	342 (88.4%)	45 (11.6%)	387

4.3.3 Multi-omics Data Integration

With the cleaned and properly labelled patient ID in each dataset, the multi-omics data integration can be carried out. First of all, the patient ID of each dataset are set as the index of the dataframe. Then, the dataframes are combined together based on the index of the dataframe which is the patient ID. The result of the concatenation is shown in Figure 4.33. It can be seen that after the concatenation, only 344 rows remain while the number of columns increased to 26131 including the output class and excluding the patient ID column.

```
[ ]  1 geneExp_momics = geneExp_vt_id.set_index('patientID')
     2 methyl_momics = methyl_vt_id.set_index('patientID')
     3 miRNA_momics = miRNA_vt_id.set_index('patientID')
     4 clinical_momics = clinical_extract.set_index('patientID')

[ ]  1 concat_momics = [geneExp_momics, methyl_momics, miRNA_momics, clinical_momics]

[ ]  1 multiOmics = pd.concat(concat_momics, axis=1, join='inner')

[ ]  1 multiOmics = multiOmics.reset_index()
     2 multiOmics = multiOmics.rename(columns={'index': 'patientID'})

[ ]  1 multiOmics
```

	patientID	[geneExp]?|100130426	[geneExp]?|100133144	[geneExp]?|100134869	[geneExp]?|10357	[geneExp]?|10431	[geneExp]?|155060	[geneExp]?|26823	[geneExp]?|280660	[geneExp]?|340602	...	[miRNA]hsa-mir-941-1
0	TCGA.18.5592.01	0.0	0.276995	0.159267	0.440993	0.180608	0.067246	0.042749	0.000000	0.000097	...	0.0
1	TCGA.18.5595.01	0.0	0.243614	0.194970	0.461358	0.146396	0.051615	0.000000	0.000000	0.000075	...	0.0
2	TCGA.21.5782.01	0.0	0.066732	0.337394	0.610301	0.128080	0.113349	0.131487	0.099856	0.004158	...	0.0
3	TCGA.21.5783.01	0.0	0.434786	0.329469	0.336864	0.032273	0.116544	0.101916	0.000000	1.000000	...	0.0
4	TCGA.21.5784.01	0.0	0.055087	0.256713	0.263417	0.193994	0.038431	0.000000	0.000000	0.000000	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
339	TCGA.O2.A52S.01	0.0	0.159473	0.553149	0.299091	0.192018	0.177495	0.193744	0.000000	0.000219	...	0.0
340	TCGA.O2.A52V.01	0.0	0.240502	0.179625	0.327427	0.206438	0.114483	0.049173	0.000000	0.001111	...	0.0
341	TCGA.O2.A52W.01	0.0	0.428926	0.392123	0.496477	0.095660	0.675989	0.000000	0.000000	0.000142	...	0.0
342	TCGA.O2.A5IB.01	0.0	0.580062	0.495378	0.260009	0.016321	0.449478	0.155782	0.000000	0.000211	...	0.0
343	TCGA.XC.AA0X.01	0.0	0.357710	0.277503	0.062864	0.099667	0.244860	0.061086	0.000000	0.001518	...	0.0

344 rows × 26132 columns

Figure 4.33 Code snippet for multi-omics integration

Similarly, the class balance of the integrated multi-omics dataset is observed and depicted in Figure 4.34. The summary of the class label distribution for the integrated multi-omics dataset along with their percentages is tabulated in Table 4.5. From the result, it can be seen that the multi-omics dataset is severely imbalanced. It has 99.1% of samples with label "Primary Tumour" and only a mere 0.9% of samples with label "Solid Tissue Normal".

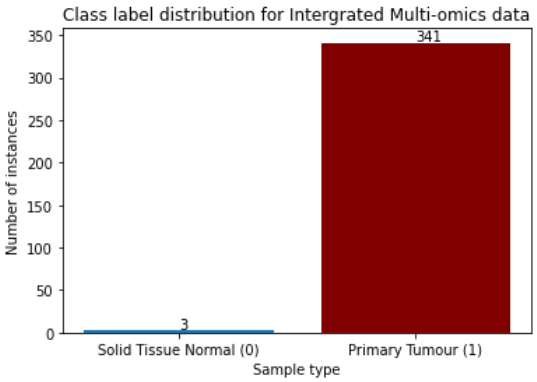


Figure 4.34 Class distribution of the integrated multi-omics data

Table 4.5 Class label distribution with percentage for each omics dataset

Omics	**Primary Tumour (1)**	**Solid Tissue Normal (0)**	**Total Sample**
Gene expression	501 (90.8%)	51 (9.2%)	552
DNA methylation	370 (89.8%)	42 (10.2%)	412
miRNA expression	342 (88.4%)	45 (11.6%)	387
Multiomics Data	**341 (99.1%)**	**3 (0.9%)**	**344**

4.3.4 Feature Selection

Feature selection is a necessary process in this study. The dataset involved, which is the multi-omics data, is enormous with 26130 omics features when all three omics datasets are concatenated. Feature selection aims to reduce the dimensionality of the data by selecting the best performing subset from the entire dataset before training the classifier.

In this study, SVM-RFE is the chosen feature selection algorithm. SVM-RFE is simply an RFE method which uses SVM as the estimator or predictor to aid the feature selection. In SVM-RFE, the process starts with training the selected estimator, which in this study, the SVM classifier, using the entire original number of features in the multi-omics data. The trained classifier then calculates the weight of each features and subsequently compute the ranking criteria for the particular iteration. The feature(s) with the lowest ranking will be eliminated.

The process continues with subsequent iterations by training a new SVM model using the remaining features. When a certain stopping criterion is met, the process ceases and return a list of features with the highest rank.

Algorithm: SVM-RFE

Input : Training data (X, y)

Features/Attributes : $X = [x_1, x_2, x_3, \dots, x_n]$

Class label : $y = [y_1, y_2, y_3, \dots, y_n]$

Output : Ranked feature R

Initialize:

Remaining subset of features S = X

Ranked features R = []

Number of features to keep N, where N < |X|

Number of features to eliminate per iteration K

While |S| != N, do:

1. Train the SVM model using the remaining subset of features S
2. Obtain the weight vector of each feature from the trained SVM model
3. Calculate the ranking criteria for each feature $c_i = (w_i)^2, i = 1, 2, \dots, |S|$
4. Look for feature(s) with lowest ranking criteria, $l = \min(c)$
5. Append feature l into ranked feature R
6. Remove feature l from S

The implementation of SVM-RFE for feature selection starts with data splitting. As mentioned in Chapter 3.4, the multi-omics dataset is split into train and test data with distribution of 70% and 30% respectively. Empirically, it is known that

train-test split with distribution of 20%-30% and 70%-80% helps the model to obtain the best results (Gholamy, Kreinovich, & Kosheleva, 2018). The predictors and the class label are first separated into variables X and y respectively as shown in Figure 4.35.

```
[ ]  1 X = pd.DataFrame(momics.iloc[:, :-1])
     2 y = pd.DataFrame(momics.iloc[:, -1])
```

Figure 4.35 Extracting omics features and class label

Then, the train_test_split function is imported from the sklearn library. The data splitting process starts with parameter test_size set to 0.3, indicating that the test data will have 30% of the features from the full multi-omics data. Stratification is also used so that the class labels are split equally into train and test split. The splitting is also seeded so that the results are reproducible for multiple runs. The implementation of data splitting is shown in Figure 4.36.

```
[ ]  1 from sklearn.model_selection import train_test_split

[ ]  1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=42, stratify = y)
```

Figure 4.36 Code snippet for train test split on the multi-omics dataset

The dimension of the train and test data after the splitting process is displayed for reference as shown in Figure 4.37. In order to visualize the dimension of the train and test data, bar charts are used plot for both the train and test data. The code snippet for the visualization is shown in Figure 4.38.

```
[ ]  1 X_class = Counter(y_train['class'])
     2 X_class

     Counter({0: 2, 1: 238})

[ ]  1 y_class = Counter(y_test['class'])
     2 y_class

     Counter({0: 1, 1: 103})
```

Figure 4.37 Code snippet to display the dimension of the train and test data

```
[ ]  1 class_name = np.sort(pd.unique(momics['class']))
     2 class_name = np.where(class_name == 0, 'Solid Tissue Normal (0)', 'Primary Tumour (1)')
     3
     4 fig = plt.figure(figsize=[15, 5])
     5
     6 plt.subplot(121)
     7 plt.bar(class_name, [X_class[0], X_class[1]], color = ['tab:blue', 'maroon'])
     8 plt.text(0, X_class[0] + 2, s=X_class[0])
     9 plt.text(1, X_class[1] + 2, s=X_class[1])
    10 plt.xlabel("Sample type")
    11 plt.ylabel("Number of instances")
    12 plt.title("Class label distribution for Multi-omics Train Data")
    13
    14 plt.subplot(122)
    15 plt.bar(class_name, [y_class[0], y_class[1]], color = ['tab:blue', 'maroon'])
    16 plt.text(0, y_class[0] + 2, s=y_class[0])
    17 plt.text(1, y_class[1] + 2, s=y_class[1])
    18 plt.xlabel("Sample type")
    19 plt.ylabel("Number of instances")
    20 plt.title("Class label distribution for Multi-omics Test Data")
    21
    22 plt.show()
```

Figure 4.38 Code snippet to visualize the dimension of the train and test data

From the results in Figure 4.39, the train data consists of a total of 240 instances, in which 2 of those are of class "0" or "Solid Tissue Normal" and 238 of class "1" or "Primary Tumour". The test data on the other hand has 1 instance for class "0" and 103 instances for class "1", a total of 104 instances. Since the original multi-omics data is severely imbalanced, the train and test data split from it are also inherently severely imbalanced. For train data, the class imbalance problem can be solved with sampling techniques such as SMOTE, which will be applied in Chapter 4.3.5 before feeding it to the deep learning models. Test data on the other hand is fixed and cannot be oversampled.

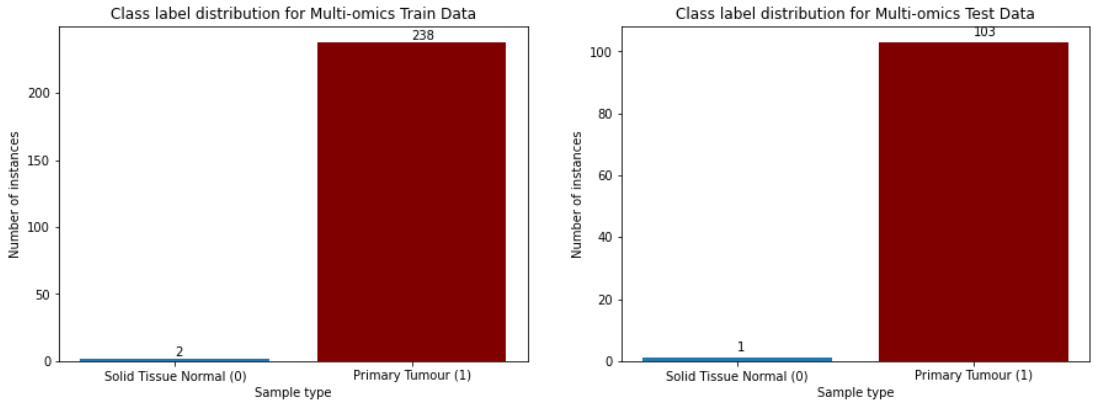


Figure 4.39 Class label distribution for the train and test data

In previous study by (Guyon et al., 2002), several subsets with decreasing feature size in power of 2 were selected using RFE, which is from 4096 features to 2 features. In this study, similar strategy is employed. However, instead of decreasing the feature size in power of 2, a thousand of features are removed per iteration. The

integrated multi-omics dataset is fed into the SVM-RFE algorithm to extract several feature subsets, ranging from 20,000 to 1,000 features. The workflow of SVM-RFE for this study is depicted in Figure 4.40.

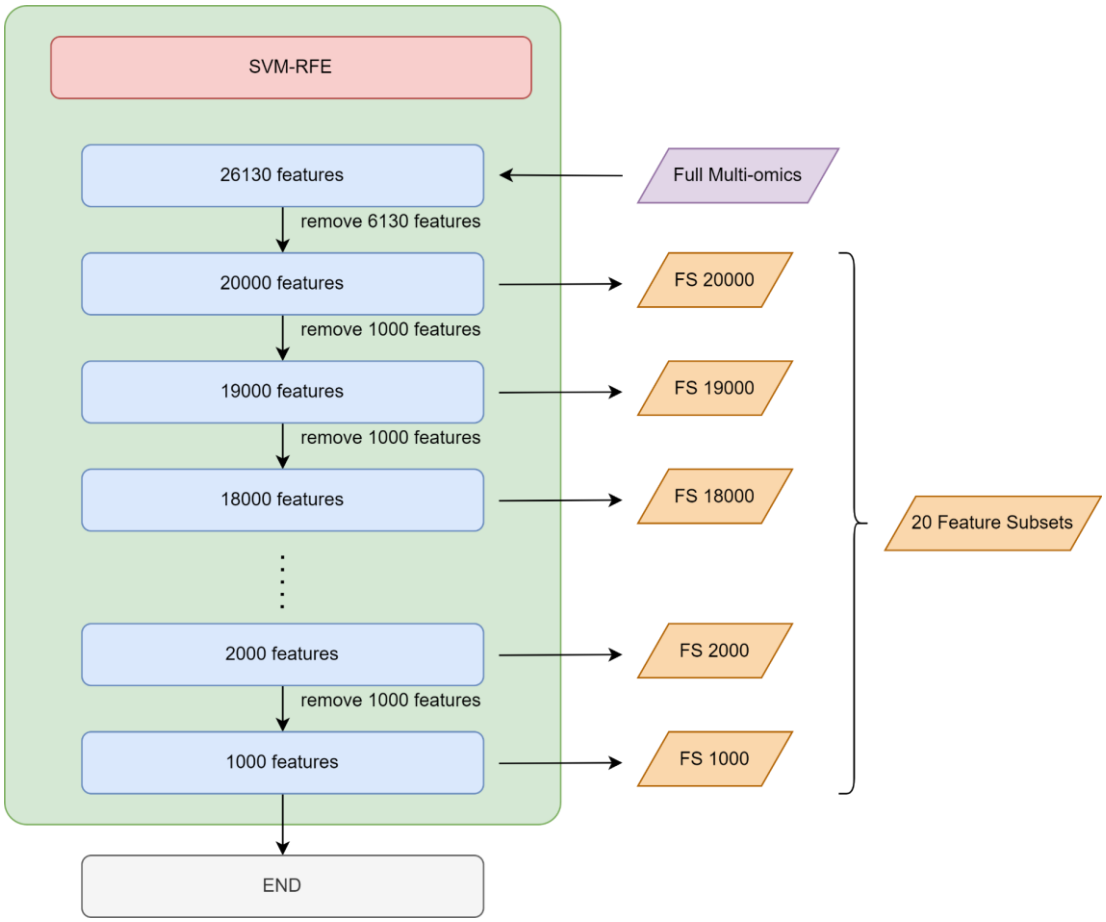


Figure 4.40 The workflow/step for the SVM-RFE algorithm for extracting feature subsets

The number of features to be extracted are stored into the "n_features" variable as shown in Figure 4.41. At the end of this phase, the extracted feature subsets are saved as separate files in form of CSV format for deep learning classification in the next phase.

```
[ ]  1 n_features = [i * 1000 for i in range(1, 21)]
     2 n_features = n_features[::-1]
```

Figure 4.41 Setting the number of feature subsets to extract using SVM-RFE

The complete SVM-RFE implementation is shown in Figure 4.42 and 4.43. Firstly, the necessary variables are prepared in Figure 4.42, such as the start time, the train and test data, the cross-validation folds, the parameter grid. Other than that, several variables are also initialized to keep track of the outputs and results of SVM-RFE. The rest of the implementation of SVM-RFE is shown in Figure 4.43.

```
[ ]
print('momics: accuracy')
start_time_whole = datetime.now()

X_remaining = X_train
X_test_remaining = X_test
cv = StratifiedKFold(n_splits=2)
param_grid = {
  'estimator__C': [1e-1, 1, 1e1, 1e2],
  'estimator__kernel': ['linear'],
  'step': [1, 2, 3]
}
scorer = make_scorer(recall_score,pos_label=0)

cross_val_scores = list()
supports = dict()
train_fs = dict()
test_fs = dict()
best_params = dict()

X_previous = dict()
X_test_previous = dict()
X_remaining_iters = dict()
```

Figure 4.42 The SVM-RFE algorithm – variables initialization

Before running the SVM-RFE function, the most optimal set of hyperparameters need to be determined. Grid search is used determine such optimal hyperparameters for each feature subset, which is done using the `gridsearchCV` function from the `sklearn` library. The grids were initialized as follow in Table 4.6:

Table 4.6 Hyperparameter grids used for SVM-RFE

Hyperparameter	**Values**
C	0.1, 1, 10, 100
Kernel	linear
Step	1, 2, 3

The first two hyperparameters (C & Kernel) in Table 4.6 are for the SVM classifier in SVM-RFE. C refers to the penalty parameter of the error term in the SVM algorithm. It controls the tradeoff between the correct classifications and the smoothness of the drawn decision boundary. According to the study done by Huang et al. (2014), the range of the C hyperparameter used is $C = \{1, 10, 50, 100\}$. The range used for the C hyperparameter refers to this study with some tweaks by

involving $C = 0.1$ to introduce a softer decision boundary margin of the SVM algorithm in RFE. As for the SVM's kernel, linear kernel is used as it is the only kernel that produces feature importance as one of its output, which is necessary for RFE to rank the features.

The Step hyperparameter belongs to the RFE algorithm, whereby it refers to the number of features to eliminate in every iteration. According to Guyon et al. (2002), RFE can remove more than one feature at a time. When computation cost is concerned, it may be more efficient to eliminate more than one feature per iteration. However, removing more than one feature per iteration could lead to potential degradation of classification performance. By considering the large dimension of the multi-omics data used in this study, the number of features to remove is set to $step = \{1, 2, 3\}$ to explore the possibility of more efficient feature selection process with minimal tradeoff of performance.

```
[ ] 24 for n in n_features:
    25
    26   start_time = datetime.now()
    27
    28   X_prev = X_remaining
    29   X_previous[str(n)] = X_remaining
    30   X_test_prev = X_test_remaining
    31   X_test_previous[str(n)] = X_test_remaining
    32
    33   rfe = RFE(estimator=SVC(), n_features_to_select=n)
    34
    35   search = GridSearchCV(estimator=rfe, param_grid=param_grid, cv=cv, scoring='accuracy', verbose=3, n_jobs=-1, error_score='raise')
    36   search_result = search.fit(X_remaining, np.ravel(y_train))
    37   best_model = search_result.best_estimator_
    38
    39   best_params[str(n)] = search_result.best_params_
    40   supports[str(n)] = best_model.get_support()
    41
    42   X_remaining = best_model.transform(X_remaining)
    43   X_test_remaining = best_model.transform(X_test_remaining)
    44
    45   X_remaining = pd.DataFrame(X_remaining, columns=[X_prev.columns[i] for i in range(len(X_prev.columns)) if supports[str(n)][i]], index=X_prev.index)
    46   X_test_remaining = pd.DataFrame(X_test_remaining, columns=[X_test_prev.columns[i] for i in range(len(X_test_prev.columns)) if supports[str(n)][i]], index=X_test_prev.index)
    47
    48   train_fs[str(n)] = X_remaining
    49   test_fs[str(n)] = X_test_remaining
    50
    51   iter_C = best_params[str(n)]['estimator__C']
    52   iter_kernel = best_params[str(n)]['estimator__kernel']
    53
    54   clf = SVC(C=iter_C, kernel=iter_kernel)
    55
    56   iter_cross_val_scores = cross_val_score(clf, X_remaining, np.ravel(y_train), scoring=scorer, cv=cv, n_jobs=-1)
    57   cross_val_scores.append(iter_cross_val_scores)
    58
    59   print('%s features => mean=%.3f, std=%.3f, cfg=%s' % (str(n), mean(iter_cross_val_scores), std(iter_cross_val_scores), best_params[str(n)]))
    60
    61   end_time = datetime.now()
    62   print('Duration: {}'.format(end_time - start_time))
    63   print()
    64
    65 end_time_whole = datetime.now()
    66 print('Overall Duration: {}'.format(end_time_whole - start_time_whole))
```

Figure 4.43 The SVM-RFE algorithm

Besides that, stratified K fold cross validation method is also applied to produce a more generalizable result. This is done by using the `StratifiedKFold` function from `sklearn`. In the StratifiedKFold function, the n_splits parameter is set to 2 to produce two folds in extracting each feature subset. The number of folds used in K Fold CV is usually between 5 to 10 (M. Kuhn & Johnson, 2013), but for

this specific dataset, the minority class (Solid Tissue Normal) has only 2 samples. This limits the number of folds in K Fold CV to a maximum fold of 2 to ensure that each fold has at least one sample with the minority class. If K Fold CV with more than 2 folds are used, for example $k = 3$, one of the folds will contain only one class, which is the majority class (Primary Tumour), and the StratifiedKFold function from the sklearn library will display an error message.

For the purpose of keeping record for reference, result of cross validation, the best hyperparameter grid, and the time required to run the algorithm are recorded for each feature subset. The mean and standard deviation for each feature subset is used to plot a boxplot for visualization.

Figure 4.44 Code snippet to plot the boxplot for the cross-validation score from SVM-RFE for each feature subset

In addition, the composition/distribution of each omics in the integrated multi-omics data for each feature subset after the feature selection process is also recorded. This serves as a purpose to determine the importance of the omics features via feature ranking using SVM-RFE. For example, if the observed composition of gene expression omics decreased after feature selection, it could be an indication that gene expression omics features might be less significant compared to the other two omics features (DNA methylation & miRNA expression).

Figure 4.45 Code snippet to display the omics distribution for each feature subset

To better visualize the composition of omics for each feature subset, a composite bar plot is created. Figure 4.46 shows the implementation of visualizing the omics composition in form of bar chart.

```
[84]  1 labels = selected
      2
      3 x = np.arange(len(labels))
      4 width = 0.5
      5
      6 fig, ax = plt.subplots(dpi=400)
      7
      8 geneExpBar = ax.bar(x, geneExp_feature_subset_percentage, width, label='Gene Exp.', color='tab:blue')
      9 methylBar = ax.bar(x, methyl_feature_subset_percentage, width, label='DNA Methylation', color='maroon')
     10 miRNABar = ax.bar(x, miRNA_feature_subset_percentage, width, label='miRNA Exp.', color='green')
     11
     12 fig.set_figheight(10)
     13 fig.set_figwidth(20)
     14 ax.set_xlabel('Feature Subsets')
     15 ax.set_ylabel('Omics composition (%)')
     16 ax.set_title('Omics Composition for each Feature Subset')
     17 ax.set_xticks(x)
     18 ax.set_xticklabels(labels)
     19
     20 ax.legend(loc='center right')
     21
     22 ax.spines['top'].set_visible(False)
     23 ax.spines['right'].set_visible(False)
     24 ax.spines['left'].set_visible(False)
     25 ax.spines['bottom'].set_color('#DDDDDD')
     26 ax.tick_params(bottom=False, left=False)
     27 ax.set_axisbelow(True)
     28 ax.yaxis.grid(True, color='#EEEEEE')
     29 ax.xaxis.grid(False)
     30
     31 for bar in ax.patches:
     32   bar_value = bar.get_height()
     33   text = f'{bar_value:,}'
     34   text_x = bar.get_x() + bar.get_width() / 2
     35   text_y = bar.get_y() + bar_value
     36   bar_color = bar.get_facecolor()
     37   ax.text(text_x, text_y, text, ha='center', va='bottom', color=bar_color,
     38           size=12, path_effects=[pe.withStroke(linewidth=1.5, foreground="white")], weight='bold')
     39
     40 fig.tight_layout()
     41
     42 plt.show()
```

Figure 4.46 Code snippet for visualizing omics distribution after SVM-RFE

4.3.5 Synthetic Minority Oversampling Technique (SMOTE)

Before implementing the autoencoders, SMOTE is performed with the hyperparameters shown in Table 4.7. "sampling_strategy" is set to 1, in which the instances for the minority class (i.e. "0" or "Solid Tissue Normal") is synthetically sampled to match the number of instances for the majority class (i.e. "1" or "Primary Tumour"). "k_neighbors" is set to 1. This is due to the fact that the train data only has two (2) instances with class label "0", and the value for "k_neighbors" must be less than the number of instances. "random_state" is set to 42 so that the result can be made reproducible.

Table 4.7 Hyperparameters used for SMOTE on train data

Hyperparameters	**Settings**
sampling_strategy	1
k_neighbors	1
random_state	42

The class distribution of the train data before and after SMOTE is shown in Figure 4.47. Before SMOTE, the train data had 2 instances for class label "Solid Tissue Normal" or "0", and 238 instances for class label "Primary Tumour" or "1". After SMOTE, the class distribution for both labels are now equal at 238 instances.

Figure 4.47 The class distribution for the train data before and after SMOTE

The comparison between the class distribution for the final train and test data is shown in Figure 4.48. The minority class label of the train data has been successfully oversampled to match the number of instances of the majority class for class balancing. The test data on the other hand is still severely imbalanced.

Figure 4.48 The final class distribution for the train and test data

4.3.6 Deep Learning Classification Models

The implementation of the deep learning classification models is split into two parts, unsupervised learning and classification. Both SDAE and VAE models undergo the same unsupervised learning process. However, each of them incorporates a different classification phase. For context, the difference in classification phase is due to the different loss function used in developing the autoencoders. The SDAE model with binary cross entropy as its loss function is able to undergo fine-tuning process whereby the decoder part of the autoencoder is replaced with a sigmoid layer with one node for classification. The VAE model with the Kullback-Leibler divergence loss function is not suitable for such approach (refer to Table 5.8 & Figure 5.11). Therefore, an external classification model is used to classify the inputs encoded in the latent layer (details in Chapter 4.3.6.2).

In the unsupervised learning part, the autoencoders are constructed with the typical structure of an autoencoder, which consists of the encoder, the bottleneck or latent space, and the decoder. The main objective of the unsupervised training is to obtain the overall reconstruction loss of the autoencoders with the selected hyperparameters. The reconstruction loss is then used as a metric to determine whether or not the developed model is suitable for classification. The reconstruction loss can be used to measure the ability of the autoencoder to reconstruct or reproduce the given input. The lower the reconstruction loss, the better the ability of the autoencoder to capture the important features of the input data and reconstruct them with output close to the original input.

In this study, the autoencoders are constructed using the keras library with tensorflow backend on Google Colab. Runtime with GPU is utilized to drastically speed up the training process of the deep learning model. The GPU offered by Google Colab varies depending on the availability. Most of the time, GPU with model Nvidia Tesla K80 is offered while the better and faster Nvidia Tesla K4 is offered rarely from time to time.

In previous study involving the use of SVM-RFE on Leukemia data by (Guyon et al., 2002), it is found out that the most optimal feature subsets are the fourth and the fifth smallest feature subsets. At the same time, due to hardware limitation (i.e. insufficient VRAM for GPU offered by Google Colab), the study limits the feature subsets to be used for classification to only the five (5) smallest feature subsets, namely feature subset 1000 to feature subset 5000. Figure 4.49 shows the summary of the neural network constructed for feature subset 20000 during unsupervised training. It can be seen that there is a total of nearly a billion of parameters for a single neural network.

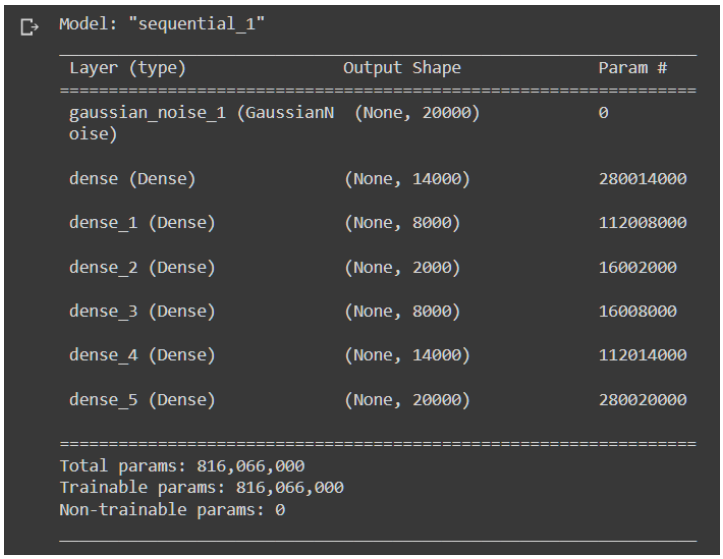


Figure 4.49 The summary of the neural network constructed for feature subset 20000 during unsupervised training

Figure 4.50 shows the memory consumption after creating the neural network for feature subset 20000 during unsupervised training. This model alone occupied nearly 90% of the GPU's memory. The issue with insufficient GPU memory prevented the study to proceed with the fine-tuning process, which also requires large amount of GPU memory.

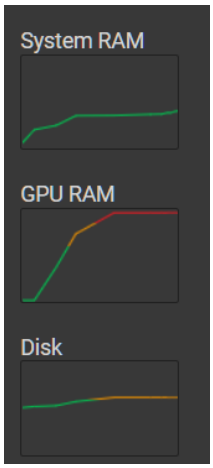


Figure 4.50 The system information shown provided by Google Colab after constructing the neural network for feature subset 20000 in unsupervised learning phase

4.3.6.1 Stacked Denoising Autoencoder (SDAE)

SDAE is a deep learning model which comprises of multiple Denoising Autoencoder (DAE). In general, each DAE in SDAE will have 3 layers, which are the input layer, one hidden layer and the output layer. The DAE in SDAE takes the output from the output layer of the previous DAE as input, while also passing their output to the input layer of the subsequent DAE. The structure of the SDAE model can be broken down into two parts, which are the encoding and decoding parts.

In the encoding part, the original input x is corrupted using stochastic mapping function $\tilde{x} \sim q_D(\tilde{x}|x)$. The corrupted input x is now termed as $\tilde{x}$. The corrupted input $\tilde{x}$ is then mapped to a hidden representation $h = f_\theta(\tilde{x}) = \sigma(W\tilde{x} + b)$. On the other hand, the hidden representation in the decoding part reconstruct z = $g_{\theta'}(h) = \sigma'(W'h + b')$. The symbol σ (σ') refers to the activation function which could either be the sigmoid function or the rectified linear unit. W (W') refers to the weight matrix while b (b') refers to the bias vector.

Both W (W') and b (b') are the parameter of the model θ (θ'), which can be defined as $\theta = \{W, b\}$ and $\theta' = \{W', b'\}$. Both θ and θ' are trained to minimize the

reconstruction error, or also known as the squared error (Jiang, 2021). The reconstruction error can be represented as follow:

$$L(x, z) = ||x - z||^2 = ||x - \sigma'(W'\sigma'(W\tilde{x} + b) + b')||^2 \quad (4.1)$$

First and foremost, the SDAE modelling starts with unsupervised learning, whereby the ultimate goal of this training is to obtain the reconstruction loss of the model to determine its capability in reconstructing the given input. Figure 4.51 shows the neural network structure of the SDAE model during the unsupervised training phase. Similar to a regular autoencoder, the SDAE model has three major components, which are the encoding layer, the latent layer, and the decoding layer. Since it is an SDAE model, there consists of multiple encoders and decoders in their respective layers. For instance, in this study, there are three encoders and three decoders in the encoding layer and the decoding layer respectively.

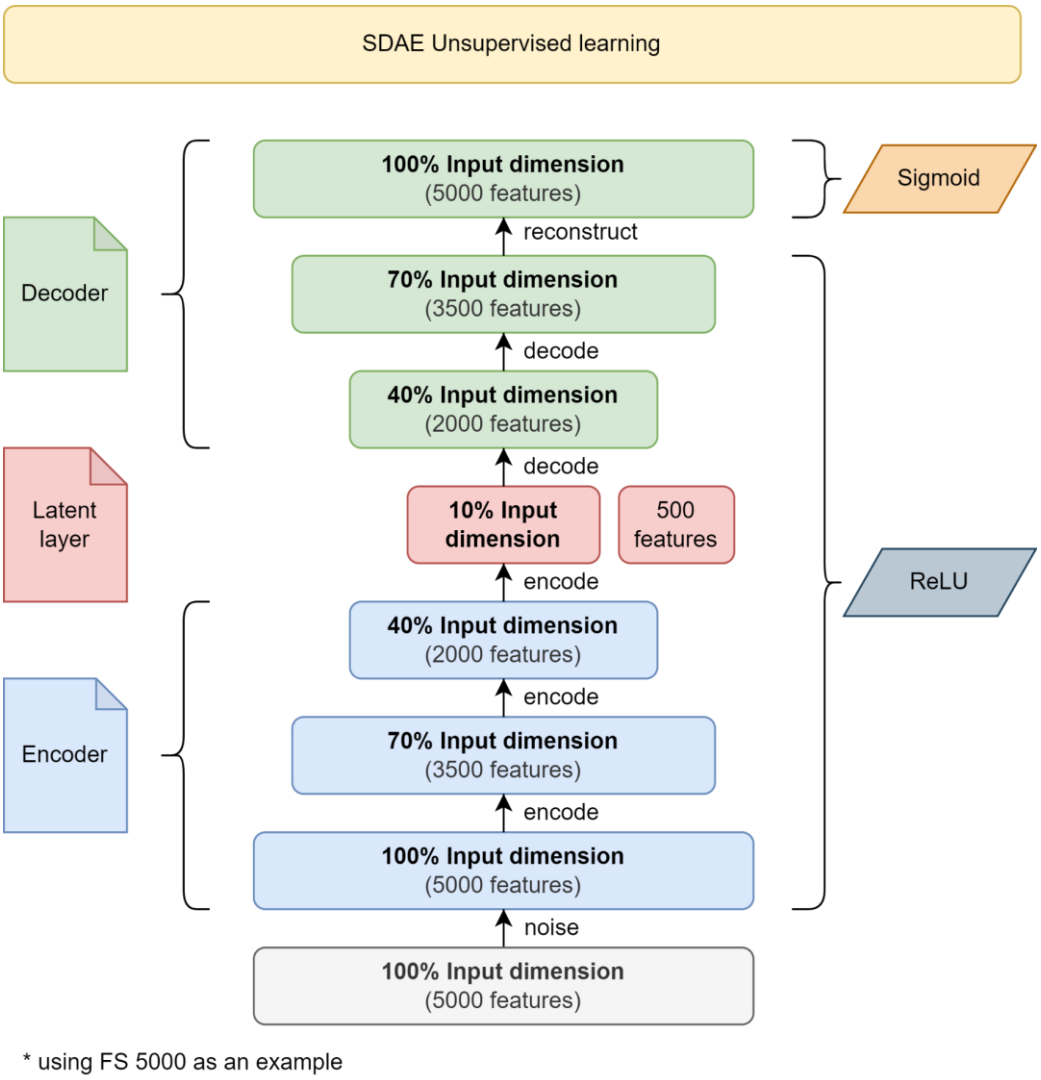


Figure 4.51 The neural network structure of the SDAE model for unsupervised training

Next, the hyperparameters of the SDAE model for the unsupervised training need to be determined. The hyperparameters used in the construction of the SDAE model are tabulated in Table 4.8. The number of layers is one of the most important hyperparameter in neural network. First of all, the SDAE network consists of eight (8) layers with symmetrical dimensions (ignoring the gaussian noise layer). In this study, the dimension of each layer is determined by different fractions of the original input dimension. Noise is introduced to the original input. For the first layer in the encoding layer, which is the input layer, it consists of 100% of the input dimension. The second layer in the encoding layer, consists of 70% of the input dimension, followed by 40% of the input dimension for the third layer. The latent layer contains 10% of the input dimension, or in other words, the original input is encoded and lost

90% of its features during the process. Next, the first layer in the decoding layer consists of 40% of the input dimension, while the second and third layer consists of 70% and 100% of the input dimension respectively. In the last layer, which is also the third layer in the decoding layer, the dimension is set back to 100% of the input layer in order to reconstruct the encoded features in the latent layer back to its original dimension. An example (coloured grey) for the actual dimension for each layer in the SDAE model is also shown in Table 4.8 by using an input dimension of 5000 features as an example.

Table 4.8 Hyperparameters used for SDAE unsupervised training

Hyperparameters	**Description**
Layers	*5000, 5000 (noisy), 3500, 2000, 500, 2000, 3500, 5000* 100%, 100% (noisy), 70%, 40%, 10%, 40%, 70%, 100%
Epoch	50
Batch size	16
Optimizer	adam
Activation functions	ReLU (Rectified Linear Unit) – Hidden layers Sigmoid – Last layer (output layer)
Loss function	binary cross entropy
Gaussian Noise Dropout Rate	10%

The epoch of the neural network represents the number of iterations for the training of the models using the training set, while the batch size refers to the number of samples processed by the model before updating the model (Keskar, Mudigere, Nocedal, Smelyanskiy, & Tang, 2016). The epoch used in this study is 50. From the results obtained using 50 epochs for training, it can be seen that the model loss can be minimized and converged within 50 epochs for each feature subset, thus considered optimal. The batch size is set to relatively small at 16, as large batch size is observed to produce degradation in the model (Keskar et al., 2016). An activation function aims to optimize the model by adjusting the weights and the learning rate (Shaziya, 2020). "adam" optimizer is used as it is currently the most recommended optimizer and is being adapted as the benchmark for deep learning models (Gupta,

2021). With reference to the work by Hira et al. (2021), this study adapts the same activation functions, that is, ReLU for the hidden layers and sigmoid for the last layer (output layer).

A loss function is also known as the cost function, which is a method to evaluate the wellness of a model (Gordon-Rodriguez, Loaiza-Ganem, Pleiss, & Cunningham, 2020). Cross entropy is one of the most used loss functions in classification problems, and binary cross entropy is a variant of cross entropy that outputs value between 0 and 1 for binary classification (Ruby & Yendapalli, 2020). Therefore, binary cross entropy is used as the loss function as the concern of this study is related to binary classification. A Gaussian Noise layer with 10% dropout rate is introduced to the original input to force the SDAE model to learn the corrupted features for reconstruction (Gondara, 2016).

As mentioned in Chapter 4.3.5, the SDAE model undergoes the fine-tuning process with supervised learning. The hyperparameters used in the previous unsupervised learning phase are carried forward to the supervised learning phase as shown in Table 4.9. However, the decoder part of the autoencoder is removed and instead, a sigmoid layer with one node is attached after the latent layer to replace the decoder. The sigmoid layer with a single node allows the autoencoders to perform binary classification by producing outputs with values between 0 and 1. In this study, 0 represents "Solid Tissue Normal" (non-cancerous) while 1 represents "Primary Tumour" (cancerous). However, since the sigmoid layer outputs values between 0 and 1 instead of 0 or 1, the values need to be rounded off to 0 or 1, in which the values smaller than 0.5 are rounded to 0, and the values larger than 0.5 are rounded to 1. The neural network structure of the SDAE model in the supervised learning phase is shown in Figure 4.52.

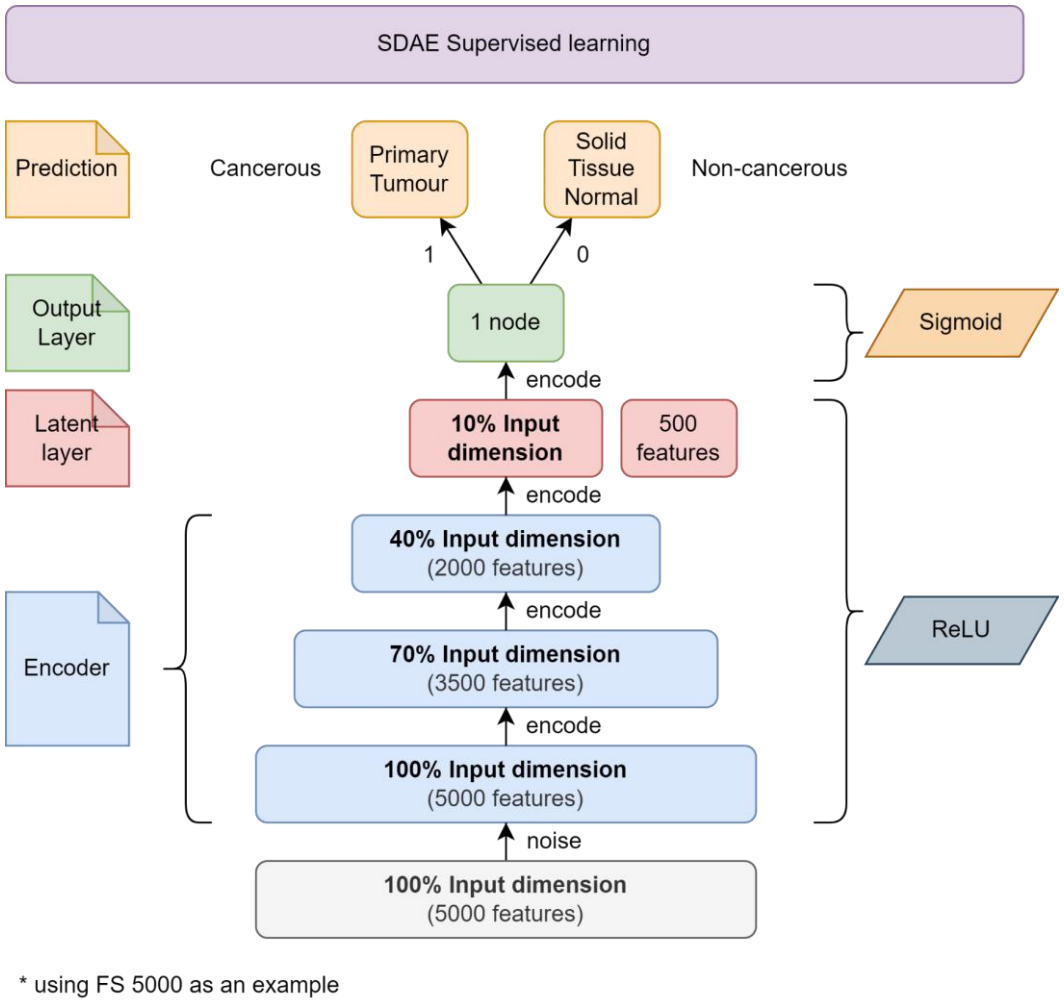


Figure 4.52 The neural network structure of the SDAE model for supervised training (fine-tuning)

Table 4.9 Hyperparameters used for SDAE supervised training (fine-tuning)

Hyperparameters	**Description**
Layers	*5000, 5000 (noisy), 3500, 2000, 500, 1* 100%, 100% (noisy), 70%, 40%, 10%, 1
Epoch	50
Batch size	16
Optimizer	adam
Activation functions	ReLU (Rectified Linear Unit) – Hidden layers Sigmoid – Last layer (output layer)
Loss function	binary cross entropy
Gaussian Noise Dropout Rate	10%

4.3.6.2 Variational Autoencoder (VAE)

VAE is another deep learning model which is a variant of autoencoder. It is capable of learning impactful data manifold from high dimensional input data. In contrast to a traditional autoencoder, VAE encodes input as a distribution over a latent space instead of a single point (Hira et al., 2021). Similar SDAE, VAE also have 3 primary layers which are the input layer, hidden layers and the output layer, and also the encoding and decoding parts. There are several steps involved in building a VAE model.

The first step is encoding. In the encoding part, the encoder takes the input and encodes them into latent variables z^i using the latent distribution $p_\theta(z)$. The encoder also creates a variational distribution $q_\Phi(z|x)$ which is also known as the encoding distribution. This distribution is created to estimate the posterior and address the intractability of the true posterior $p_\theta(z|x)$ in calculating the distribution of X or $p_\theta(X)$ (Hira et al., 2021).

The next step involves the sampling part, where a sampler samples points from the latent space through the encoding distribution $q_\Phi(z|x)$. The sampled points are then passed to the next part which is the decoding part (Hira et al., 2021).

The third step in VAE is the decoding process. In the decoding part, the decoder decodes the sampled points from the previous step using a conditional distribution $p_\theta(x|z)$. The purpose of this is to reconstruct the input x'. The symbol θ in $p_\theta(x|z)$ refers to the set of learnable parameters of the decoder. In the decoding process, the loss or error of the reconstruction is calculated using the loss function which consists of the reconstruction term and regularization term. The reconstruction term is responsible of calculating the reconstruction loss while the regularization term regularizes the latent space by quantifying the distance between the estimated posterior $q_\Phi(z|x)$ and the true posterior $p_\theta(z|x)$. Generally, the regularization term in most VAE utilizes the Kullback-Leibler divergence to optimize the encoder and decoder using the following loss function:

$$L_{VAE} = E_{q_\Phi(Z|X)}[logp_\theta(x|z)] - D_{KL}(q_\Phi(z|x)||p_\theta(z)) \quad (4.2)$$

which uses the traditional evidence lower bound (ELBO) criterion. The term D_{KL} is the Kullback-Leibler (KL) divergence between two probability distributions (Hira et al., 2021). The last step in VAE involves the backpropagation process, in which the calculated loss is back-propagated through the neural network to update the model.

Similar to the development of the SDAE model, the VAE modelling also starts with unsupervised learning, with the same objective of obtaining the reconstruction loss of the model to assess the ability of the model to reconstruct the given input. Figure 4.53 shows the neural network structure of the VAE model during the unsupervised training phase. The VAE model consists of four (4) parts instead of three (3) compared to the SDAE model. In addition to the encoding layer, the latent layer and the decoding layer, the VAE model has a sampling layer before the latent layer. Similarly, there are three (3) encoders and three (3) decoders in the encoding and decoding layers respectively. The sampling layer obtains the probability distribution of the mean and variance from the encoded inputs and generates output to the latent layer using the same probability distribution via sampling.

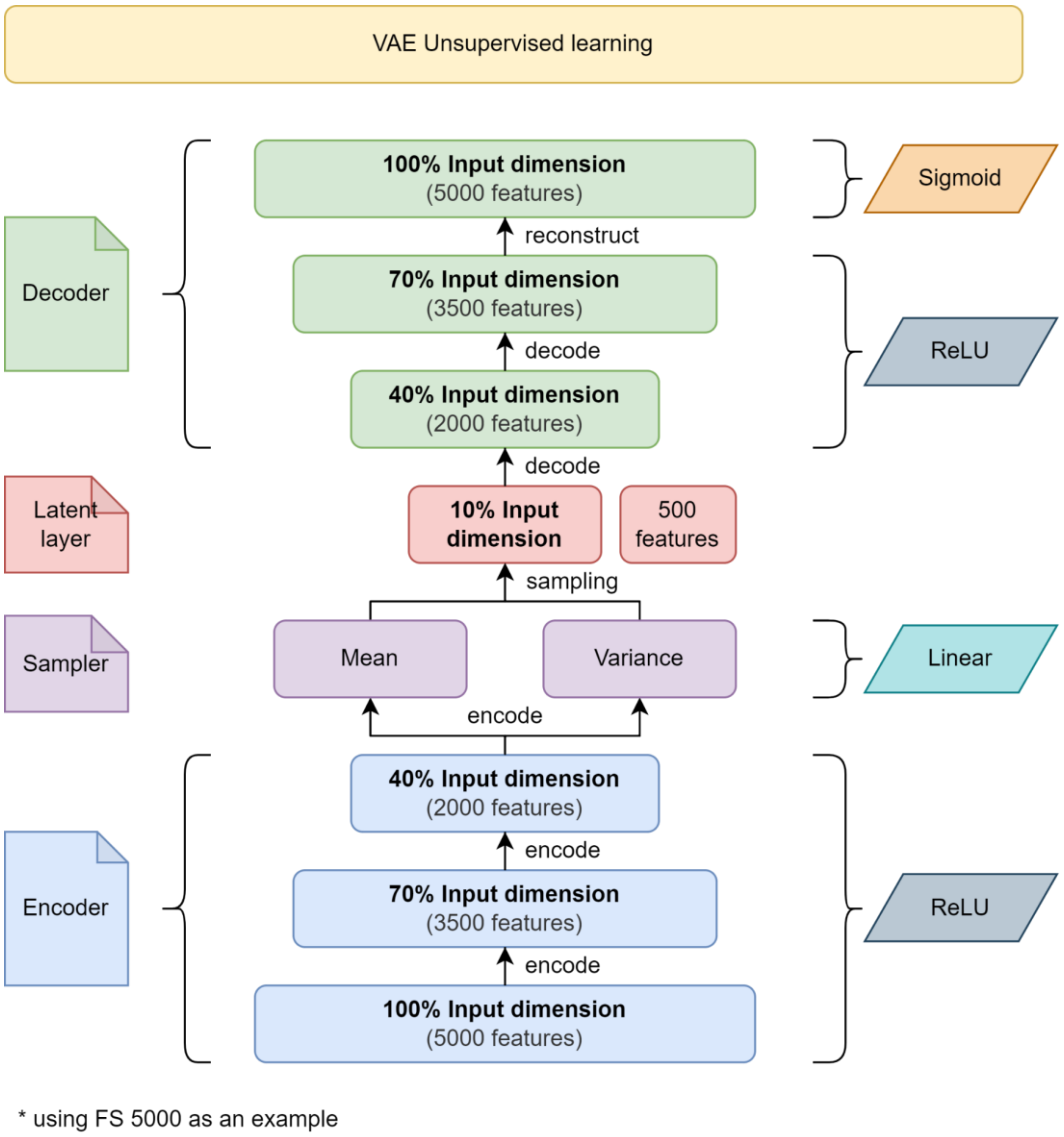


Figure 4.53 The neural network structure of the VAE model for unsupervised training

Next, the hyperparameters used for the unsupervised training of the VAE model are tabulated in Table 4.10. First of all, the VAE consists of eight (8) layers with asymmetrical dimensions as opposed to the SDAE model which has seven (7) symmetric dimensions. This is due to the properties of VAE having an extra sampling layer. Similar to the SDAE model, the dimension of each layer is determined by different fractions of the original input dimension. The three layers in the encoding layer consists of 100%, 70% and 40% of the input dimension. Next, the sampling layer is attached after the encoder with 40% input dimension. The sampling layer then samples the latent layer with 10% input dimension using the probability distribution of the mean and variance collected from the previous encoders. After the

latent layer is the decoding layer. Similar to the SDAE model, the decoding layer consists of 3 decoders with 40%, 70% and 100% input dimension respectively. The last layer, which is also the output layer of the model, attempts to reconstruct the original input with the same dimension. An example (coloured grey) for the actual dimension for each layer in the SDAE model is also shown in Table 4.10 by using an input dimension of 5000 features as an example.

Table 4.10 Hyperparameters used for VAE unsupervised training

Hyperparameters	**Description**
Layers	*5000, 3500, 2000, 500, 2000, 3500, 5000* 100%, 70%, 40%, 10%, 40%, 70%, 100%
Epoch	50
Batch size	16
Optimizer	adam
Activation functions	ReLU (Rectified Linear Unit) – Hidden layers Linear – Bottleneck layer (latent layer) Sigmoid – Last layer (output layer)
Loss function	Generative loss Kullback-Leibler (KL) loss

The hyperparameters used for the unsupervised training of the VAE model is similar to the SDAE model, except for the activation functions and the loss function. The dimensions of the layers have the same dimensions as in the SDAE model, with the exception of an additional sampling layer before the latent layer. Besides that, the epoch, batch size and optimizer are 50, 16 and "adam" respectively. There is a slight difference with the activation functions used in the VAE model as opposed to the SDAE model. With reference to Hira et al. (2021) also, "linear" activation function is used for the bottleneck layer, and the rest of the activation function remained the same as the SDAE model. There are two different loss functions used for the VAE model. The first loss function is the generative loss, whereby it measures the difference between the input and the generated output. The second loss is the latent loss, or the Kullback-Leibler (KL) loss for the VAE model. The overall loss of the

VAE is the combination of both the generative loss and the KL loss (Hira et al., 2021).

As mentioned in Chapter 4.3.5, the VAE models undergoes a different classification phase compared to the SDAE model. In the classification phase, the samples in the latent layer are extracted during the unsupervised learning phase. Then, another classification model, in this case, an SVM classifier is used to classify the encoded/compressed inputs. The reason of incorporating a different classification process is that the KL divergence loss function used in VAE is not suitable for classification using a sigmoid layer with one node. The accuracy obtained by using a sigmoid layer is fluctuating around 50%, which does not represent the actual accuracy of the model. Figure 4.54 shows the neural network structure of the VAE model for supervised training.

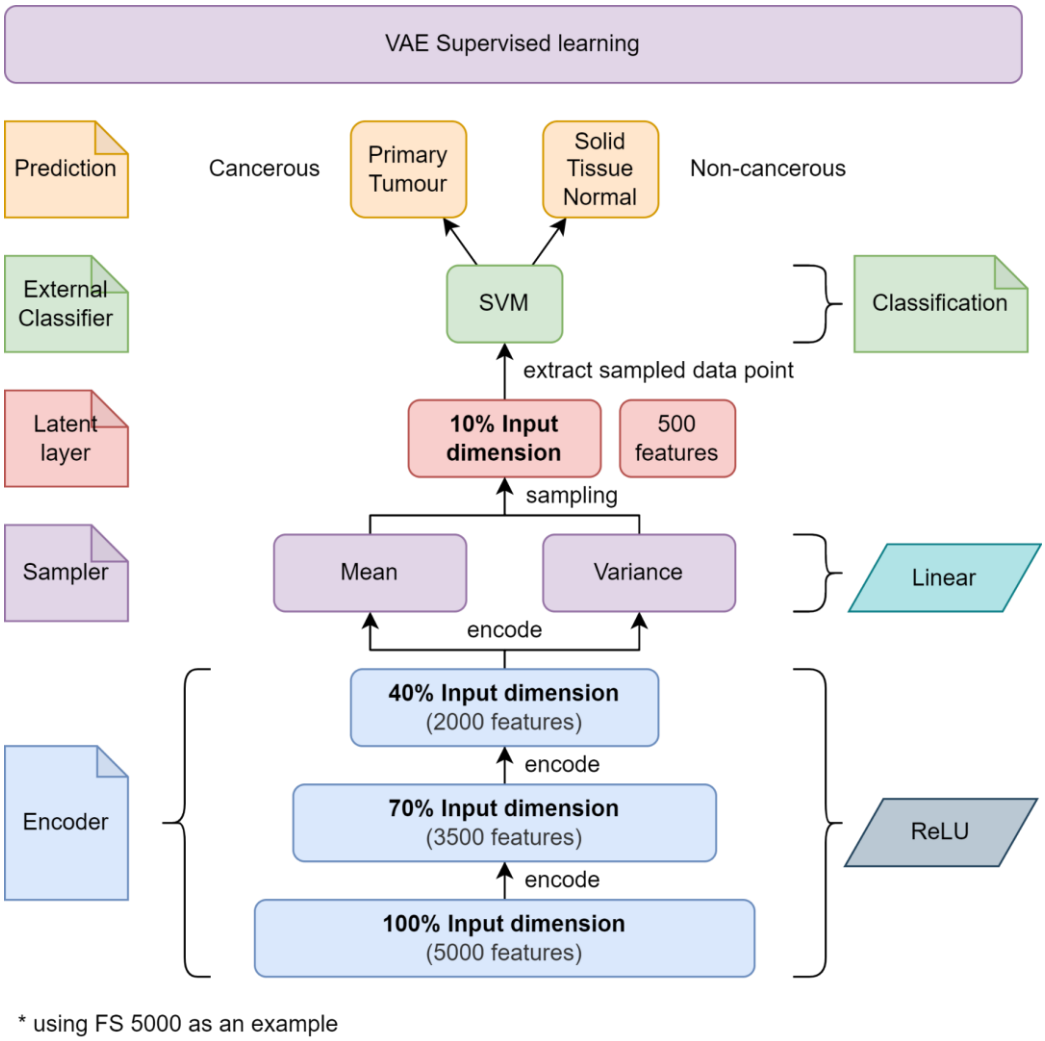


Figure 4.54 The neural network structure of the VAE model for supervised training (fine-tuning)

As mentioned, the encoded inputs in the latent layer, which is sampled from the sampler from the sampling layer, are extracted and used in classification using SVM as the external classifier. Initially, to keep the experiment with external classifier simple, the default parameters of the SVM model are used as per defined in the Python sklearn library, with the exception of the kernel manually set to "linear". Since linear kernel is selected, the only hyperparameter that is involved is "C", and its default value is 1.0. The hyperparameters used for the SVM as an external classifier is tabulated in Table 4.11. From the results obtained from the initial experiment, which is by using default SVM hyperparameters, it is found out that the default set of hyperparameters are already performing very well (details in Chapter 5.3.3 and 5.3.4). Thus, the study has decided to consider the default hyperparameters

of the SVM model as the optimal set of hyperparameters without further explorations.

Table 4.11 Default hyperparameters used for SVM as external classifier

Hyperparameters	Description
C	1.0
Kernel	linear

4.4 Chapter Summary

In this chapter, the experimental workflow of the study is discussed in detail. The implemented process of data preprocessing, exploratory data analysis and multi-omics data integration are thoroughly discussed. With the integrated multi-omics data, feature selection using SVM-RFE is performed to extract 20 feature subsets which are thought to be the most optimal using the set of hyperparameters selected using grid search. For SDAE and VAE implementation, the 5 smallest feature subsets obtained from the SVM-RFE feature selection phase are used to develop the two unsupervised learning models. The unsupervised SDAE and VAE models are fine-tuned into supervised learning models for classification.

CHAPTER 5

RESULT, ANALYSIS AND DISCUSSION

5.1 Introduction

In this chapter, the results obtained from the implementation of the algorithms according to the experimental workflow is analyzed and discussed in this chapter.

First and foremost, the output of SVM-RFE, which is the 20 feature subsets will be discussed. Initial classification of the feature selected subsets is done using a classifier, in this case, an SVM model. Stratified K fold cross validation is used in the initial classification to obtain a more generalizable result. The accuracy score for each feature subset is recorded and a boxplot is plotted according to their mean and standard deviation obtained from the cross validation. Besides that, the omics composition or distribution for each feature subsets is also compared and plotted in a composite bar chart.

Next, the performance of the SDAE and VAE models will be assessed based on several criteria. Firstly, the model loss of both the SDAE and VAE models in unsupervised training phase is recorded and analyzed. Then, for SDAE, the result of the classification is obtained from the supervised fine-tuning process. As for VAE, the encoded inputs in the latent layer is extracted and classified using an external classifier (i.e. SVM). Furthermore, confusion matrix is plotted to provide more assessment metrics such as the accuracy, precision, recall and F1 score.

Finally, the classification results obtained from the SDAE and VAE models are compared against each other. This allows the better neural network to be decided for the classification of multi-omics data in this study.

5.2 Assessment of SVM-RFE

In this subchapter, the SVM-RFE model is assessed. There are two subchapters dedicated for the assessment. Firstly, the results obtained from the SVM-RFE model is tabulated and depicted with detailed information extraction. Then, the results are discussed thoroughly to understand the reasoning behind the obtained results.

5.2.1 Result of SVM-RFE

The assessment of SVM-RFE starts with observing the output of the algorithm. With the outputs from running the experiment with SVM-RFE with cross validation (i.e. mean and standard deviation of accuracy), a boxplot is plotted to visualize the accuracy of each selected feature subset. The computation time for the selection of each feature subset is observed. The omics composition or distribution for each feature subset is also examined.

The output of the SVM-RFE algorithm for each feature subset is shown in Figure 5.1. For each feature subset, the cross-validation result for accuracy including the calculated mean and standard deviation, the most optimal set of hyperparameters selected by the grid search algorithm, and the computation time are recorded.

```
momics: accuracy
Fitting 2 folds for each of 12 candidates, totalling 24 fits
20000 features => mean=0.996, std=0.004, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 1:18:34.758923

Fitting 2 folds for each of 12 candidates, totalling 24 fits
19000 features => mean=0.996, std=0.004, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 0:11:10.744818

Fitting 2 folds for each of 12 candidates, totalling 24 fits
18000 features => mean=0.996, std=0.004, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 0:10:36.999596

Fitting 2 folds for each of 12 candidates, totalling 24 fits
17000 features => mean=0.996, std=0.004, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 0:10:01.563326

Fitting 2 folds for each of 12 candidates, totalling 24 fits
16000 features => mean=0.996, std=0.004, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 0:09:34.708557

Fitting 2 folds for each of 12 candidates, totalling 24 fits
15000 features => mean=0.996, std=0.004, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 0:08:54.693417

Fitting 2 folds for each of 12 candidates, totalling 24 fits
14000 features => mean=0.996, std=0.004, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 0:08:19.152421

Fitting 2 folds for each of 12 candidates, totalling 24 fits
13000 features => mean=1.000, std=0.000, cfg={'estimator__C': 0.1, 'estimator__kernel': 'linear', 'step': 1}
Duration: 0:08:00.837523
```

Figure 5.1 Results obtained from SVM-RFE for each feature subset (using feature subset 20000 to 13000 as example)

A supervised learning model, which is an SVM model, is used to classify the feature subsets selected by SVM-RFE. This allows the discovery of early insights regarding the selected subsets of features. The hyperparameter of this SVM is set according to the set of hyperparameters selected by the grid search algorithm. Stratified K fold with 2 splits is used for cross validation. The accuracy obtained from the CV result is plotted in form of boxplot as shown in Figure 5.2. It can be seen that as the feature subsets become smaller, the accuracy of the SVM classifier improves. Specifically, the mean accuracy of the SVM model for feature subset 20000 to 14000 recorded at 0.996 with standard deviation of 0.004. Starting from feature subset 13000 to 1000, the mean accuracies are recorded at 100% accuracy with zero standard deviation.

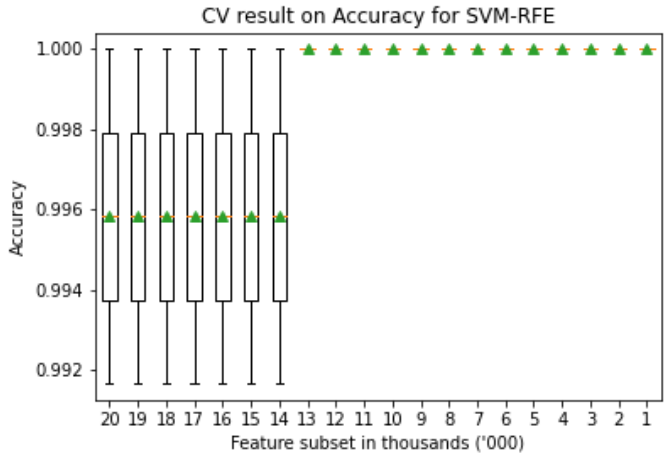


Figure 5.2 Boxplot for the CV result (accuracy) for each feature subset

By using the grid search algorithm with the provided parameter grid, the most optimal set of hyperparameters has been found for each feature subset. According to the result of grid search, the algorithm decided that one set of hyperparameter could be applied to the feature selection of all feature subsets using SVM-RFE, which is by setting the penalty parameter "C" equals to 0.1 and the number of features to remove per RFE iteration "step" equals to 1."linear" kernel is the only kernel provided in the parameter grid, therefore is always selected. The selected set of hyperparameters for SVM-RFE is tabulated in Table 5.1.

Table 5.1 The selected set of hyperparameters for the feature selection using SVM-RFE for each feature subset

Feature Subset	**Computation Time**
C	0.1
kernel	linear
step	1

The computation time for the FS for each feature subset is tabulated in Table 5.2. The overall runtime for the SVM-RFE to finish selecting 20 feature subsets is 3 hours and 9 minutes. The feature selection for the first feature subset, which is feature subset 20000, took about 1 hour and 20 minutes for the computation to complete. This huge difference in computation time compared to the FS of the other feature subset is due to the fact that the dimension of the multi-omics data is reduced

from 26130 features to 20000 features, while the FS of the other feature subsets are reduction of 1000 features each. It can also be seen that as the size of the feature subset become smaller, the computation time is also faster. The removal of each thousands of features improves the computation time by around 30 seconds.

Table 5.2 The computation time for the FS for each feature subset

Feature Subset	**Computation Time**		**Feature Subset**	**Computation Time**
20000	1h 18m 34s		10000	5m 59s
19000	11m 10s		9000	4m 45s
18000	10m 36s		8000	4m 06s
17000	10m 01s		7000	3m 47s
16000	9m 34s		6000	3m 08s
15000	8m 54s		5000	2m 38s
14000	8m 19s		4000	2m 04s
13000	8m 00s		3000	1m 39s
12000	7m 18s		2000	1m 10s
11000	6m 37s		1000	45s
Total				**3h 09m 14s**

The composition/distribution of the omics features after feature selection for each feature subset is also recorded, which is tabulated in Table 5.3. The same data is also plotted in a composite bar chart as shown in Figure 5.3 to better visualize the omics distribution for each feature subset. In Figure 5.3, the omics compositions are shown in percentages. The bars with colours blue, red and green represents the gene expression, DNA methylation and miRNA expression respectively. According to the figure, it appears that the composition of the gene expression omics reduces from feature subset 20000 to feature subset 5000, while the composition of the DNA methylation omics increases for the same range of feature subsets. Then, it is observed that starting from feature subset 5000 to 1000, the composition of gene expression omics increases while the composition of DNA methylation omics decreases. As the size of the feature subsets decreases, the composition of miRNA expression decreases steadily until feature subset 34000, which is recorded at only 1.35%. Its composition then increases from feature subset 4000 to 1000.

Table 5.3 Omics composition for each feature subset

Feature Subset	Omics Types		
	Gene Exp.	**DNA Methylation**	**miRNA Exp.**
26130 (original)	20244 (0.78)	5000 (0.19)	886 (0.03)
20000	15017 (0.75)	4554 (0.23)	429 (0.02)
19000	14196 (0.75)	4407 (0.23)	397 (0.02)
18000	13391 (0.74)	4244 (0.24)	365 (0.02)
17000	12582 (0.74)	4078 (0.24)	340 (0.02)
16000	11794 (0.74)	3891 (0.24)	315 (0.02)
15000	11027 (0.74)	3687 (0.25)	286 (0.02)
14000	10265 (0.73)	3479 (0.25)	256 (0.02)
13000	9503 (0.73)	3264 (0.25)	233 (0.02)
12000	8716 (0.73)	3070 (0.26)	214 (0.02)
11000	7934 (0.72)	2876 (0.26)	190 (0.02)
10000	7181 (0.72)	2644 (0.26)	175 (0.02)
9000	6431 (0.71)	2418 (0.27)	151 (0.02)
8000	5691 (0.71)	2173 (0.27)	136 (0.02)
7000	4962 (0.71)	1927 (0.28)	111 (0.02)
6000	4235 (0.71)	1668 (0.28)	97 (0.02)
5000	3525 (0.70)	1397 (0.28)	78 (0.02)
4000	2833 (0.71)	1113 (0.28)	54 (0.01)
3000	2125 (0.71)	832 (0.28)	43 (0.01)
2000	1450 (0.72)	520 (0.26)	30 (0.01)
1000	731 (0.73)	251 (0.25)	18 (0.02)

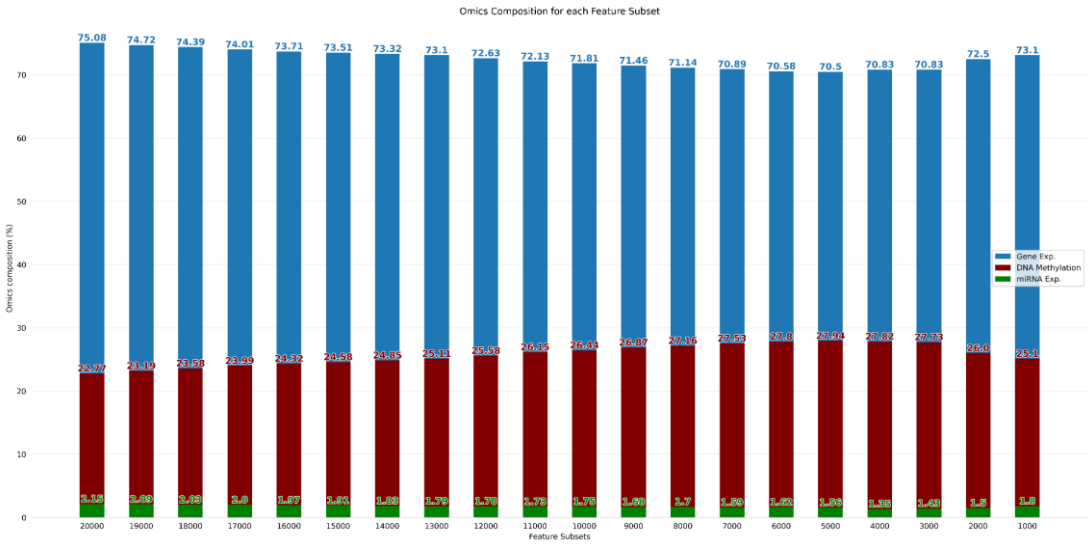


Figure 5.3 Omics composition after SVM-RFE represented in bar chart

5.2.2 Discussion on SVM-RFE

According to the boxplot for the accuracy obtained from the classification of each selected feature subset using an SVM model shown in Figure 5.2, some early insights related to the selected feature subsets is acquired. The accuracies of the classification of feature subset 20000 to 14000 recorded at an average of 0.996 with standard deviations of 0.004. This indicates that out of 2 splits of stratified K folds, the classifier failed to correctly classify the minority class (Solid Tissue Normal) in one of the splits. However, for feature subset 13000 to 1000, all the instances are correctly classified, hence an accuracy of 100%.

It could be deduced that, as the size of the feature subsets reduces, more features with low predictive power are removed. Since the remaining features have better predictive power for the class labels, smaller feature subsets could perform better in classification. In other words, the hyperplane drawn by the SVM classifier could do a better job at separating the data points into their respective class labels.

The omics distribution for each feature subset is plotted in a composite bar chart as shown in Figure 5.3, and several trends could be observed. As mentioned in the previous section, the composition of the gene expression omics declines as the size of the feature subset becomes smaller until feature subset 5000, and then rises again until feature subset 1000. On the other hand, the composition of the DNA methylation expression is observed to be the opposite of the trend displayed by gene expression omics. miRNA expression saw a decline in composition until feature subset 11000, and continues to fluctuates until feature subset 1000.

This could potentially suggest that from feature subset 20000 to 5000, most of the features from gene expression omics are features with low predictive power, therefore have lower weights assigned to them and further removed by the SVM-RFE algorithm. The observed rise in the composition of DNA methylation expression omics could be explained as these features are assigned with larger weights by the SVM-RFE algorithm and therefore kept. From feature subset 4000 onwards, as more unimportant features from the gene expression omics are removed, only the ones with good predictive power are left, which is why the opposite trend of the omics composition is observed. From feature subset 4000 to 1000, more features from DNA methylation expression are assigned lower weights by the SVM-RFE algorithm and removed as they are now less relevant compared to the rest of the omics features. With that said, the explanations given only limits to this particular multi-omics dataset in this particular experiment.

5.3 Assessment of Deep Learning Models

The assessments of the deep learning models (i.e. SDAE & VAE) are done in this subchapter. Due to hardware limitation of insufficient GPU memory, only the feature subset 5000 to 1000 is used for the construction of the deep learning models. The results and discussions for the two deep learning models are shown separately in their respective subchapters. Firstly, the results obtained from the unsupervised training and the fine-tuning supervised learning for the models are shown. In unsupervised learning, the model loss is the main metric to be taken note.

For the SDAE model, classification is done by fine-tuning the model into a supervised classification model. As for the VAE model, the classification is done on the encoded inputs extracted from the latent layer using an external classifier (i.e. SVM). The results of the classifications are recorded in the form of confusion matrix. From the confusion matrix, various metrics are extracted to finalize the performance of the models, such as the accuracy, precision, recall and F1 score.

5.3.1 Result of the SDAE Model

The model losses for the SDAE model for feature subset 5000 to 1000 in the unsupervised training phase is tabulated in Table 5.4. As mentioned in Chapter 3.2.3, the training dataset (which is a 70% split from the full dataset) is split again into 70% training data and 30% validation data. Both the training and validation datasets are used in the unsupervised learning for the SDAE model. The model loss for SDAE using both the training and validation datasets (denoted as T and V respectively) for each feature subset are recorded and tabulated in Table 5.4. In the tabulated data consists of the SDAE model loss for 50 epochs. However, only the first and the last 10 epochs are shown to simplify the table.

Table 5.4 Model loss for the unsupervised learning for the SDAE model

Epoch	SDAE model loss according to feature subset									
	5000		4000		3000		2000		1000	
	T	V	T	V	T	V	T	V	T	V
1	0.633	0.572	0.577	0.482	0.542	0.484	0.530	0.506	0.541	0.517
2	0.506	0.461	0.480	0.466	0.476	0.483	0.475	0.501	0.465	0.509
3	0.478	0.453	0.474	0.467	0.474	0.483	0.472	0.504	0.462	0.508
4	0.474	0.453	0.473	0.467	0.472	0.480	0.470	0.502	0.460	0.508
5	0.473	0.453	0.472	0.466	0.471	0.480	0.469	0.501	0.459	0.508
6	0.472	0.453	0.471	0.466	0.470	0.481	0.469	0.502	0.458	0.508
7	0.472	0.453	0.472	0.466	0.468	0.481	0.466	0.500	0.456	0.507
8	0.472	0.452	0.471	0.465	0.457	0.479	0.461	0.496	0.453	0.501
9	0.471	0.453	0.471	0.465	0.463	0.480	0.460	0.499	0.450	0.504
10	0.471	0.452	0.470	0.465	0.464	0.480	0.460	0.498	0.448	0.499
…	…	…	…	…	…	…	…	…	…	…
41	0.453	0.448	0.446	0.460	0.440	0.474	0.440	0.494	0.427	0.498
42	0.452	0.448	0.446	0.460	0.439	0.474	0.438	0.494	0.426	0.498
43	0.452	0.448	0.446	0.460	0.439	0.474	0.437	0.494	0.426	0.498
44	0.451	0.448	0.445	0.460	0.438	0.475	0.435	0.495	0.425	0.499
45	0.450	0.448	0.444	0.460	0.437	0.477	0.435	0.494	0.425	0.498
46	0.451	0.448	0.445	0.460	0.439	0.477	0.434	0.494	0.424	0.498
47	0.452	0.448	0.446	0.460	0.438	0.475	0.434	0.494	0.423	0.498
48	0.452	0.446	0.444	0.461	0.436	0.474	0.432	0.494	0.422	0.498
49	0.450	0.449	0.446	0.461	0.435	0.474	0.431	0.494	0.422	0.498
50	0.449	0.448	0.443	0.461	0.433	0.475	0.430	0.495	0.422	0.498

To better visualize the trend of the SDAE model loss per epoch during the unsupervised learning, a line chart is drawn for each feature subset as shown in Figure 5.4. According to the charts, the SDAE model losses for each feature subset are low, which are below 0.5. It could be observed that the model losses for the validation set are able to converge, while the model losses for the training set appears to be still declining. One thing to note is that the model loss for both training and validation dataset for feature subset 5000 are identical at the 50^{th} epoch. Feature

subset 5000 is the only model with validation data model loss resembles the training data model loss. For the rest of the models, there is a noticeable difference in the model loss between the training and validation set.

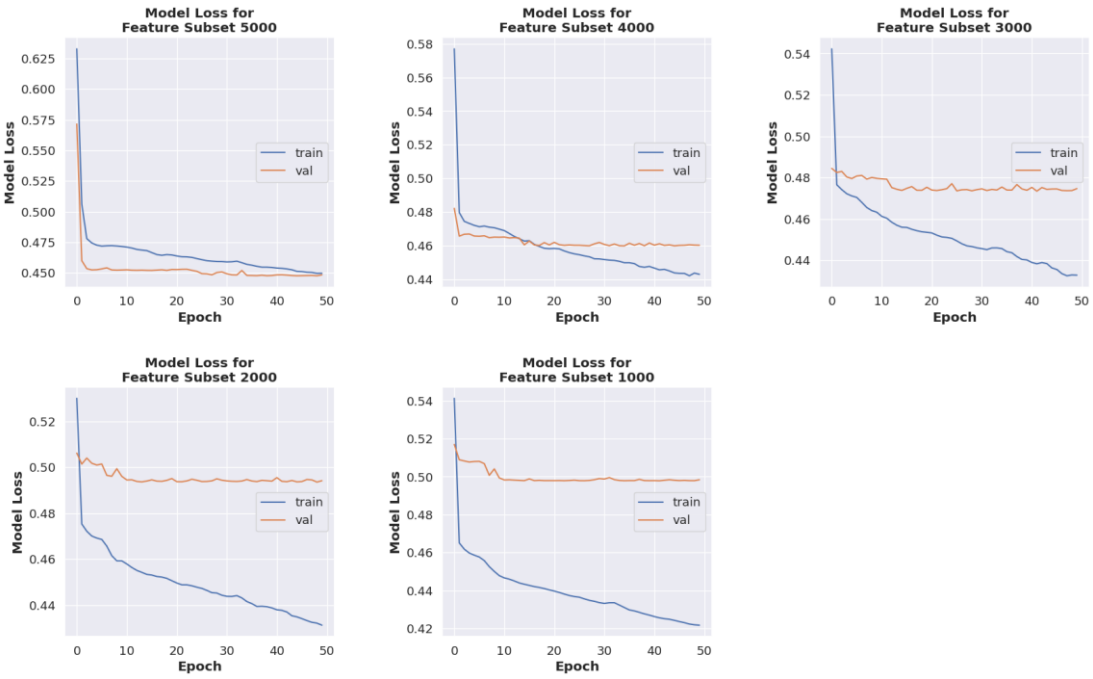


Figure 5.4 Model loss for the unsupervised learning for the SDAE model

After the unsupervised learning phase, the SDAE model undergoes fine-tuning process, in which the neural network of the model is modified into a supervised learning model for classification of the encoded inputs. This time, the training set is used for the model building while the test set is used to obtain the final metric of the SDAE model. In supervised learning of the SDAE model, the set of hyperparameters used is similar to those in unsupervised learning of the SDAE model, with the exception of the neural network layers. The decoder part of the SDAE model is replaced with a layer with only one node using the "sigmoid" activation function. This acts as the new output layer for the fine-tuned SDAE model. The new output layer produces numerical outputs which are between 0 and 1 for classification, in which 0 represents the non-cancerous outcome (Solid Tissue Normal) while 1 represents the cancerous outcome (Primary Tumour). The accuracy per epoch of the fine-tuned SDAE classifier for each feature subset is tabulated in Table 5.5.

Table 5.5 Accuracy for the supervised learning for the SDAE model

Epoch	Feature subset									
	5000		4000		3000		2000		1000	
	T	V	T	V	T	V	T	V	T	V
1	0.916	0.990	0.950	0.990	0.987	0.990	0.987	0.990	0.987	1.000
2	0.987	1.000	0.985	0.990	1.000	0.990	1.000	0.990	1.000	1.000
3	1.000	0.990	1.000	0.990	1.000	0.990	1.000	0.990	1.000	1.000
4	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
5	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
6	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
7	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
8	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
9	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
10	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
…	…	…	…	…	…	…	…	…	…	…
41	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
42	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
43	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
44	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
45	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
46	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
47	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
48	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
49	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000
50	1.000	0.990	1.000	1.000	1.000	0.990	1.000	0.990	1.000	1.000

Similarly, a line chart is also plotted for the accuracy per epoch obtained from the classification using the SDAE model as shown in Figure 5.5. At first glance, it could be observed that feature subset 1000 and 4000 achieved 100% accuracy on both the training and testing data. The SDAE model built with feature subset 1000 is able to achieve 100% accuracy on the testing set on the first epoch, and 100% accuracy on the training set after the first few epochs. The SDAE model with feature subset 4000 on the other hand achieved 100% accuracy on both the training and

testing set after the first few epochs. Besides that, it could be seen that all SDAE models are able to achieve 100% accuracy on the training set. However, the SDAE models built with feature subset 2000, 3000 and 5000 were unable to achieve 100% accuracy on the validation set.

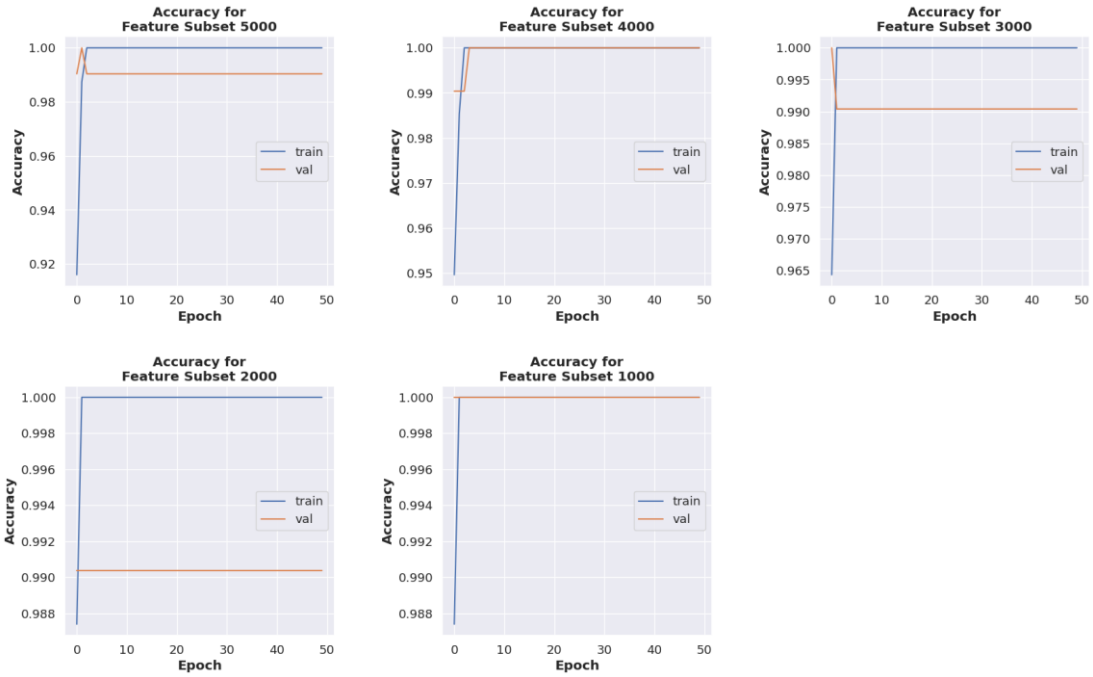


Figure 5.5 The accuracy for supervised learning for the SDAE model

The confusion matrices are plotted with the classification results obtained from each of the SDAE models. Figure 5.6 depicts the said confusion matrices. As discussed in the previous paragraph, it can be seen that the SDAE model built with feature subset 1000 and 4000 classified all the instances in the testing set correctly. Meanwhile, the SDAE models built with feature subset 2000, 3000 and 5000 are able to classify all the instances with majority class label (i.e. Primary Tumour) but misclassifies the only instance with the minority class label (i.e. Solid Tissue Normal).

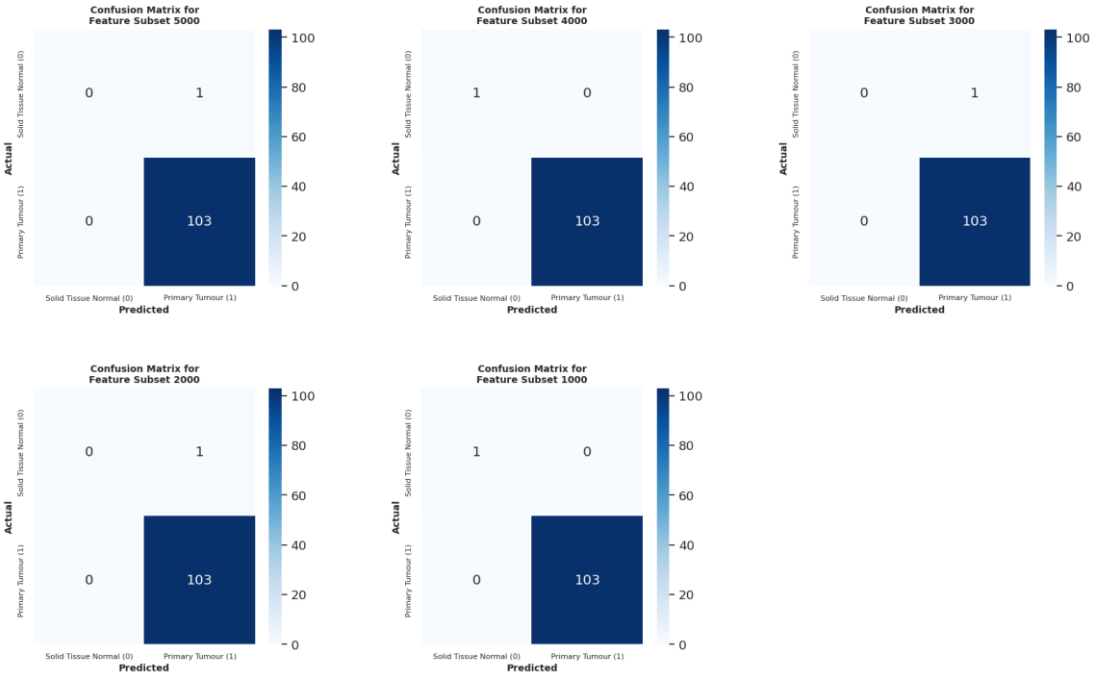


Figure 5.6 The confusion matrix from the classification using the fine-tuned SDAE model

With the plotted confusion matrix, several assessment metrics can be calculated. The metrics are calculated based on the equations mentioned in Chapter 3.4 Performance Measure. These metrics are tabulated in Table 5.6 for each feature subset.

Table 5.6 Metrics obtained from the classification result of SDAE model

Feature Subset	Accuracy	AUC (ROC)	Precision	Recall	F1 score
5000	0.9904	0.5000	0.9904	1.0000	0.9951
4000	1.0000	1.0000	1.0000	1.0000	1.0000
3000	0.9904	0.5000	0.9904	1.0000	0.9951
2000	0.9904	0.5000	0.9904	1.0000	0.9951
1000	1.0000	1.0000	1.0000	1.0000	1.0000

5.3.2 Discussion on SDAE Model

It could be observed that feature subset 1000 and 4000 achieved perfect score for all the measured metrics, which is as expected since all the samples are classified correctly. Despite the testing set being severely imbalanced, the SDAE models trained with the two feature subsets are still able to capture important features which are able to distinguish the difference between the class labels.

As for the metrics obtained from the classification on the rest of the feature subsets (i.e. 2000, 3000 and 5000), the values are the same which are also expected. Despite measuring an accuracy of 0.9904, the AUC score is only recorded at 0.5. An AUC score of 0.5 suggests that the SDAE model has no capability of separating the class labels whatsoever. Accuracy is actually a misleading metric to assess the model performance in this case study, since the testing set is severely imbalanced. Even by predicting only the majority class, the SDAE model could still achieve very high accuracy (0.9904).

A brief conclusion to be drawn from the results obtained from the SDAE models built with feature subset 1000 to 5000 suggests that, feature subset 1000 and 4000 are the only two feature subsets which contain the appropriate number of features which contains the relevant features which allows the model to distinguish and classify the samples into their correct classes. On the other hand, feature subset 2000, 3000 and 5000 are probably saturated with features which are more relevant at predicting the majority class label (Primary Tumour), in which the SDAE models failed to capture the subtle difference between the values of the selected features to correctly classify the minority class.

5.3.3 Result of the VAE Model

The VAE model is constructed using three (3) different loss functions as stated in Chapter 4.3.6.2, namely the reconstruction loss, the KL loss and the overall model loss, which is calculated using the two losses. All the VAE losses are recorded and plotted accordingly. Firstly, the overall model losses for the VAE model for feature subset 5000 to 1000 in the unsupervised training phase is tabulated in Table 5.7. Similarly, the training dataset (which is a 70% split from the full dataset) is split again into 70% training data and 30% validation data. Both the training and validation datasets are used in the unsupervised learning for the VAE model. The model loss for VAE using both the training and validation datasets (denoted as T and V respectively) for each feature subset are recorded and tabulated in Table 5.7. In the tabulated data consists of the VAE model loss for 50 epochs. However, only the first and the last 10 epochs are shown to simplify the table.

Table 5.7 Overall model loss for the unsupervised learning for the VAE model

Epoch	**VAE overall loss according to feature subset**									
	5000		**4000**		**3000**		**2000**		**1000**	
	T	**V**	**T**	**V**	**T**	**V**	**T**	**V**	**T**	**V**
1	127.19	23.21	101.37	41.26	83.47	63.88	58.61	55.54	34.71	37.54
2	72.39	21.40	59.52	26.02	48.71	18.73	33.08	12.24	19.65	10.26
3	66.19	19.48	54.71	30.84	43.07	14.90	28.12	11.98	15.14	8.04
4	64.94	17.44	54.37	13.85	41.84	15.49	27.62	10.78	14.36	7.77
5	63.77	28.22	52.94	22.78	42.09	17.02	27.18	10.02	14.26	7.69
6	66.19	25.89	52.83	16.18	41.17	12.54	26.68	11.19	14.04	7.60
7	65.64	18.93	51.91	14.16	39.49	13.61	26.66	9.32	14.37	7.64
8	64.01	17.46	50.64	13.87	38.97	12.34	25.88	12.41	14.22	7.93
9	62.70	16.25	50.64	14.05	39.67	17.47	26.97	10.52	14.40	8.40
10	61.73	19.25	50.89	13.92	39.82	13.80	26.58	10.08	13.75	7.35
…	…	…	…	…	…	…	…	…	…	…
41	58.33	12.40	47.19	13.96	37.44	12.19	24.61	8.43	13.22	7.06
42	59.37	12.24	47.97	14.45	37.03	12.69	24.84	7.69	13.75	5.75

43	59.07	12.27	47.32	10.97	37.08	8.68	24.50	7.08	13.32	5.22
44	58.22	12.52	47.56	14.57	36.76	8.65	24.40	7.34	13.09	5.15
45	58.22	15.97	48.04	10.27	37.02	10.27	24.36	8.54	13.24	5.98
46	58.21	13.58	48.17	11.42	37.58	10.61	24.73	8.33	13.14	4.99
47	58.25	11.80	47.52	10.02	36.33	10.07	24.51	12.13	13.24	6.37
48	58.48	23.76	47.32	14.26	37.68	10.25	24.36	7.18	12.66	5.70
49	58.49	11.96	47.36	9.93	36.41	9.06	24.67	7.95	13.39	5.78
50	58.02	14.53	46.47	11.36	36.25	8.73	23.99	6.99	13.11	5.68

The trend of the overall VAE model loss during the unsupervised training phase for each feature subset is depicted in the form of line charts in Figure 5.7. According to the charts, the trend for the overall VAE model loss for each feature subset are very similar, whereby the VAE models are able to minimize the overall loss for both the training and validation set. The overall model loss for the training sets for all feature subsets converged steadily and stayed almost constant from the 40^{th} to the 50^{th} epoch. Meanwhile, the overall model losses for each feature subsets are also able to converge; however, the convergence is unsmooth and some fluctuations are observed. It is also observed that the overall VAE model loss become smaller as the size of the feature subset becomes smaller. The difference in the overall model loss between each 1000 feature reduction is around 10.

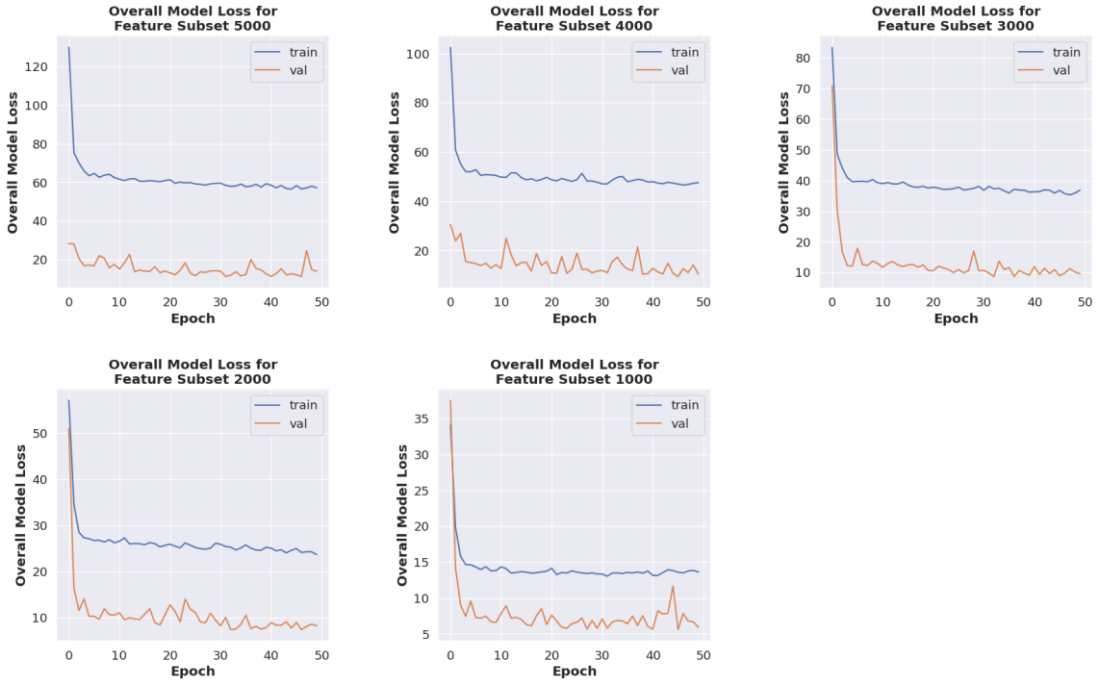


Figure 5.7 The overall loss of the VAE model during unsupervised learning phase

Up until now, only the overall VAE model loss has been discussed. The other two losses, which are the model reconstruction loss and the KL loss are plotted separately in Figure 5.8 and 5.9 respectively. The model reconstruction loss is observed to have similar traits shown in the VAE overall model loss. The VAE model reconstruction loss of the training set is converged smoothly, while the validation set converged at a lower loss value, but shown some degree of fluctuation. For smaller size of feature subset, the model reconstruction is also smaller.

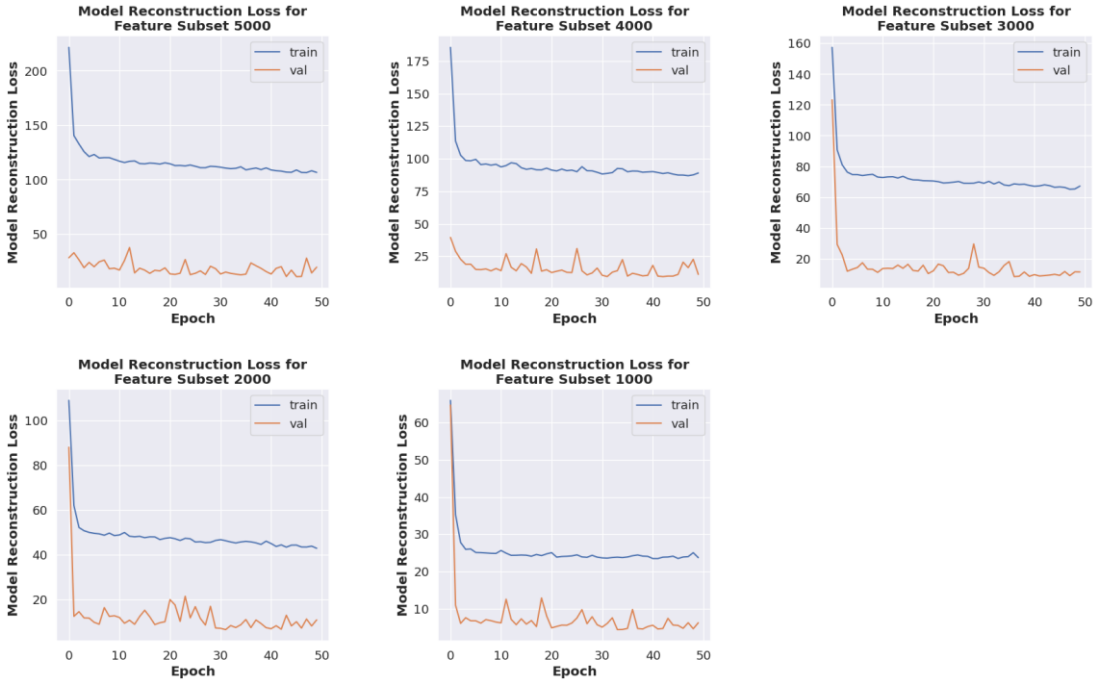


Figure 5.8 The reconstruction loss of the VAE model during unsupervised learning phase

The trend of KL loss is observed to be different. For all feature subsets, the VAE models constructed shown unstable and fluctuating KL loss for both the training and validation set, in which the degree of fluctuation displayed in the validation set is much larger. Opposite to the overall model loss and the model reconstruction loss, the KL loss for the validation set for all feature subsets are observed to be higher than that of the training set.

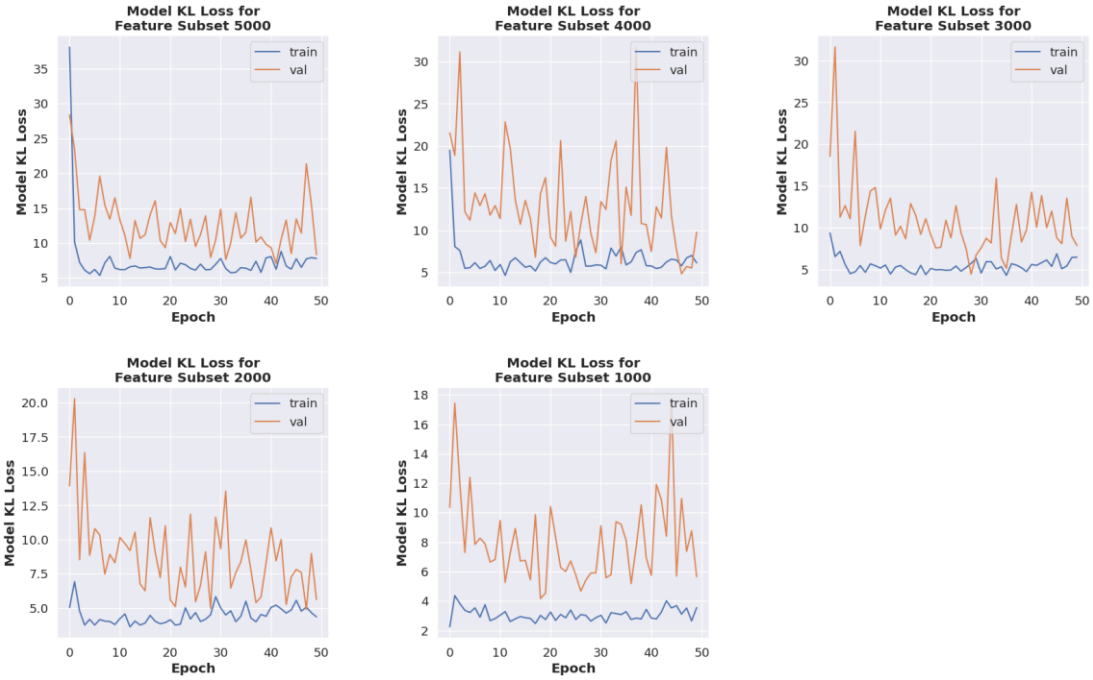


Figure 5.9 The KL loss of the VAE model during unsupervised learning phase

The sampled data from the latent layer by the sampler of the VAE model are visualized by plotting a scatterplot based on the encoded features. Figure 5.10 shows the sampled data for two randomly chosen features according to the sampler in feature subset 1000. According to the scatterplot, the VAE model is able to generate new data based on the feature learnt by the sampler. There is a clear separation between the class labels of the data.

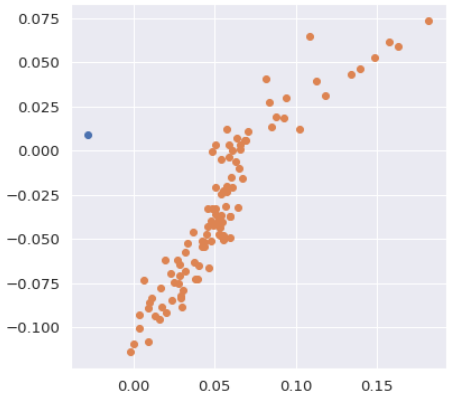


Figure 5.10 Data points sampled by the VAE model based on two randomly selected features

Similarly, the trained VAE model from the unsupervised learning phase is used for classification. There are two classification results obtained from the VAE model. The first result is from the fine-tuning of the VAE model into a supervised learning model for classification (with the same fine-tuning process as the SDAE model), while the other result is from the classification of the extracted sampled data by the sampler in the unsupervised learning phase using an external classifier (i.e. SVM). Both of the results obtained are discussed in the subsequent paragraphs.

Similar to the fine-tuning process of the SDAE model, the VAE model undergoes fine-tuning process by modifying the structure of the neural network, in which the decoder part of the VAE model is replaced with a layer with a single node with sigmoid activation function. This new output layer allows the VAE model to classify the data sampled by the sampler in the VAE model. The training set is used for training the fine-tuned VAE model while the testing set is used to obtain the final metric of the VAE model. The hyperparameters used are kept the same. The accuracy per epoch of the fine-tuned VAE classifier for each feature subset is tabulated in Table 5.8.

Table 5.8 Accuracy for the supervised learning for the VAE model

Epoch	**Feature subset**									
	5000		**4000**		**3000**		**2000**		**1000**	
	T	**V**	**T**	**V**	**T**	**V**	**T**	**V**	**T**	**V**
1	0.521	0.500	0.532	0.577	0.494	0.423	0.490	0.471	0.511	0.567
2	0.500	0.500	0.536	0.587	0.492	0.587	0.515	0.596	0.502	0.519
3	0.523	0.452	0.525	0.539	0.523	0.567	0.502	0.404	0.525	0.452
4	0.502	0.519	0.496	0.462	0.513	0.519	0.469	0.654	0.525	0.490
5	0.540	0.510	0.553	0.490	0.506	0.423	0.481	0.462	0.561	0.567
6	0.466	0.577	0.483	0.433	0.460	0.490	0.536	0.529	0.511	0.567
7	0.477	0.452	0.498	0.510	0.481	0.500	0.483	0.558	0.586	0.664
8	0.494	0.414	0.512	0.452	0.462	0.452	0.479	0.519	0.584	0.606
9	0.513	0.606	0.508	0.471	0.502	0.442	0.483	0.539	0.534	0.567
10	0.519	0.510	0.456	0.423	0.483	0.519	0.513	0.471	0.540	0.567

…	…	…	…	…	…	…	…	…	…	…
41	0.464	0.471	0.458	0.462	0.502	0.519	0.475	0.519	0.563	0.490
42	0.502	0.481	0.511	0.500	0.494	0.510	0.519	0.539	0.559	0.587
43	0.506	0.539	0.475	0.519	0.523	0.433	0.492	0.519	0.546	0.615
44	0.492	0.462	0.481	0.587	0.538	0.510	0.496	0.490	0.544	0.490
45	0.502	0.452	0.500	0.529	0.477	0.558	0.523	0.519	0.563	0.567
46	0.506	0.471	0.542	0.404	0.548	0.452	0.491	0.462	0.546	0.490
47	0.494	0.471	0.567	0.510	0.532	0.587	0.513	0.519	0.519	0.577
48	0.537	0.606	0.548	0.57	0.519	0.567	0.504	0.442	0.546	0.529
49	0.469	0.490	0.478	0.558	0.502	0.471	0.492	0.519	0.554	0.510
50	0.487	0.577	0.494	0.452	0.502	0.462	0.463	0.529	0.542	0.519

To better visualize the results obtained from the fine-tuning of the VAE model, a line chart is plotted for the accuracy per epoch obtained from the classification as shown in Figure 5.11. The accuracy obtained for each feature subset are almost indifferent. Both the accuracy for the training and testing sets fluctuated and averaged around 0.5. Despite using the similar fine-tuning method, the classification result obtained from the VAE model is completely different compared to the classification result obtained from the SDAE model. This could be implying that the loss functions used in the VAE model (i.e. model reconstruction loss, KL loss & overall loss) are not suitable to train the VAE model into a supervised learning model, as opposed to the SDAE model which used the BCE loss function.

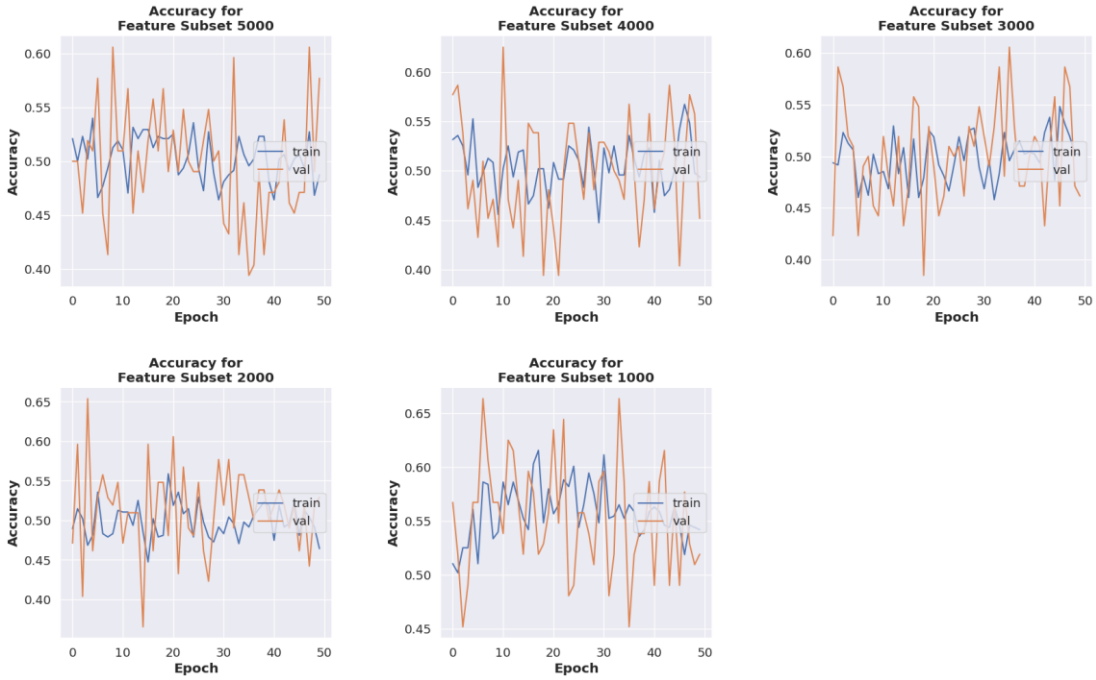


Figure 5.11 Accuracy obtained from the classification result of the fine-tuned supervised learning VAE model

To further understand the classification result of the VAE models, confusion matrices are plotted for each feature subset. The confusion matrices plotted for each feature subset are very similar with subtle difference. According to the confusion matrices, the VAE models built with all feature subsets are showing about 50% of false negative rate, meaning the VAE models are classifying the majority class label (Primary Tumour) wrongly almost 50% of the time. It could also be noticed that only the VAE model built with feature subset 3000 and 4000 are able to correctly classify the minority class label (Solid Tissue Normal), while the feature subset 1000, 2000 and 5000 are classifying it wrongly. From the confusion matrices, the other metrics are calculated and tabulated in Table 5.9.

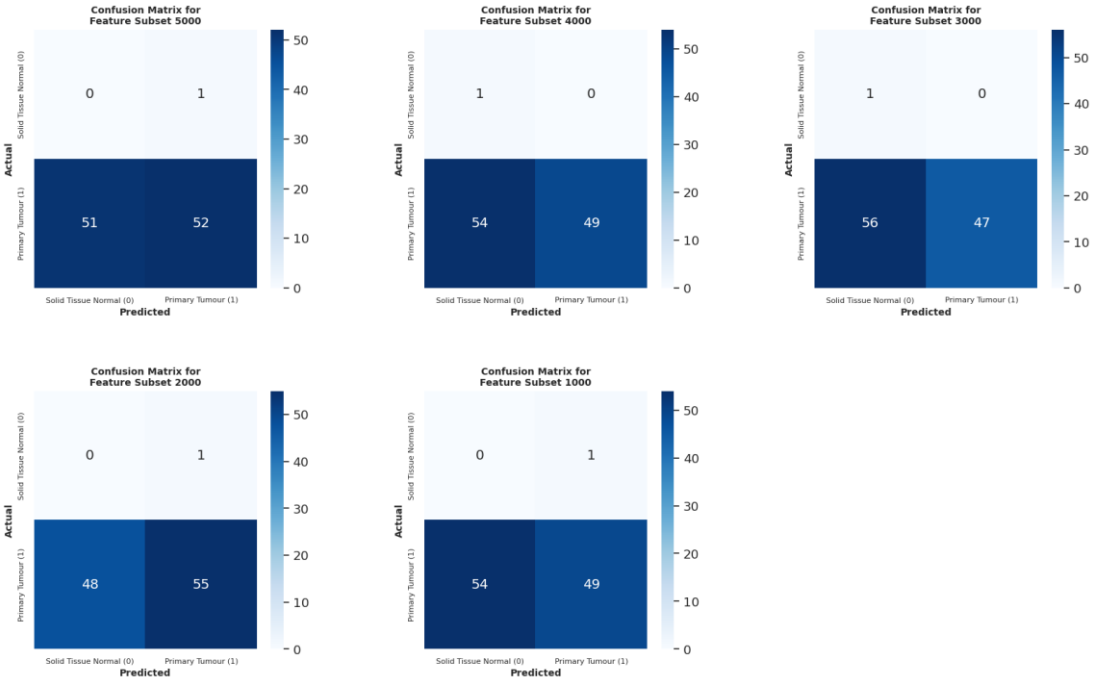


Figure 5.12 Confusion matrices produced using the classification results of the fine-tuned supervised learning VAE model

Table 5.9 Metrics obtained from the classification result of fine-tuned supervised learning VAE model

Feature Subset	**Accuracy**	**AUC (ROC)**	**Precision**	**Recall**	**F1 score**
5000	0.5000	0.2524	0.9811	0.5049	0.6667
4000	0.4808	0.7379	1.0000	0.4757	0.6447
3000	0.4615	0.7282	1.0000	0.4563	0.6267
2000	0.5288	0.2670	0.9821	0.5340	0.6918
1000	0.4712	0.2379	0.9800	0.4757	0.6052

Since the classification result obtained from the fine-tuned supervised learning VAE model is oddly unsatisfactory, another classification experiment is done by deploying an external classification model. In this case, an SVM classifier is used. In order to classify the data sampled by the sampler in the VAE model, the sampled data has to be extracted first. This is done by creating a separate neural network which consists of only the encoding part of the VAE model, along with the sampler. This neural network is named the VAE encoder. The output of VAE encoder will be the sampled data by the sampler. With the VAE encoder, the

sampled data can be extracted by feeding the original input into the VAE encoder to produce the sampled data, which is present in the latent layer. The extracted sampled data are then fed to the SVM classifier for classification.

The results of the classification using SVM for each sampled data from each feature subset are plotted in form of confusion matrices as shown in Figure 5.13. According to the classification result using SVM, all the sampled data from each feature subset are able to be classified correctly, achieving 100% for both True Positive (TP) rate and True Negative (TN) rate, resulting in an accuracy of 100%. With the confusion matrix, the rest of the metrics are calculated and tabulated in Table 5.10. As expected, all feature subsets achieved 100% score for each metrics.

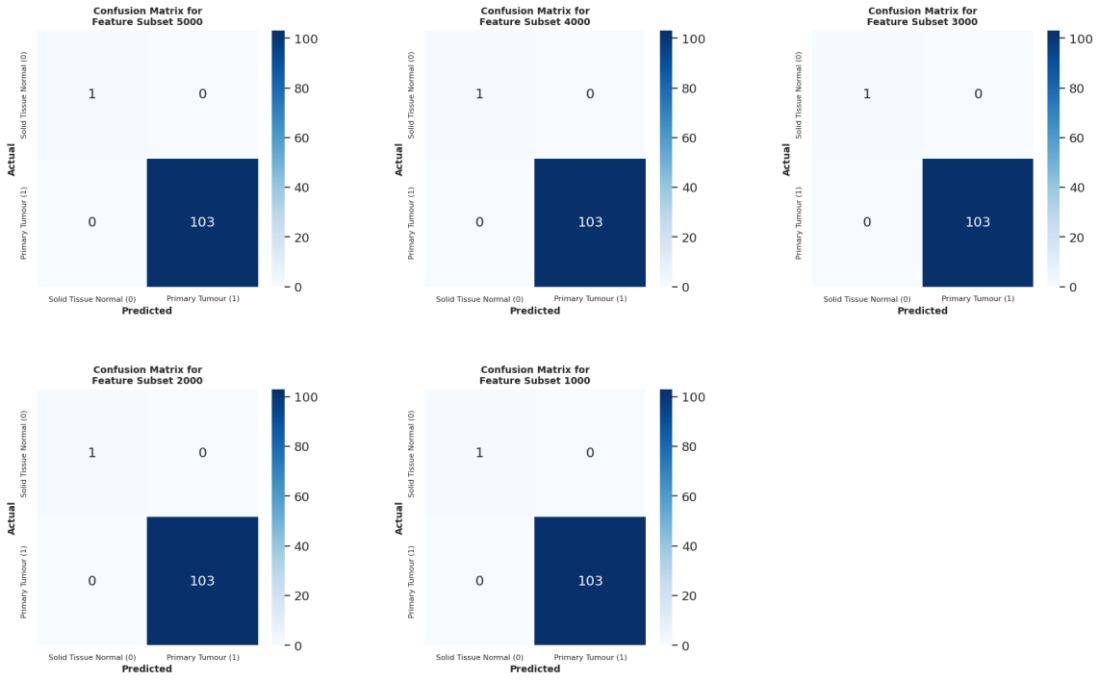


Figure 5.13 Confusion matrices produced using the classification results of SVM as the external classifier on the encoded inputs in the VAE model

Table 5.10 Metrics obtained from the classification result of VAE model using SVM

Feature Subset	Accuracy	AUC (ROC)	Precision	Recall	F1 score
5000	1.0000	1.0000	1.0000	1.0000	1.0000
4000	1.0000	1.0000	1.0000	1.0000	1.0000
3000	1.0000	1.0000	1.0000	1.0000	1.0000
2000	1.0000	1.0000	1.0000	1.0000	1.0000
1000	1.0000	1.0000	1.0000	1.0000	1.0000

5.3.4 Discussion on VAE Model

According to the model reconstruction loss during unsupervised learning for the VAE models, it is observed that the training set produced a larger loss value compared to the validation set for each feature subset. This is due to the fact the loss function defined for the VAE model is different from the BCE loss function used in the SDAE model. The reconstruction loss produced by the VAE model is directly related to the number of samples being used for the training/validation. Since the number of samples used for the training of the VAE model (70%) is larger than that of the validation set (30%), the model reconstruction loss produced is larger. Besides the overall model loss for the VAE model, the KL loss for each feature subset is observed to be larger in the validation set compared to the training set. The overall model loss for the VAE models resembles the model reconstruction loss, as the model reconstruction loss has much higher value compared to the value of the KL loss, which allows the model reconstruction loss to be the higher contributing factor to the overall loss of the VAE models.

In supervised learning, the two classification results produced by the two classification methods are vastly different. For the classification result obtained from the fine-tuning of the unsupervised learning VAE model into a supervised learning VAE model, the accuracy produced for each feature subset is very similar which is averaged at around 50% for both the training and testing set. These classification

results are severely underperforming and is far from the expectation, which is why the other method to obtaining the classification result is used. When the sampled data by the sampler in the VAE model are extracted and fed to the external SVM model, a vastly different result is obtained. It is observed that the SVM model is able to classify all the sampled data points correctly for each feature subset. These results are validated by observing the distribution of the sampled data points in form of scatterplot (Figure 5.10). According to the scatterplot, the sampled data points have clear distinction between their class labels, which is why the SVM is able to create a hyperplane that can effectively separate and classify the data points correctly. The comparison between metrics obtained using the two classification methods for the VAE model is tabulated in Table 5.11.

Table 5.11 Comparison between the metrics obtained from the classification result of the fine-tuned supervised learning VAE model and the classification using SVM

Feature Subset	**Accuracy**		**AUC (ROC)**		**Precision**		**Recall**		**F1 score**	
	FT	**SVM**	**FT**	**SVM**	**FT**	**SVM**	**FT**	**SVM**	**FT**	**SVM**
5000	0.500	1.000	0.252	1.000	0.981	1.000	0.505	1.000	0.667	1.000
4000	0.481	1.000	0.738	1.000	1.000	1.000	0.476	1.000	0.645	1.000
3000	0.462	1.000	0.728	1.000	1.000	1.000	0.456	1.000	0.627	1.000
2000	0.529	1.000	0.267	1.000	0.982	1.000	0.534	1.000	0.692	1.000
1000	0.471	1.000	0.238	1.000	0.980	1.000	0.476	1.000	0.605	1.000

5.4 Comparative Analysis between the SDAE and VAE Models

The comparison between the SDAE and VAE models is done in this subchapter. The hyperparameter used for both the SDAE and VAE models in both the unsupervised and supervised learning phases are recorded in Table 5.12 and Table 5.13 respectively.

Table 5.12 The hyperparameters used for the SDAE and VAE models in unsupervised learning

Hyperparameter	Unsupervised Deep Learning Models	
	SDAE	**VAE**
Layers	*5000, 5000 (noisy), 3500, 2000, 500, 2000, 3500, 5000* 100%, 100% (noisy), 70%, 40%, 10%, 40%, 70%, 100%	*5000, 3500, 2000, 500, 2000, 3500, 5000* 100%, 70%, 40%, 10%, 40%, 70%, 100%
Epoch	50	50
Batch size	16	16
Optimizer	adam	adam
Activation functions	ReLU (Rectified Linear Unit) – Hidden layers Sigmoid – Last layer (output layer)	ReLU (Rectified Linear Unit) – Hidden layers Linear – Bottleneck layer (latent layer) Sigmoid – Last layer (output layer)
Loss function	binary cross entropy	Generative loss Kullback-Leibler (KL) loss
Gaussian Noise Dropout Rate	10%	-

Table 5.13 The hyperparameters used for the SDAE and VAE models in supervised learning

Hyperparameter	Supervised Deep Learning Models	
	SDAE	**VAE**
Layers	*5000, 5000 (noisy), 3500, 2000, 500, 1* 100%, 100% (noisy), 70%, 40%, 10%, 1	*5000, 3500, 2000, 500, 1* 100%, 70%, 40%, 10%, 1
Epoch	50	50
Batch size	16	16
Optimizer	adam	adam
Activation	ReLU (Rectified Linear Unit)	ReLU (Rectified Linear Unit)

functions	– Hidden layers Sigmoid – Last layer (output layer)	– Hidden layers Linear – Bottleneck layer (latent layer) Sigmoid – Last layer (output layer)
Loss function	binary cross entropy	Generative loss Kullback-Leibler (KL) loss
Gaussian Noise Dropout Rate	10%	-

The comparisons include the model loss and the classification metrics obtained from the confusion matrix. The values of the model loss of the two autoencoders cannot be compared directly. This is due to the fact that the two neural networks used different loss functions, whereby the SDAE model used BCE while the VAE model used the combination of reconstruction loss and KL loss to produce an overall loss. Even so, it could be seen that the model loss for both the SDAE and VAE model showed converging trend for both training and validation set. However, the model loss for the SDAE model has smoother converging trend compared to the VAE model, especially in the validation set. When assessing the quality of the models solely on the model loss, it could be said that the SDAE models have higher quality and should perform better in classification than the VAE models however, further assessments are done below to confirm this.

For the comparison between the classification metrics produced by the confusion matrix of the two models, only the results obtained from the fine-tuned supervised learning SDAE model and the classification of the sampled data using SVM for the VAE model are considered. The classification result obtained from the fine-tuned supervised learning VAE model is not included due to the odd results produced by it, whereby the accuracies for all feature subsets are recorded at around 50%, which is below the expected values. The classification metrics obtained from the classification results of the two models are tabulated in Table 5.14 for side-by-side comparison.

Table 5.14 Comparison between the metrics obtained from the classification result of the fine-tuned supervised learning SDAE model and the classification using SVM on the sampled data by the VAE model

Feature Subset	Accuracy		AUC (ROC)		Precision		Recall		F1 score	
	SDAE (FT)	VAE (SVM)	SDAE (FT)	VAE (SVM)	SDAE (FT)	VAE (SVM)	SDAE (FT)	VAE (SVM)	SDAE (FT)	VAE (SVM)
5000	0.990	1.000	0.500	1.000	0.990	1.000	1.000	1.000	0.995	1.000
4000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
3000	0.990	1.000	0.500	1.000	0.990	1.000	1.000	1.000	0.995	1.000
2000	0.990	1.000	0.500	1.000	0.990	1.000	1.000	1.000	0.995	1.000
1000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

To visualize the tabulated metrics in Table 5.14, a grouped bar chart is plotted in Figure 5.14. All the classification metrics are listed on the X axis, with their values as the Y axis. The results of the SDAE models are coloured in blue gradients while the results of the VAE models are coloured in red gradients.

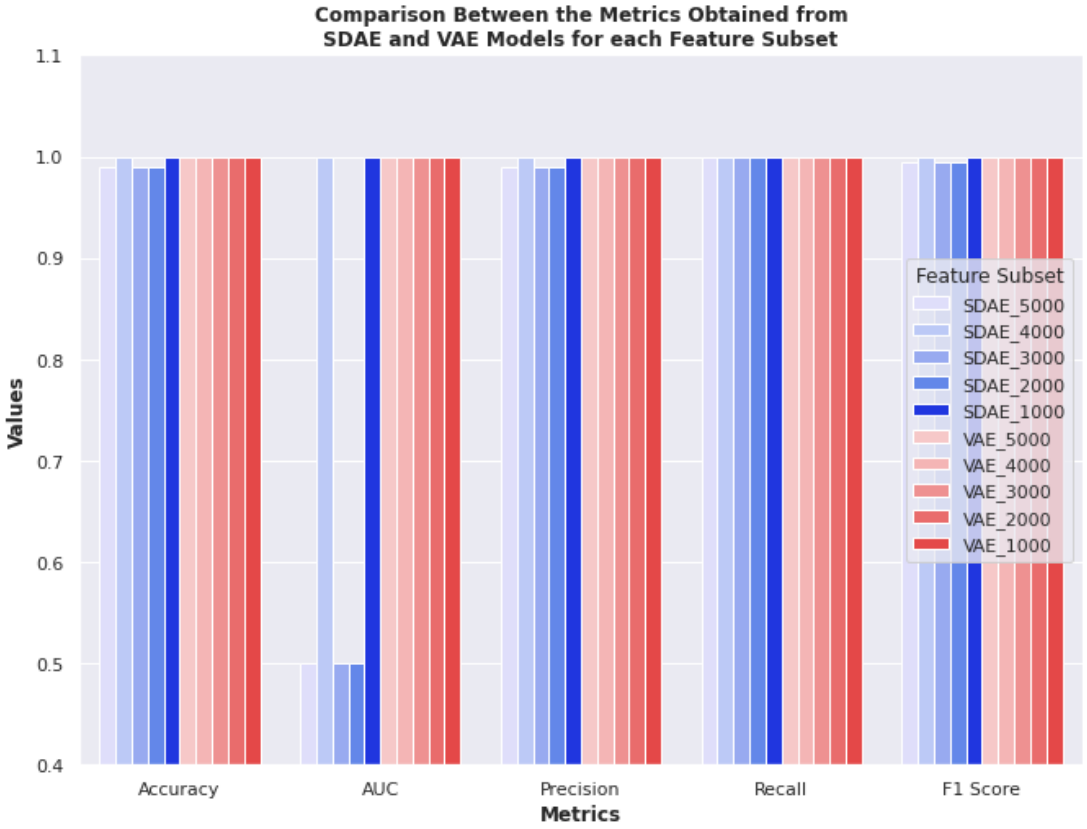


Figure 5.14 Comparison between the metrics obtained from the classification result of the fine-tuned supervised learning SDAE model and the classification using SVM on the sampled data by the VAE model

When observing the comparison chart for the classification metrics produced by the two autoencoders from the most general and broadest view, it could be observed that the results are seemingly outstanding and on par with each other, except for the AUC metric which is observed to have several dips for the SDAE model for feature subset 5000, 3000 and 2000. However, it is a mistake to compare and evaluate the values of these metrics without also observing the confusion matrices. This is an illusion produced by the classification results on the severely imbalanced testing set, which consists of only one instance of minority class label (i.e. Solid Tissue Normal) while having 103 instances of majority class label (i.e. Primary Tumour).

The baseline accuracy for this particular testing set is 0.9904, which is obtained by predicting only the majority class label. The SDAE models with feature subset 5000, 3000 and 2000 achieved the baseline accuracy, in which they are only capable of classifying the majority class label while misclassifying the minority label.

For precision, the scores obtained are similar to their accuracy for SDAE models with feature subset 5000, 3000 and 2000. This is due to the fact that precision measures the number of correctly classified positive outcomes (TP) out of the total predicted positive outcomes (TP + FP), in which the models have one FP. Since both the deep learning models are able to correctly classify the majority class labels for each feature subset, their recall is naturally recorded at 100%.

When looking at AUC scores of the two autoencoders for all feature subsets, it is noticeably obvious that the models built with feature subset 5000, 3000 and 2000 to have low AUC scores, recorded at only 0.5, while the rest of the models are recorded at 1.0. AUC measures the capability of a classification model to distinguish between the class labels. In this study, it measures how well the autoencoders can classify the samples into Solid Tissue Normal and Primary Tumour class labels. Since the testing set used in the study has only one instance for the minority class, which is also the negative class, correctly classifying this instance while also correctly classifying the positive classes will yield 1.0 for AUC. Conversely, misclassifying the only negative class while correctly classifying the positive classes will yield AUC of 0.5 instead, which is seen in the case of the SDAE models built with feature subset 5000, 3000 and 2000. AUC is a more relevant metric to assess the performance of the models in this study with the consideration of the severe class imbalance issue in the used testing set.

In conclusion, the VAE models built with all feature subsets outperform the SDAE models in every measured metrics. On the other hand, both SDAE and VAE models built with feature subset 1000 and 4000 are able to produce similar classification results. This could potentially indicate that the feature subset 1000 and 4000 are the two most optimal feature subsets with optimal number of features extracted by the SVM-RFE algorithm during the feature selection phase.

5.5 Chapter Summary

In this chapter, the results obtained from the unsupervised and supervised training of both the SDAE and VAE models using feature subset 1000, 2000, …, 5000 are discussed and analyzed. Furthermore, a comparative analysis between the performance of the two autoencoders is also carried out to assess the performance between the two models.

To conclude this chapter, the VAE models built using the same feature subsets are able to outperform the SDAE model in classification. Since FS 1000 and FS 4000 are the only two subsets which performed equally well in both the SDAE and VAE models, they are thought to be the most optimal subset of features selected by the SVM-RFE algorithm. At the end of this chapter, RO3 has been achieved.

CHAPTER 6

CONCLUSION

6.1 Introduction

This chapter aims to conclude the research. Firstly, literature review is done on the related algorithms to be used in this study, which involves SVM-RFE, SDAE and VAE. Then, the lung cancer omics data consisting three omics types are retrieved from the source stated in Chapter 3.3. A series of data-preprocessing steps are done on the raw omics dataset involving data transposition, data imputation, data normalization and VT analysis. The cleaned single omics data are then integrated through the concatenation of omics features into a multi-omics data. From the inspection of class label distribution before and after multi-omics integration, it is noticed that each of the single omics data are imbalanced with 90% majority class, and severely imbalanced for the integrated multi-omics data with 99% majority class.

The multi-omics data is split into train-test set with ratio 70:30. By using the train set, 20 feature subsets are selected using the SVM-RFE algorithm. Using the 20 selected feature subsets, the omics composition is taken note. It is observed that the composition of each omics type changes when the size of the feature subset decreases as more less relevant features from each omics types are removed by SVM-RFE (more details in Chapter 5.2).

The feature selected train set for each feature subset undergoes class balancing using SMOTE to handle the class imbalance issue. By using the balanced train set, the SDAE and VAE classification algorithms are developed. During the unsupervised training of both models, it is observed that both models produced converged model loss in 50 epochs. Both unsupervised SDAE and VAE models are fine-tuned into supervised learning model for classification (details in Chapter 4.3.6). The classification metrics involving the accuracy, precision, recall, F1-score and

ROC AUC are recorded for both models. In the end, the study concludes that, for this specific study, FS 1000 and FS 4000 are the two most optimal feature subsets selected by SVM-RFE as both the SDAE and VAE models produced 100% accuracy by using the two feature subsets.

This chapter also covers the achievement of the project. The constrains involved throughout the progress of the research are also mentioned along with the suggested improvements. Potential future work for this project which could be made to improve the study is also stated.

6.2 Achievement of Project

The achievements of the case study can be split into three parts according to the research objectives: 1) RO1 – literature review & data acquisition, 2) RO2 – project implementation, and 3) RO3 – assessment of results from SVM-RFE, SDAE & VAE.

In the early phase of the experiment, literature review is carried out to gain knowledge regarding the domain of study. The algorithms used in the experiment (SVM-RFE, SDAE & VAE) are studied to understand the flow and logic behind the operation of the algorithms, which are then used in the algorithm implementation phase. With sufficient knowledge, the study proceeded with data acquisition of the multi-omics data of lung cancer (LUSC). At the end of this phase, RO1 is achieved.

The next phase involves the implementation of the project using the methodology obtained. In pre-processing phase, the three raw omics datasets and the clinical dataset have been successfully cleaned and integrated into a single multi-omics dataset. These processes include data transposition, data imputation, data normalization and variance threshold analysis on the three omics datasets. With the cleaned and normalized omics dataset, a multi-omics dataset with 344 rows and 26130 columns is obtained after concatenating the three omics dataset with the class label extracted from the clinical dataset. The summary of the three omics datasets

and the clinical dataset before and after the preprocessing and multi-omics integration steps is shown in Table 6.1. The achievement for the pre-processing phase is the cleaned and integrated multi-omics dataset which is ready to be used for analysis.

Table 6.1 Summary of the datasets before and after data pre-processing and multi-omics integration

Dataset	**Dimension of the Dataset (row, column)**			
	Raw dataset	**Data Transposition**	**Variance Threshold**	**Multi-omics Integration**
Gene expression	(20531, 552)	(552, 20531)	(552, 20245)	(344, 26131)
DNA Methylation	(5000, 412)	(412, 5000)	(412, 5000)	
miRNA	(1046, 387)	(387, 1046)	(387, 886)	
Clinical Data	(626, 127)	(626, 127)	(626, 1)	

In the feature selection phase, 20 different feature subsets of multi-omics data are extracted from the original integrated multi-omics data. Through the help of the grid search algorithm, the most optimal set of hyperparameter for the SVM-RFE algorithm is determined. At the end of the feature selection phase, 20 optimal feature subset of multi-omics data is extracted to be used for deep learning classification.

In the deep learning classification phase, the SDAE and VAE models are built based on the 5 smallest feature subsets extracted by the SVM-RFE algorithm. The SDAE and VAE models which are first trained with unsupervised learning are successfully fine-tuned into supervised learning model capable of classification. Throughout data pre-processing to the implementation of the classification algorithms, RO2 is achieved.

The last phase discussed the results obtained from the classification of the feature subsets using SDAE and VAE models. From the result of the classification, it could be deduced that the VAE models for each feature subset outperforms the SDAE models when comparing their classification metrics. Other than that, since feature subset 1000 and 4000 are the only two feature subsets which enabled both the

SDAE and VAE models to achieve the highest score for each classification metrics, it could be deduced that feature subset 1000 and 4000 are the two most optimal feature subsets with optimal number of features selected by the SVM-RFE algorithm. After discussing the results, RO3 is achieved.

6.3 Research Constraint

Throughout the implementation of the study, there are two observed research constrain, which are 1) severely imbalanced dataset, and 2) insufficient GPU memory.

The final integrated multi-omics dataset is observed to be severely imbalanced. Table 6.2 shows the class distribution of each individual omics and the integrated multi-omics data. Before multi-omics integration, the single omics datasets are already imbalanced by having about a ratio of 90:10. After data integration, there is a total of 344 instances for the multi-omics data, in which only 3 instances are of Solid Tissue Normal class label, while the number of instances for Primary Tumour class label is recorded at 341. When converted to percentages, the majority class label, which is the Primary Tumour class label, composed of 99.1% of the full multi-omics data, while the minority class label Solid Tissue Normal is composed of only 0.9%.

Table 6.2 The class distribution of the datasets for each individual omics and the multi-omics data

Omics	**Primary Tumour (1)**	**Solid Tissue Normal (0)**	**Total Sample**
Gene Expression	501 (90.8%)	51 (9.2%)	552
DNA Methylation	370 (89.8%)	42 (10.2%)	412
miRNA Expression	342 (88.4%)	45 (11.6%)	387
Multi-omics Integration	**341 (99.1%)**	**3 (0.9%)**	**344**

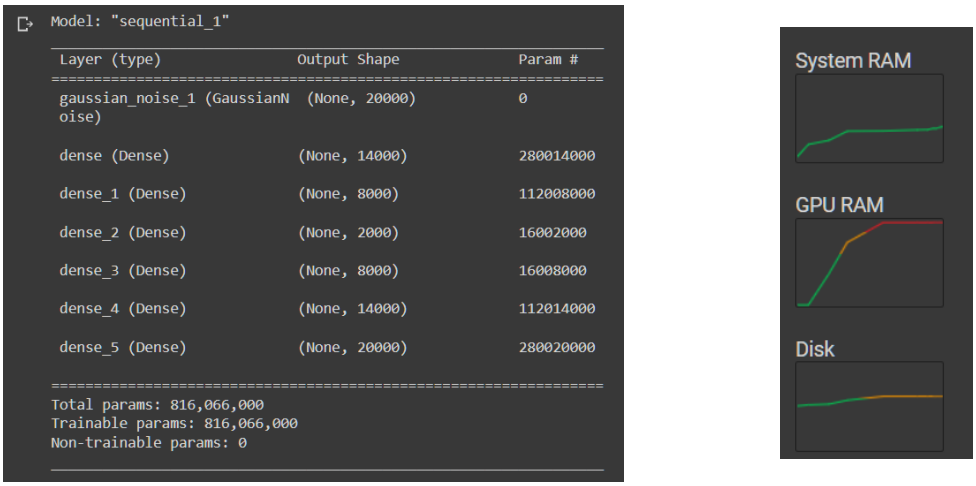
The effect from the imbalanced dataset becomes obvious when the multi-omics data is split into training and testing sets. The class distribution of the training and testing set is shown in Table 6.3. With 70%/30% of training/testing split, the training set will now hold 2 out of 3 instances with Solid Tissue Normal class label, while the testing set will hold 1 out of 3 instances with Solid Tissue Normal class label.

Table 6.3 The class distribution of the training and testing set

Dataset	**Primary Tumour (1)**	**Solid Tissue Normal (0)**
Training set	238	2
Testing set	103	1
Total	**341**	**3**

During the training phase of the models, new samples are synthesized with the help of SMOTE in order to increase the number of instances for Solid Tissue Normal class label to match the number of instances for Primary Tumour class label. Meanwhile, the testing set is still imbalanced, which has caused issues such as significantly increasing the baseline accuracy of any classification model to 0.9904 by predicting only the majority class labels. This has made the measurement of accuracy as an assessment metric less dependable, in which AUC metric is used to assess the performance of the models instead.

The next constrain for the study is related to the issue of insufficient memory of the GPU used for the research. This is a hardware constrain whereby the GPU offered by Google Colab does not offer an appropriate amount of memory to construct large neural network with millions of parameters. For example, the SDAE neural network built during unsupervised learning phase using feature subset 20000 yielded nearly 1 billion parameters or nodes as shown in Figure 6.1(a). This huge neural network has used up nearly 90% of the GPU memory as shown in Figure 6.1(b), causing the experiment to halt since the rest of the models could not be built.



(a) (b)

Figure 6.1 (a) The number of parameters/nodes required to create the neural network with feature subset 20000. (b) The GPU memory usage maxed out when creating the neural network in (a).

6.4 Suggestions for Improvement & Future Works

In order to cater the research constrains stated in the previous subchapter, two suggestions for improvement is provided. In order to solve the severe data imbalance problem, a new and updated multi-omics dataset could be used in place of the omics datasets used in this research. The new dataset has to be somewhat balanced and not have the similar data imbalance issue as the previous dataset.

As for the hardware constrain of the GPU, the study suggests that future researchers procure a workstation with better and stronger GPU to allow the construction of larger neural networks such as the neural network for the feature subset 20000. Aside from physically owning a better workstation, future researchers could also subscribe to the premium plan on the Google Colab platform to obtain a better GPU. Besides that, premium subscription on Google Colab also allows the researchers to have a longer session of experiments without frequent reconnection, which is a common problem for free users.

6.5 Chapter Summary

In this chapter, the overall conclusion of the study conducted has been summarized. The achievement of the study has been stated with detailed explanation. Besides, the problems and obstacles faced throughout the experiment in the study has been stated, along with potential solutions and improvement to solve the issues.

REFERENCES

Al-jabery, K. K., Obafemi-Ajayi, T., Olbricht, G. R., & Wunsch Ii, D. C. (2020). 4 - Selected approaches to supervised learning. In K. K. Al-jabery, T. Obafemi-Ajayi, G. R. Olbricht, & D. C. Wunsch Ii (Eds.), *Computational Learning Approaches to Data Analytics in Biomedical Applications* (pp. 101-123): Academic Press.

Bowers, A., & Zhou, X. (2019). Receiver Operating Characteristic (ROC) Area Under the Curve (AUC): A Diagnostic Measure for Evaluating the Accuracy of Predictors of Education Outcomes. *Journal of Education for Students Placed at Risk (JESPAR), 24*, 1-25. doi:10.1080/10824669.2018.1523734

Chandra, B., & Sharma, R. K. (2016). Fast learning in Deep Neural Networks. *Neurocomputing*, *171*, 1205-1215. doi:https://doi.org/10.1016/j.neucom.2015.07.093

Chaunzwa, T. L., Hosny, A., Xu, Y., Shafer, A., Diao, N., Lanuti, M., . . . Aerts, H. J. W. L. (2021). Deep learning classification of lung cancer histology using CT images. *Scientific Reports, 11*(1). doi:10.1038/s41598-021-84630-x

Committee on the Review of Omics-Based Tests for Predicting Patient Outcomes in Clinical, T., Board on Health Care, S., Board on Health Sciences, P., & Institute of, M. (2012). In C. M. Micheel, S. J. Nass, & G. S. Omenn (Eds.), *Evolution of Translational Omics: Lessons Learned and the Path Forward*. Washington (DC): National Academies Press (US)

Coorssen, J. R. (2013). Proteomics. In S. Maloy & K. Hughes (Eds.), *Brenner's Encyclopedia of Genetics (Second Edition)* (pp. 508-510). San Diego: Academic Press.

Coudray, N., Ocampo, P. S., Sakellaropoulos, T., Narula, N., Snuderl, M., Fenyö, D., . . . Tsirigos, A. (2018). Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning. *Nat Med, 24*(10), 1559-1567. doi:10.1038/s41591-018-0177-5

Davidson, M. R., Gazdar, A. F., & Clarke, B. E. (2013). The pivotal role of pathology in the management of lung cancer. *Journal of thoracic disease, 5 Suppl 5*(Suppl 5), S463-478. doi:10.3978/j.issn.2072-1439.2013.08.43

Dayalan, S., Xia, J., Spicer, R. A., Salek, R., & Roessner, U. (2019). Metabolome Analysis. In S. Ranganathan, M. Gribskov, K. Nakai, & C. Schönbach (Eds.), *Encyclopedia of Bioinformatics and Computational Biology* (pp. 396-409). Oxford: Academic Press.

Dickson, B. (2019). What are artificial neural networks (ANN)? Retrieved from https://bdtechtalks.com/2019/08/05/what-is-artificial-neural-network-ann/

Diebold, F. (2000). Big Data" Dynamic Factor Models for Macroeconomic Measurement and Forecasting.

Dinesh, T., Ashiq, M., & Joselin, J. (2018). A Review on Neural Networks. *International Journal of Trend in Scientific Research and Development*, 565-569.

Dong, G., Liao, G., Liu, H., & Kuang, G. (2018). A Review of the Autoencoder and Its Variants: A Comparative Perspective from Target Recognition in Synthetic-Aperture Radar Images. *IEEE Geoscience and Remote Sensing Magazine, 6*, 44-68. doi:10.1109/MGRS.2018.2853555

El-Manzalawy, Y., Hsieh, T.-Y., Shivakumar, M., Kim, D., & Honavar, V. (2018). Min-redundancy and max-relevance multi-view feature selection for predicting ovarian cancer survival using multi-omics data. *BMC Medical Genomics, 11*(3), 71. doi:10.1186/s12920-018-0388-0

Emmert-Streib, F., Yang, Z., Feng, H., Tripathi, S., & Dehmer, M. (2020). An Introductory Review of Deep Learning for Prediction Models With Big Data. *Frontiers in Artificial Intelligence, 3*(4). doi:10.3389/frai.2020.00004

Gao, Y., Zhou, R., & Lyu, Q. (2020). Multiomics and machine learning in lung cancer prognosis. *Journal of thoracic disease, 12*(8), 4531-4535. doi:10.21037/jtd-2019-itm-013

Gholamy, A., Kreinovich, V., & Kosheleva, O. (2018). *Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation*.

Gondara, L. (2016). Medical Image Denoising Using Convolutional Denoising Autoencoders. *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 241-246.

Gordon-Rodriguez, E., Loaiza-Ganem, G., Pleiss, G., & Cunningham, J. P. (2020). Uses and Abuses of the Cross-Entropy Loss: Case Studies in Modern Deep Learning. doi:10.48550/ARXIV.2011.05231

Gupta, A. (2021). A Comprehensive Guide on Deep Learning Optimizers. Retrieved from https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res., 3*(null), 1157–1182.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning, 46*(1), 389-422. doi:10.1023/A:1012487302797

Hasin, Y., Seldin, M., & Lusis, A. (2017). Multi-omics approaches to disease. *Genome Biology, 18*(1), 83. doi:10.1186/s13059-017-1215-1

Hira, M. T., Razzaque, M. A., Angione, C., Scrivens, J., Sawan, S., & Sarkar, M. (2021). Integrated multi-omics analysis of ovarian cancer using variational autoencoders. *Scientific Reports, 11*(1), 6265. doi:10.1038/s41598-021-85285-4

Hoang, G., Udupa, S., & Le, A. (2019). Chapter Six - Application of metabolomics technologies toward cancer prognosis and therapy. In D. C. Montrose & L. Galluzzi (Eds.), *International Review of Cell and Molecular Biology* (Vol. 347, pp. 191-223): Academic Press.

Huang, M.-L., Hung, Y.-H., Lee, W. M., Li, R. K., & Jiang, B.-R. (2014). SVM-RFE Based Feature Selection and Taguchi Parameters Optimization for Multiclass SVM Classifier. *The Scientific World Journal, 2014*, 795624. doi:10.1155/2014/795624

Huynh, P.-H., Nguyen, V. H., & Do, T.-N. (2020). Improvements in the Large p, Small n Classification Issue. *SN Computer Science, 1*(4), 207. doi:10.1007/s42979-020-00210-2

Jan, B., Farman, H., Khan, M., Imran, M., Islam, I., Ahmad, A., . . . Jeon, G. (2017). Deep learning in big data Analytics: A comparative study. *Computers & Electrical Engineering, 75*. doi:10.1016/j.compeleceng.2017.12.009

Jiang, Z. (2021). SDAE-based feature selection method for biological Omics data. *Journal of Physics: Conference Series, 1848*(1), 012022. doi:10.1088/1742-6596/1848/1/012022

Jordan, J. (2018). Introduction to autoencoders. Retrieved from https://www.jeremyjordan.me/autoencoders/

Kaczor-Urbanowicz, K. E., & Wong, D. T. W. (2020). 4 - Proteomics. In S. T. Sonis & A. Villa (Eds.), *Translational Systems Medicine and Oral Disease* (pp. 93-118): Academic Press.

Kalavacharla, V., Subramani, M., Ayyappan, V., Dworkin, M. C., & Hayford, R. K. (2017). Chapter 16 - Plant Epigenomics. In T. O. Tollefsbol (Ed.), *Handbook of Epigenetics (Second Edition)* (pp. 245-258): Academic Press.

Kang, H. (2013). The prevention and handling of the missing data. *Korean J Anesthesiol, 64*(5), 402-406. doi:10.4097/kjae.2013.64.5.402

Kavlakoglu, E. (2020). AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference? Retrieved from https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *CoRR, abs/1609.04836*. Retrieved from http://arxiv.org/abs/1609.04836

Khan, S., & Yairi, T. (2018). A review on the application of deep learning in system health management. *Mechanical Systems and Signal Processing, 107*, 241-265. doi:https://doi.org/10.1016/j.ymssp.2017.11.024

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97*(1), 273-324. doi:https://doi.org/10.1016/S0004-3702(97)00043-X

Kriegsmann, M., Haag, C., Weis, C.-A., Steinbuss, G., Warth, A., Zgorzelski, C., . . . Kriegsmann, K. (2020). Deep Learning for the Classification of Small-Cell and Non-Small-Cell Lung Cancer. *Cancers, 12*(6), 1604. doi:10.3390/cancers12061604

Kuhn, M., & Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models (1st ed.). Chapman and Hall/CRC*.

Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*.

Lal, T. N., Chapelle, O., Weston, J., & Elisseeff, A. (2006). Embedded Methods. In I. Guyon, M. Nikravesh, S. Gunn, & L. A. Zadeh (Eds.), *Feature Extraction:*

Foundations and Applications (pp. 137-165). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lin, E., & Lane, H.-Y. (2017). Machine learning and systems genomics approaches for multi-omics data. *Biomarker Research, 5*. doi:10.1186/s40364-017-0082-y

Lipton, Z. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning.

Martin-Sanchez, F., Lopez-Campos, G., & Gray, K. (2014). Chapter 11 - Biomedical Informatics Methods for Personalized Medicine and Participatory Health. In I. N. Sarkar (Ed.), *Methods in Biomedical Informatics* (pp. 347-394). Oxford: Academic Press.

Mhatre, M. S., Siddiqui, F., Dongre, M., & Thakur, P. A review paper on artificial neural network: a prediction technique.

Milne, G. L., & Morrow, J. D. (2009). Chapter 5 - Measurement of Biological Materials. In D. Robertson & G. H. Williams (Eds.), *Clinical and Translational Science* (pp. 69-86). San Diego: Academic Press.

Milward, E. A., Shahandeh, A., Heidari, M., Johnstone, D. M., Daneshi, N., & Hondermarck, H. (2016). Transcriptomics. In R. A. Bradshaw & P. D. Stahl (Eds.), *Encyclopedia of Cell Biology* (pp. 160-165). Waltham: Academic Press.

Monn, D. (2017). Denoising Autoencoders explained. Retrieved from https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2

Morabito, F. C., Campolo, M., Ieracitano, C., & Mammone, N. (2019). Chapter 11 - Deep Learning Approaches to Electrophysiological Multivariate Time-Series Analysis∗∗To my loved daughter, Valeria. In R. Kozma, C. Alippi, Y. Choe, & F. C. Morabito (Eds.), *Artificial Intelligence in the Age of Neural Networks and Brain Computing* (pp. 219-243): Academic Press.

Nielsen, M. (2019). Why are deep neural networks hard to train? *Neural Networks and Deep Learning.* Retrieved from http://neuralnetworksanddeeplearning.com/chap5.html

Pereira Braga, C., & Adamec, J. (2019). Metabolome Analysis. In S. Ranganathan, M. Gribskov, K. Nakai, & C. Schönbach (Eds.), *Encyclopedia of*

Bioinformatics and Computational Biology (pp. 463-475). Oxford: Academic Press.

Perez-Riverol, Y., Kuhn, M., Vizcaino, J., Hitz, M., & Audain, E. (2017). Accurate and fast feature selection workflow for high-dimensional omics data. *PLoS One, 12*. doi:10.1371/journal.pone.0189875

Picard, M., Scott-Boyer, M.-P., Bodein, A., Périn, O., & Droit, A. (2021). Integration strategies of multi-omics data for machine learning analysis. *Computational and Structural Biotechnology Journal, 19*, 3735-3746. doi:https://doi.org/10.1016/j.csbj.2021.06.030

Pinu, F. R., Beale, D. J., Paten, A. M., Kouremenos, K., Swarup, S., Schirra, H. J., & Wishart, D. (2019). Systems Biology and Multi-Omics Integration: Viewpoints from the Metabolomics Research Community. *Metabolites, 9*(4), 76. doi:10.3390/metabo9040076

Rocca, J. (2019). Understanding Variational Autoencoders (VAEs). Retrieved from https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Ruby, U., & Yendapalli, V. (2020). Binary cross entropy with deep learning technique for Image classification. *International Journal of Advanced Trends in Computer Science and Engineering, 9*. doi:10.30534/ijatcse/2020/175942020

Saeys, Y., Inza, I., & Larranaga, P. (2007). A review of feature selection techniques in bioinformatics. *bioinformatics, 23*(19), 2507-2517.

Samb, M. L., Camara, F., N'Diaye, S., Slimani, Y., Esseghir, M., & Anta, C. (2012). *A Novel RFE-SVM-based Feature Selection Approach for Classification*.

Sanz, H., Valim, C., Vegas, E., Oller, J. M., & Reverter, F. (2018). SVM-RFE: selection and visualization of the most relevant features through non-linear kernels. *BMC Bioinformatics, 19*(1), 432. doi:10.1186/s12859-018-2451-4

Sharma, S. (2017). Artificial Neural Network (ANN) in Machine Learning. Retrieved from https://www.datasciencecentral.com/profiles/blogs/artificial-neural-network-ann-in-machine-learning

Shaziya, H. (2020). A Study of the Optimization Algorithms in Deep Learning. doi:10.1109/ICISC44355.2019.9036442

Singh, D., & Singh, B. (2020). Investigating the impact of data normalization on classification performance. *Applied Soft Computing, 97*, 105524. doi:https://doi.org/10.1016/j.asoc.2019.105524

Subramanian, I., Verma, S., Kumar, S., Jere, A., & Anamika, K. (2020). Multi-omics Data Integration, Interpretation, and Its Application. *Bioinformatics and biology insights, 14*, 1177932219899051-1177932219899051. doi:10.1177/1177932219899051

Tadist, K., Najah, S., Nikolov, N. S., Mrabti, F., & Zahi, A. (2019). Feature selection methods and genomic big data: a systematic review. *Journal of Big Data, 6*(1), 79. doi:10.1186/s40537-019-0241-0

Taguchi, Y. h., & Turki, T. (2022). Novel feature selection method via kernel tensor decomposition for improved multi-omics data analysis. *BMC Medical Genomics, 15*(1), 37. doi:10.1186/s12920-022-01181-4

Ulfenborg, B. (2019). Vertical and horizontal integration of multi-omics data with miodin. *BMC Bioinformatics, 20*(1), 649. doi:10.1186/s12859-019-3224-4

Uppal, K. (2021). Models of Metabolomic Networks. In O. Wolkenhauer (Ed.), *Systems Medicine* (pp. 134-142). Oxford: Academic Press.

Vikashrajluhaniwal. (2020). A comprehensive guide to Feature Selection using Wrapper methods in Python. Retrieved from https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/#:~:text=In%20wrapper%20methods%2C%20the%20feature,fit%20on%20a%20given%20dataset.&text=Finally%2C%20it%20selects%20the%20combination,the%20specified%20machine%20learning%20algorithm.

Weng, L. (2017). An Overview of Deep Learning for Curious People. Retrieved from https://lilianweng.github.io/lil-log/2017/06/21/an-overview-of-deep-learning.html#why-does-deep-learning-work-now

Wörheide, M. A., Krumsiek, J., Kastenmüller, G., & Arnold, M. (2021). Multi-omics integration in biomedical research – A metabolomics-centric review. *Analytica Chimica Acta, 1141*, 144-162. doi:https://doi.org/10.1016/j.aca.2020.10.038

Xu, Y., She, Y., Li, Y., Li, H., Jia, Z., Jiang, G., . . . Duan, L. (2020). Multi-omics analysis at epigenomics and transcriptomics levels reveals prognostic

subtypes of lung squamous cell carcinoma. *Biomedicine & Pharmacotherapy, 125*, 109859. doi:https://doi.org/10.1016/j.biopha.2020.109859

Yadav, D., Tanveer, A., Malviya, N., & Yadav, S. (2018). Chapter 1 - Overview and Principles of Bioengineering: The Drivers of Omics Technologies. In D. Barh & V. Azevedo (Eds.), *Omics Technologies and Bio-Engineering* (pp. 3-23): Academic Press.

Yadav, S. P. (2007). The wholeness in suffix -omics, -omes, and the word om. *Journal of biomolecular techniques : JBT, 18*(5), 277-277. Retrieved from https://pubmed.ncbi.nlm.nih.gov/18166670

Yang, H., Chen, L., Cheng, Z., Yang, M., Wang, J., Lin, C., . . . Li, W. (2021). Deep learning-based six-type classifier for lung cancer and mimics from histopathological whole slide images: a retrospective study. *BMC Medicine, 19*(1). doi:10.1186/s12916-021-01953-2

Yu, K. H., Wang, F., Berry, G. J., Ré, C., Altman, R. B., Snyder, M., & Kohane, I. S. (2020). Classifying non-small cell lung cancer types and transcriptomic subtypes using convolutional neural networks. *J Am Med Inform Assoc, 27*(5), 757-769. doi:10.1093/jamia/ocz230

Yu, L.-R., Stewart, N. A., & Veenstra, T. D. (2010). Chapter 8 - Proteomics: The Deciphering of the Functional Genome. In G. S. Ginsburg & H. F. Willard (Eds.), *Essentials of Genomic and Personalized Medicine* (pp. 89-96). San Diego: Academic Press.

Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation, 31*(7), 1235-1270. doi:10.1162/neco_a_01199

Zhou, M., Varol, A., & Efferth, T. (2021). Multi-omics approaches to improve malaria therapy. *Pharmacological Research, 167*, 105570. doi:https://doi.org/10.1016/j.phrs.2021.105570

Zhu, J., & Hastie, T. (2004). Classification of Gene Microarrays by Penalized Logistic Regression. *Biostatistics (Oxford, England), 5*, 427-443. doi:10.1093/biostatistics/5.3.427

Appendix A Gantt Chart

	Task Name	Duration	Start	End
	PSM 1	35 days	21.03.2021	
1	Chapter 1 Introduction	6 days	21.03.2021	26.03.2021
1.1	Introduction	2 days	21.03.2021	22.03.2021
1.2	Problem Background	2 days	23.03.2021	24.03.2021
1.3	Research Aim	1 day	25.03.2021	25.03.2021
1.4	Research Question	1 day	25.03.2021	25.03.2021
1.5	Research Objective	1 day	25.03.2021	25.03.2021
1.6	Research Scope	1 day	26.03.2021	26.03.2021
1.7	Research Contribution	1 day	26.03.2021	26.03.2021
1.8	Report Organization	1 day	26.03.2021	26.03.2021
2	Chapter 2 Literature Review	40 days	27.03.2021	05.05.2021
2.1	Introduction	2 days	27.03.2021	28.03.2021
2.2	Lung Cancer Classification	2 days	29.03.2021	30.03.2021
2.3	Omics	4 days	31.03.2021	03.04.2021
2.4	Multi-omics	4 days	04.04.2021	07.04.2021
2.5	Feature Selection	4 days	08.04.2021	11.04.2021
2.6	Support Vector Machine (SVM)	6 days	12.04.2021	17.04.2021
2.7	Artificial Neural Network (ANN)	3 days	18.04.2021	20.04.2021
2.8	Deep Learning (DL)	6 days	21.04.2021	26.04.2021
2.9	Autoencoder (AE)	8 days	27.04.2021	04.05.2021
2.10	Chapter Summary	1 day	05.05.2021	05.05.2021
3	Chapter 3 Methodology	16 days	06.05.2021	21.05.2021
3.1	Introduction	2 days	06.05.2021	07.05.2021
3.2	Research Framework	7 days	08.05.2021	14.05.2021
3.3	Dataset	3 days	15.05.2021	17.05.2021
3.4	Performance Measure	3 days	18.05.2021	20.05.2021
3.5	Chapter Summary	1 days	21.05.2021	21.05.2021
4	Chapter 4 Research Design and Implementation	60 days	22.05.2021	20.07.2021
4.1	Introduction	2 days	22.05.2021	23.05.2021
4.2	Experiment Design	14 days	24.05.2021	06.06.2021
4.3	Parameter and Testing Method	58 days	24.05.2021	20.07.2021
4.4	Chapter Summary	2 days	19.07.2021	20.07.2021
5	Chapter 5 Conclusion	7 days	21.07.2021	27.07.2021
5.1	Introduction	2 days	21.07.2021	22.07.2021
5.2	Achievement of Project	5 days	23.07.2021	27.07.2021
5.3	Future Works	5 days	23.07.2021	27.07.2021