Multivariate Analysis of Single Cell Transcriptomics of Cancer Tumours

Authors:
Grace Carmichael
Campbell McDuling

Student Numbers:
CRMGRA004
MCDCAM001

April, 2023

Contents

1 Abstract

The classification of patients' samples and identification of key characteristics of these classes is an important field in medical research. Recent advances in single cell RNA-sequencing (scRNA-Seq) technology have enabled the transcriptomic profiling of samples at a single-cell resolution. This presents the possibility of classifying and analysing samples at this fine resolution, however, the data often exhibits high dimensionality and thus the application of multivariate methods is necessary. This report seeks to demonstrate the application of multivariate dimension reduction and clustering techniques with the goal of classifying cell populations, while outlining the general approach which is typically taken in these analyses.

2 Introduction and Literature Review

The study of gene expression profiles on a single-cell basis can be extremely useful in medical research. This is especially true in oncological research, which often entails the analysis of tumour tissue samples – comprising of an assortment of different and mutating cell-types. Single cell transcriptomics (SCT) refers to the RNA sequencing of individual cells in a population of cells. The messenger RNA from a large number of genes is sequenced and read counts are obtained. The abundance of a particular mRNA corresponds to the level of expression of the gene that the RNA comes from. The benefit of SCT over normal RNA sequencing of a sample is that differences in expression can be detected at the resolution of single cells and not just at sample resolution [24]. This allows the discovery of heterogeneity between cells and cell types, identification of rare cell populations, distinct cell-lineage trajectories and mechanisms involved in complex cellular processes [24].

The data obtained from SCT is categorized by high-dimensionality and sparseness. Usually each cell can express a large number of genes - from tens to hundreds of thousands depending on the sample. The result is large multivariate datasets, usually containing many more genes than cells (there is often multicollinearity in the data). Furthermore, it is common for these data sets to have many zero entries as many of the genes expressed in some cells may not to be expressed in other cells. This presents the so-called "Curse of Dimensionality", a fundamental problem in statistical analysis that arises when dealing with data that have a large number of variables or features. Sparseness can compound this problem, and traditionally the data is subjected to transformations and feature selection to reduce the noise-to-signal ratio. The high dimensionality can make statistical analyses challenging, as the size and complexity of the data can quickly become overwhelming and introduce major computational inefficiencies. This can make it difficult to accurately cluster cells, identify meaningful patterns in the data, or perform other common statistical analyses.

Most SCT analyses thus begin by reducing the dimensionality of the data. This can be done through two general approaches. The first method involves generating a lower-dimensional representation of the data which preserves some underlying characteristic (such as between-cell heterogeneity) through some statistical procedure such as PCA or tSNE. The second method involves implementing some feature selection algorithm to exclude uninformative genes and/or unabundant genes from the data. It is usually beneficial to use a combination of these approaches to reduce the feature space optimally [2].

It is common practice to use unsupervised clustering algorithms on the reduced feature space to distinguish heterogeneous cell populations. Andrews and Hemberg provide reviews of clustering for cell identification from sc-RNASeq data ([2], [3]). These methods can be broadly grouped into non-hierarchical, hierarchical, and graph-based clustering algorithms. The most frequently used non-hierarchical algorithm is k-means. K-means is computationally efficient and scales well to large datasets. However, this assumes a predetermined number of cell populations of similar shape and size. As a result, k-means performs poorly in distinguishing rare cell types - often merging them with larger true clusters. Another disadvantage of k-means is that it uses a stochastic initialisation, resulting in a non-unique solution that does not guarantee a global minimum for the objective function [2]. Hierarchical agglomerative algorithms do not require the pre-specification of the number of clusters, but also assume clusters of similar shape and size. The advantage of hierarchical methods is that the results can be visualized using a dendrogram and clusters of different granularities can be

investigated. For example, one could base the assignment of cells off of two clusters which represent plastic vs. non-plastic cells, or generate more granular clusters where different types of plastic and non-plastic cells are distinguished. In other words, there is often an inherent hierarchy to cell type, and hierarchical algorithms by design allow this to be explored. These methods, however, are not as computationally efficient and thus do not scale well to large data with many samples. Further methods have been explored, such as density-based clustering algorithms like DBSCAN – however these assume all cell populations are equally homogeneous [9]. These approaches all thus have some obvious limitations relating to the identification of less frequent cell types and cell types with different between-cell homogeneity [2] .

Graph-based clustering, also known as community-based detection, is a category of density-based clustering algorithms applied to data that is represented in graph form. In this case, cells would be represented by nodes - with edges connecting the nodes representing the degree of similarity in gene expression. These edges connecting cells are typically determined by a cell's k-nearest neighbours, which is often performed on the reduced feature space to combat the drawbacks of high dimensionality [2]. The Louvian algorithm, or some adaptation of it, is then used to partition the graph into densely connected clusters by optimising some objective function [5]. The objective function is typically the modularity of the partition - a measure of the density of networks within clusters compared to that between clusters. While graph-based clustering overcomes some of the limitations of other unsupervised algorithms, there is evidence that the Louvian algorithm performs poorly on smaller data sets [10].

Various adaptations and combinations of these algorithms have been developed for the specific application of single cell classification, almost all of which entail some combination of dimension reduction and unsupervised clustering. A number of software packages have been developed to provide a self-contained workflow for this analysis in R, Matlab, Python and other software environments. The performance of the more commonly used packages have been quantitatively benchmarked against each other in recent literature ([11], [12], [22], [23]). Among these studies, there is a consensus on the superior performance and computational efficiency of the R package *Seurat* over many of the other methods - especially with respect to relatively larger data sets. Along with facilitating data pre-processing, *Seurat* packages the above mentioned methods - dimension reduction (PCA) and clustering (graph-based) - into one integrated workflow [13].

One of the major challenges of unsupervised clustering approaches is that, even if the chosen algorithm correctly distinguishes cell populations, the assignment of cell cluster to cell population is typically done manually by examining the differentiated mean gene-expression profiles of the clusters. This is neither an efficient nor a trivial task, and is heavily reliant both on the clustering result and the clinical expertise of the investigator. There have thus been recent attempts to develop semi-supervised and fully-supervised classifiers to overcome some of the obstacles the unsupervised methods present. The semi-supervised methods usually entail using some (semi)supervised algorithm to match gene-markers from cell clusters obtained from unsupervised methods to a labelled reference sample. Fully-supervised classifiers typically train deep learning algorithms on labelled training data, after which the models can be used to directly predict cell type based on the gene-expression profile. The performance of a selection of these classifiers have been compared in recent literature ([1], [28]). As yet, there is little evidence to show that these approached have been fully accepted or utilised in the field however promising results from the initial publications indicate that the supervised approach may gain traction in future research.

This report seeks to illustrate how dimension reduction techniques and clustering algorithms can be used on SCT data to distinguish cell types, illustrating the techniques on scRNA-Seq data from breast cancer cells and lymph node metastases. Section 3.1 describes the data used and the pre-processing conventionally done in these analyses; Section 3.2 describes the dimension reduction and unsupervised clustering methods in detail, and presents results from application to the data. Section 3.3 presents a selection of supervised techniques and the resulting application to the data. Lastly, in section 4 the key aspects of this report are summarised and discussed.

3 Methods and Results

3.1 Data pre-processing

The data was obtained from the NCBI Gene Expression Omnibus (GEO) at the following link: `https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE75688`. The data comes from Chung, W. et al. (2017) and contains RNA-seq data from 515 primary breast cancer cells and lymph node metastases from 11 patients with with distinct molecular cancer subtypes. The data contains expression levels for 57912 genes across these 515 cells. The data was pre-processed by the authors by removing low quality samples and log normalising the transcripts per million (TPM). This normalisation involves dividing the gene expression for each cell by the overall gene expression in the sample, multiplying this by a scale factor of 1000000 and then $\log_2$ transforming the result. The data comes in the form of the full expression matrix (with the low quality samples and pooled sample expressions also included) and a data frame containing sample information (this file has the low quality samples removed). The sample information includes the classification of the cells which will be used to validate the groupings and clusters identified in the data. There is a hierarchy to the definition of cell type in this case, which is visualised in Figure 1. The primary labeling distinguishes 317 tumour cells from 198 non-tumour cells; the secondary labels show that most of the non-tumour samples are immune cells while only 23 are stromal cells; and the tertiary labels classify the immune cells into three sub-types.

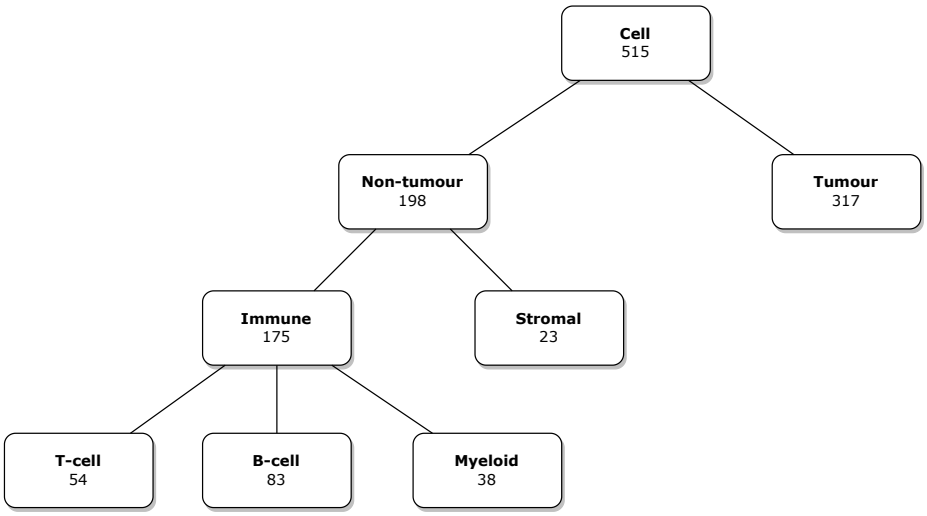


Figure 1: Cell Classification Hierarchy

To process the data, the poor quality samples were first removed from the expression matrix and the sample information data frame was ordered alphabetically to match the expression matrix. The pooled samples were also removed from the matrix as only the single cell expression is of interest for this analysis. The R package *Seurat* was used to scale the data [13]. The scaling is just standard scaling which involves subtracting the mean gene expression and dividing by the standard deviation of the gene expression for each gene. *Seurat* is often used to perform the normalisation if the data does not come normalised. *Seurat* was also used to reduce the data by removing genes with lower variability across cells. This is typically done in single-cell transcriptomics to make the data more manageable by removing genes that are not of interest in the analysis. The 35000 most variable genes were retained for the analysis. Table 1 shows the first 5 rows and columns of the normalised and scaled expression data. This dataset was transposed for many of the analyses to get

genes as variables (columns) and cells as observations (rows).

Table 1: First 5 rows and columns of the normalised and scaled expression data.

Gene ID	**BC01-02**	**BC01-03**	**BC01-04**	**BC01-05**	**BC01-06**
ENSG00000000003.10	-0.504924661	-0.504924661	-0.504924661	-0.504924661	-0.504924661
ENSG00000000005.5	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
ENSG00000000419.8	-0.004722714	0.759761812	-0.694676986	1.260276920	-0.254108211
ENSG00000000457.9	-0.257333903	-0.374661576	2.765090048	-0.107262673	1.910627258
ENSG00000000460.12	-0.298664980	0.305929257	-0.282194957	-0.286029565	-0.298664980

3.2 Unsupervised techniques

In this section, some unsupervised techniques for dimension reduction and identification of groupings in the data are presented. These are all analysis techniques that are typically implemented in the field of single-cell transcriptomics, with the exception of kernel PCA. This was implemented as an additional method that could be applicable and as the first step for an extension to support vector machines in the supervised analysis section.

3.2.1 Principal Component Analysis (PCA)

Typically, the first step in analysing single-cell transcriptomic data involves some kind of dimension reduction technique. One example of such a technique is PCA, which involves identifying principal components ($\boldsymbol{Y}$) which are linear combinations of the measured variables ($\boldsymbol{X}$). This model takes the form:

$$\boldsymbol{Y} = \boldsymbol{a}\boldsymbol{X} \tag{1}$$

Where $\boldsymbol{a}$ is made up of the eigenvectors of the covariance matrix of $\boldsymbol{X}$ (ordered from eigenvector corresponding to the largest eigenvalue to eigenvector corresponding to the smallest eigenvalue). To express the data in a lower dimension, typically only the first few principal components are used to express the variability in the data. In single cell transcriptomics, $\boldsymbol{X}$ is given by the genes and the goal is to identify principal components that are able to capture the variability in these genes.

PCA was applied to the breast cancer data to determine if cell groupings could be recovered in fewer dimensions and to reduce the dimensions of the data for subsequent analyses. The scaling of the data should make the PCA more successful, despite the non-linearity in the data. The first 2 principal components only explain a very small percentage of the variability in the data (around 2%) and the first 10 only explain around 4%. Despite this, the plots in Figure 2 indicate that the first 2 principal components still seem to group the tumour and non-tumour cells relatively well. They do not group the sub-types of cells or the cancer type/samples very well though (the plots demonstrating this can be found in Appendix A).

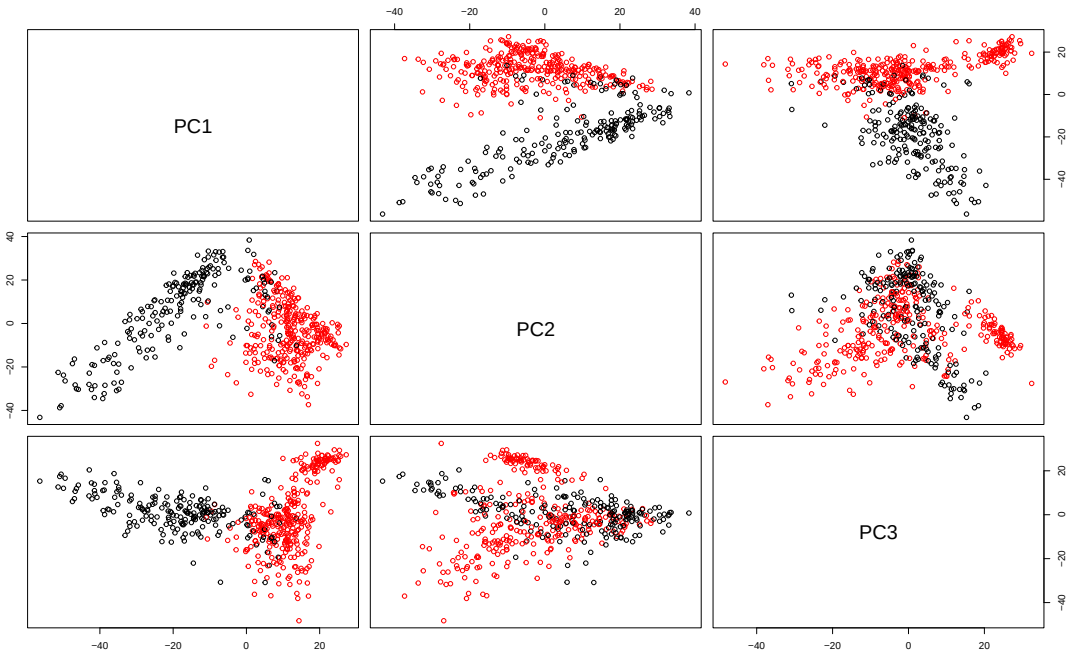


Figure 2: Pairs plot of the first 3 principal components coloured by tumour and non-tumour cells.

The low variability explained by each principal component is typical of this kind of data as there is a large amount of variability in gene expression data that will not all be captured by a few principal components. However, the most obvious sources of variation are still often captured by PCA, which is why it is still used in the field as a dimension reduction technique, just often with a larger number of principal components. This large number of principal components is still a considerably reduced dataset compared to the dataset before dimension reduction (which typically contains hundreds of thousands of genes). The scree plot in Figure 3 shows that the proportion of variance explained does not reduce much with each additional principal component (indicated by the values on the y-axis). However, the plot levels out after the red line at 30 principal components. Thus, these 30 principal components will be used as the reduced dataset for the rest of the analysis and cumulatively explain roughly 10% of the variability in the data.

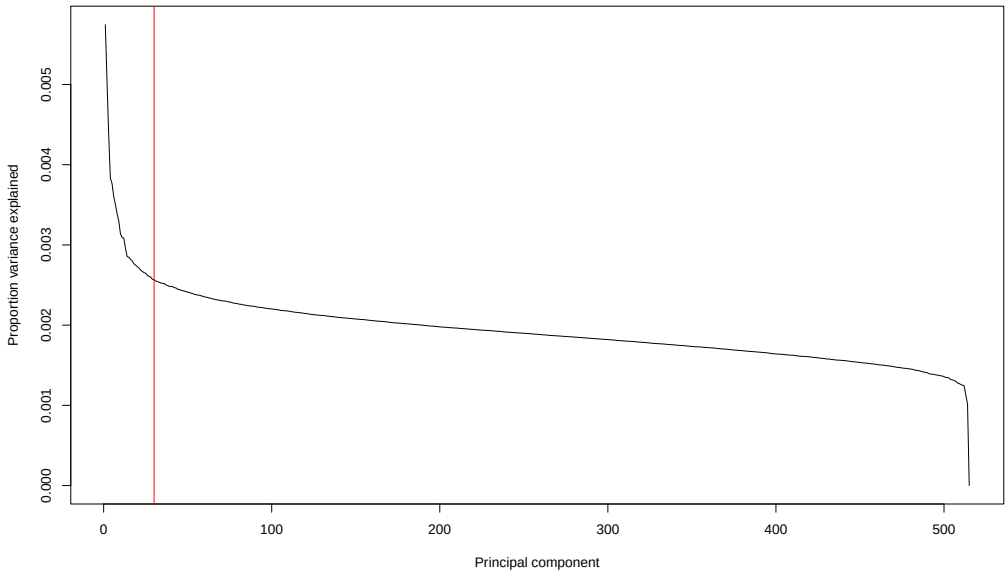


Figure 3: Scree plot for the PCA.

3.2.2 t-Distributed Stochastic Neighbour Embedding (tSNE)

tSNE is a non-linear dimension reduction technique that is often used to reduce the dimensions of single-cell transcriptomic data. tSNE is an adaptation of Stochastic Neighbour Embedding that keeps data points that are similar in higher dimensional representation similar in lower dimensional representation. This allows tSNE to capture local and global structure and better identify clusters of similar observations in the data. The cost function for SNE is altered in tSNE to a symmetric version (by using joint probabilities instead of conditional ones to represent similarities) with a Student t distribution in the lower dimensional space rather than a Gaussian Distribution. This symmetric cost function is given by:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log(\frac{p_{ij}}{q_{ij}}) \tag{2}$$

Where, p_{ij} is the pairwise similarity between observation i and j in the high dimensional space. This similarity is generated by the symmetrized conditional probabilities of x_i and x_j ($p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$). q_{ij} is the pairwise similarity between observation i and j in the low dimensional space and is generated by the joint probability of x_i and x_j but in the lower dimensional space. In the higher dimensional space, distances are converted to joint probabilities using a Gaussian distribution and in the lower dimensional a Student t distribution is used.

tSNE was applied to the cancer dataset to attempt to identify lower dimensional representations of the data that allow the identification of groupings in the data. This was done using the *Rtsne* package in R where multiple perplexity values were compared to generate a good lower dimensional representation. Perplexity values between 30 and 100 worked best and all produced very similar results so the default perplexity of 30 was chosen. The results of this analysis can be seen in Figures 4 and 5. When colouring the data by tumour and non-tumour cells we can see that there is a seperation, however, without the class labels this would be difficult to identify as there are many more groups in the data than 2. When colouring the same results by cancer type/sample it seems that some of the observations are clustering with observations from the same samples, especially samples BC01, BC02 and BC05. However, samples from non-tumour cells across all samples group together on the right of the plot.

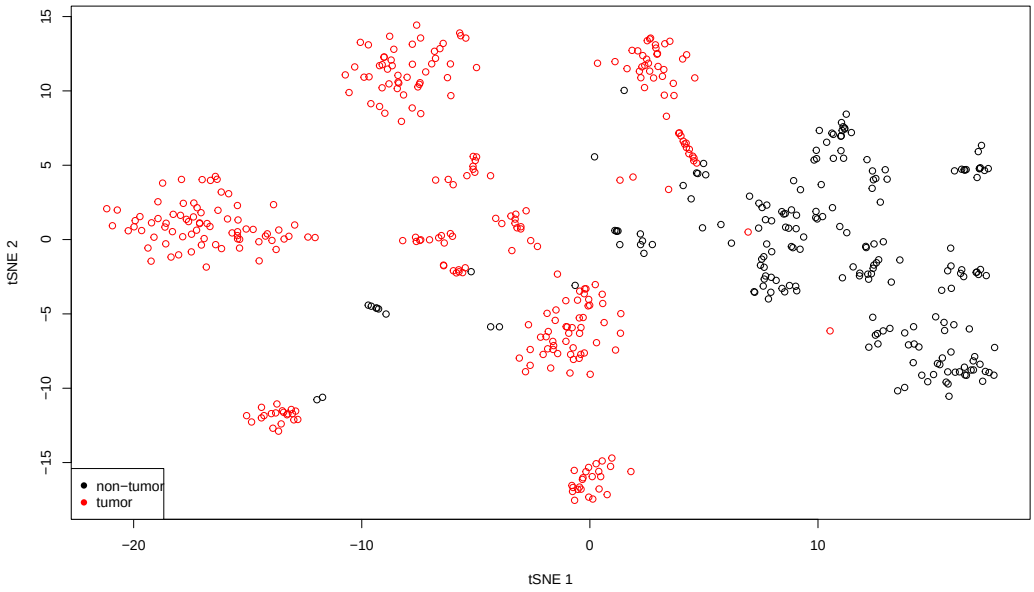


Figure 4: tSNE results with data points coloured by tumour and non-tumour cells.

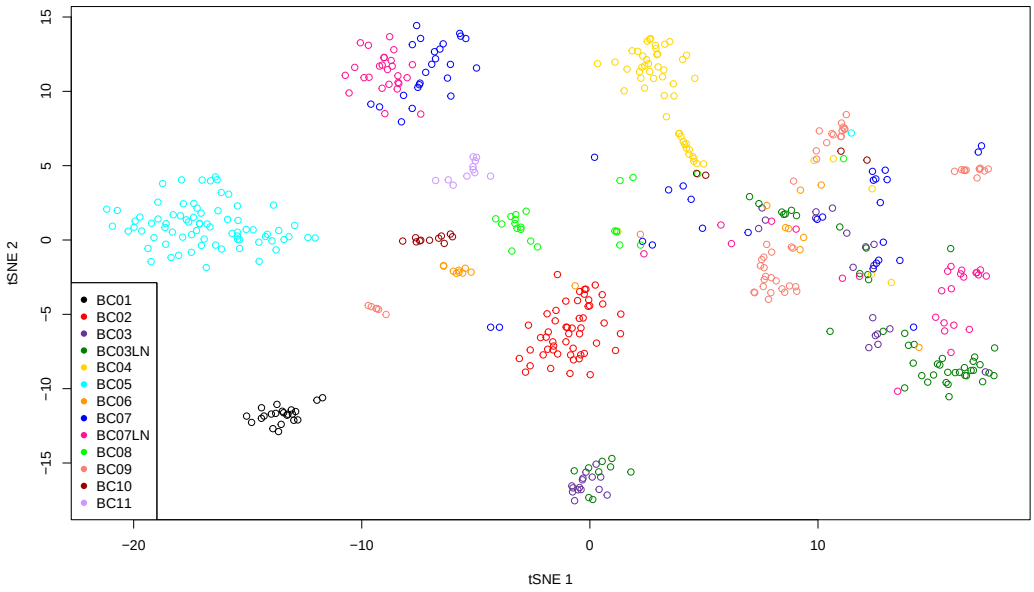


Figure 5: tSNE results with data points coloured by cancer type/sample.

3.2.3 Kernel PCA

Kernel PCA is another non-linear dimension reduction technique. It is not commonly used in the field like tSNE but is demonstrated here as it is still a technique that is applicable to reducing the dimensions of non-linear data (which single-cell transcriptomic data is). It is based on the fact that separations that are

non-linear in a lower dimensional space can appear as linear in a higher dimensional space. The groupings are theoretically done in the higher dimensional space and then mapped down to a lower dimensional space. However, to avoid actually having to work in the higher dimensional space, a kernel function is used. To compare dot products in the higher dimensional space, a kernel function takes as its input, vectors in the original space and returns the dot product of the vectors in the higher dimensional space. The kernel function is given by:

$$k(x, y) =< \phi(X)\phi(Z) > \tag{3}$$

Where, X is the data in the original space, $Z \in X$ and ϕ is the map such that $\phi : X \to R^N$.

Kernel PCA was applied to the cancer dataset to see if better groupings could be identified in a non-linear framework. This was done using various kernel function (Gaussian Radial Basis Function kernel, Polynomial kernel, ANOVA kernel and Laplace kernel) all with a variety of parameter values. The best performing kernel was the Gaussian Radial Basis Function kernel with $\sigma = 0.0001$. However, the separation of tumour and non-tumour cells seems to be better with just a standard PCA rather than the kernel PCA. This can be seen in Figure 6, there is more overlap of data points in this plot and the two groups are less separate from one another than in the PCA.

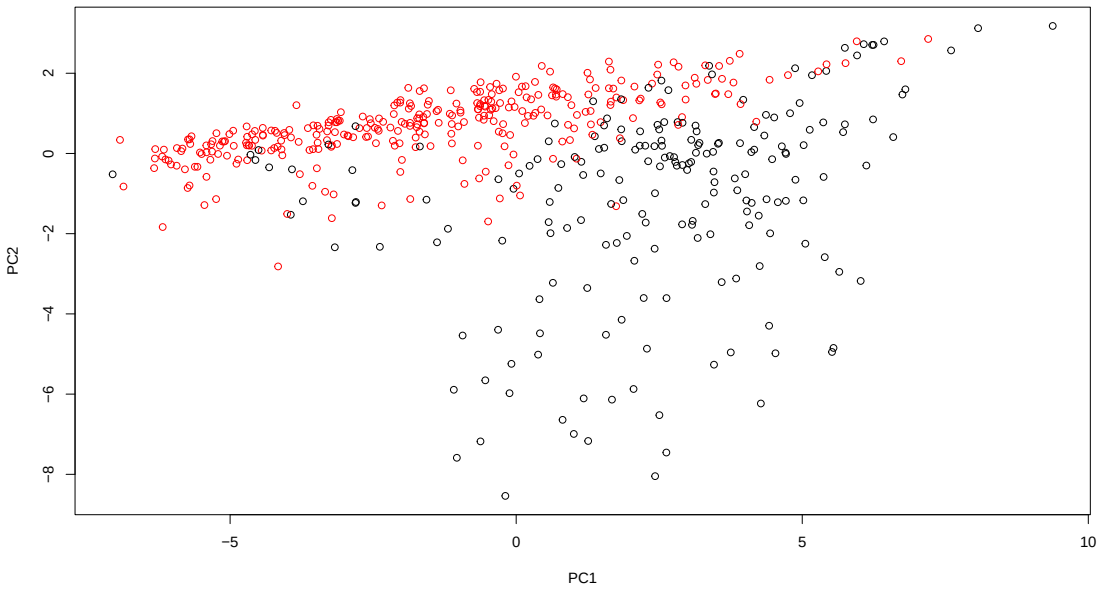


Figure 6: First 2 principal components from kernel PCA.

3.2.4 K-means clustering

The k-means clustering algorithm is most appropriate for strictly numerical data since it typically partitions the data into k clusters based on the Euclidean distance between observations (cells). The algorithm is initiated by stochastically assigning each observation to one of k clusters. The mean-profile for each cluster (called a cluster centroid) is computed, and the Euclidean distance from each observation to each centroid is found. Each observation is then assigned to the closest cluster centroid, and the computations are recalculated. This reallocation repeats iteratively until the objective function, the total within-cluster sum of squares, is minimized (or a specified number of iterations is reached). The random initial cluster assignment

results in non-unique clustering solutions that do not guarantee a global minimum. Usually, one thus specifies a number of random initialisations over which the algorithm is performed and the final clustering which has the lowest objective function value is chosen. In the analyses that follow, each k-means implementation is run with 10 random starts.

The number of clusters, k, needs to be pre-specified and is usually determined by comparing metrics/visualisations such as the total within-cluster sum of squares (WSS), GAP statistics and silhouette plots across a range of k. The GAP statistic is a goodness-of-fit measure for clustering, where larger values signify a better fit. It compares the observed average within-cluster dissimilarity, $W(k)$, with the expected dissimilarity under a null hypothesis of uniformity:

$$GAP_n(k) = E_n[log(W(k))] - log(W(k))$$

where the expectation is defined by bootstrapping $n = 5$ samples from a uniform distribution.

The WSS and GAP statistics were computed for $k = 2 : 8$ (from k-means performed over the first 300 principle components) and are illustrated in Appendix B. A large number of principle components is chosen to retain a reasonable proportion of the variation in the original feature space (see Appendix A). The WSS statistic shows the most significant reduction at $k = 2$, while the GAP statistic exhibits a significant peak at $k = 2$ and $k = 5$. These results are certainly not conclusive, however, and thus the number of clusters chosen is informed from the data (Figure 1). The algorithm is thus first performed with $k = 2$, in the attempt to distinguish the tumour and non-tumour cells.

While clustering is typically performed on the principle component scores to combat the high dimensionality, the algorithm can be applied directly to the transformed expression matrix. Figure 7 shows the (scaled) mean expression profiles over the 35000 genes across the cells, coloured by k-means cluster. The plot shows significantly less variability among the tumour cluster (red) across most of the genes when compared to the cluster mostly comprised of non-tumour cells (black). There are a selection of genes that are on average expressed more extremely among the cells in the tumour cluster. Given the large number of genes, it is hard to interpret these results visually. This could, however, be input to a differential gene expression analysis, which could help to reveal the genes which are relatively unique to each cluster.

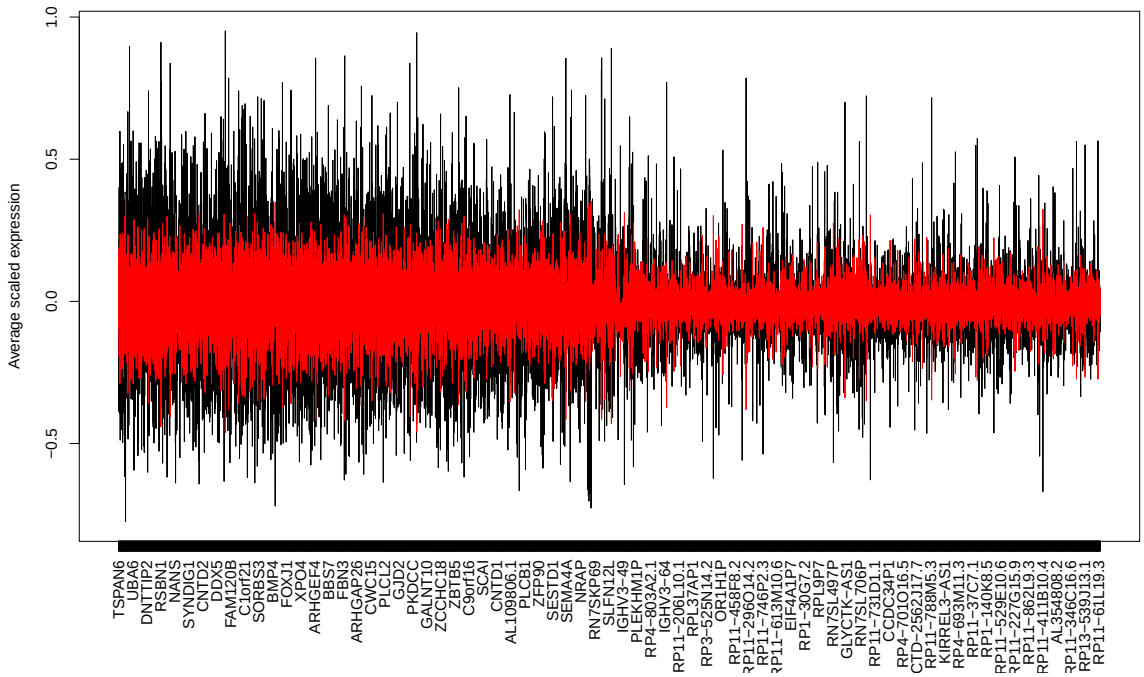


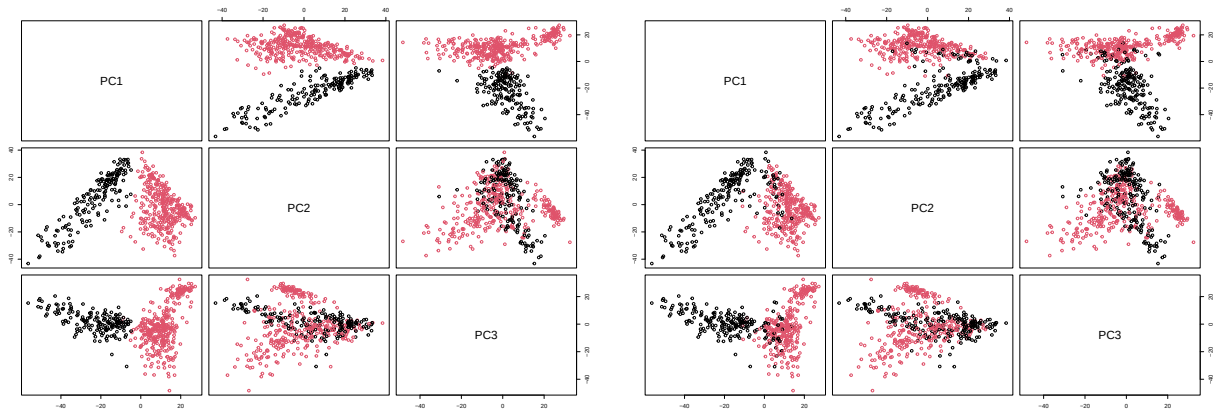
Figure 7: Mean profile over all 35000 genes in the analysis for each k-means cluster (using the full dimensionality). The cluster in red corresponds to tumour cells and black to non-tumour cells.

Next the k-means algorithm was applied with $k = 2$ to the first 30 PCA scores generated in section 3.1. Since the true cell types are present in this data set, the k-means clustering can be compared to the true labels. A confusion matrix is thus presented in Table 2. The k-means clustering correctly groups 93.6% of the cells, with 2 tumour cells and 31 non-tumour cells misclassified.

Table 2: Confusion table: k-means on PCA scores

	True label	
Prediction	Tumour	Non-tumour
Tumour	315	31
Non-tumour	2	167

The clustering results can be visualised in two dimensions by generating pairwise scatterplots of the first three PC scores and colouring the cells by cluster. Since we have the true labels in this example, this can be compared to the true cell types. These plots are shown in Figure 8, with the true labels on the right. Visually, it is clear that the k-means algorithm generated clusters which closely mirror the true labels. The 31 non-tumour cells (black) which the k-means algorithm classified as tumour cells can be seen to group with the tumour cells (red) along the first principle component (Figure 8b). The misclassification is thus attributable to those non-tumour cells having scores for the first principle component that are on average more similar to the tumour cells than to the average non-tumour cells. This result implies that the clustering is highly dependant on the results of the dimension reduction method. Since the clustering algorithm groups cells based on their similarity in the lower dimensional space, if the heterogeneity in this space does not reflect the true cell types then the algorithm will not be able to distinguish them.



(a) Coloured by k-means clusters

(b) Coloured by true labels

Figure 8: K-means on PC scores (red = tumour cell)

As with the clustering performed on the expression matrix, the differentiation between the clusters can be investigated by looking at the mean profiles but this time across the principle components. This is illustrated in Figure 9, showing that the cells in the two clusters on average differ markedly in their scores along the first 9 principle components. If combined with the clinical expertise necessary to interpret the loadings on the principle components, this comparison could form the basis of identifying/annotating the cell types represented by the clusters.

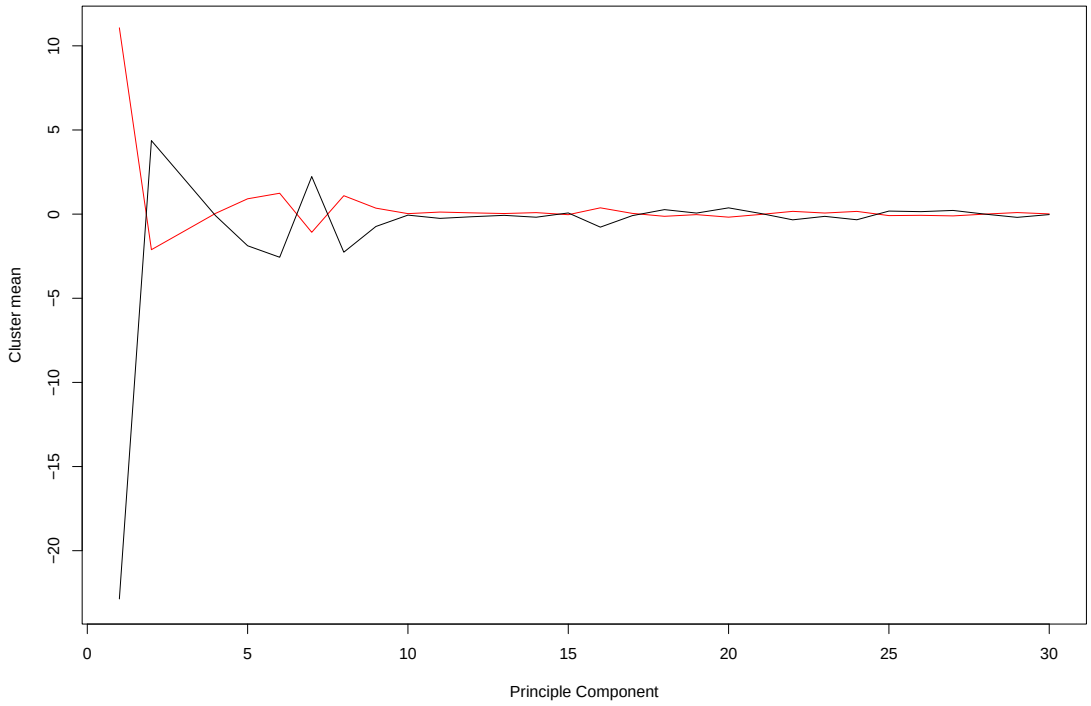


Figure 9: Mean profiles over first 30 principle components. (red=tumour cell)

3.2.5 Hierarchical clustering

The hierarchical agglomerative clustering algorithm is initiated by considering each observation (cell) as a separate cluster. The algorithm then iteratively merges the closest clusters until all points are combined into a single cluster. The definition of the distances between clusters - and hence the resulting clusters - depends on the type of linkage used. The most common linkages are single, complete, average and Ward's linkage. A number of linkages were applied with this data and Ward's was found to produce much more sensible clustering than the others.

Ward's linkage is a type of agglomerative clustering algorithm that minimizes the variance of the distances between points in a cluster. In Ward's linkage, the distance between two clusters is defined as the increase in the total error sum of squares that results from merging two clusters. This linkage criterion tends to produce clusters of similar sizes and compact shapes, and it is less sensitive to noise and outliers than single linkage or complete linkage.

The algorithm was thus applied to the expression matrix using Ward's method. The resultant clustering is visualised by the dendrogram in Figure 10. The algorithm distinguishes clusters at various levels. Visually, the distinction between clusters can be seen by the height between branches in the tree. The greatest distinction seems is at the first level and the clusters become more similar as more partitions are made. As with k-means, metrics such as the WSS and silhouette coefficient could be used to help determine the number of clusters. The average silhouette width was informative in this case, showing the statistic is minimized at $k = 2$ (Appendix B: Figure 20).

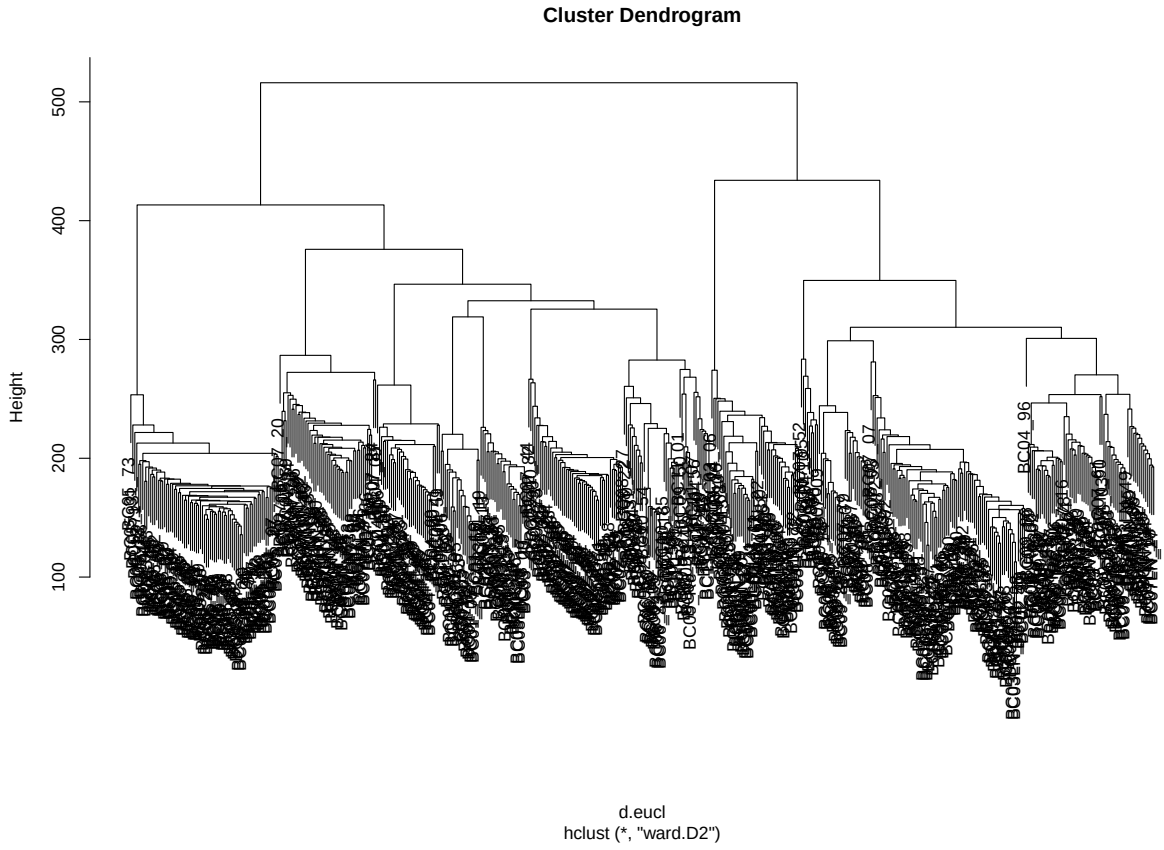


Figure 10: Dendrogram: hierarchical clustering with Ward's linkage

The clustering was thus cut at the first level to produce two clusters. Table 3 shows that the algorithm

correctly classified 283 tumour and 189 non-tumour cells correctly for an overall accuracy of 91.6%. The sensitivity was thus 10.1 percentage points lower compared to k-means, but the specificity was 11.6 points higher.

Table 3: Confusion table: hierarchical clustering with Ward's linkage

	True label	
Prediction	Tumour	Non-tumour
Tumour	283	9
Non-tumour	34	189

As with the clusters obtained from the k-means algorithm, these clusters can be compared by investigating the differences between their mean profiles. Figure 11 illustrates this along the first 30 principle components. The plot shows that the average principle component scores differ between the two clusters for the first 12 principle components or so, after which the average scores tend towards zero. These results are similar to those in Figure 9, albeit not the same. This is to be expected since the clusters are not identical in terms of the cells that comprise them. Again, the loadings of the principle components which differ most here could be investigated to identify the genes that are clinically meaningful in separating these cell types.

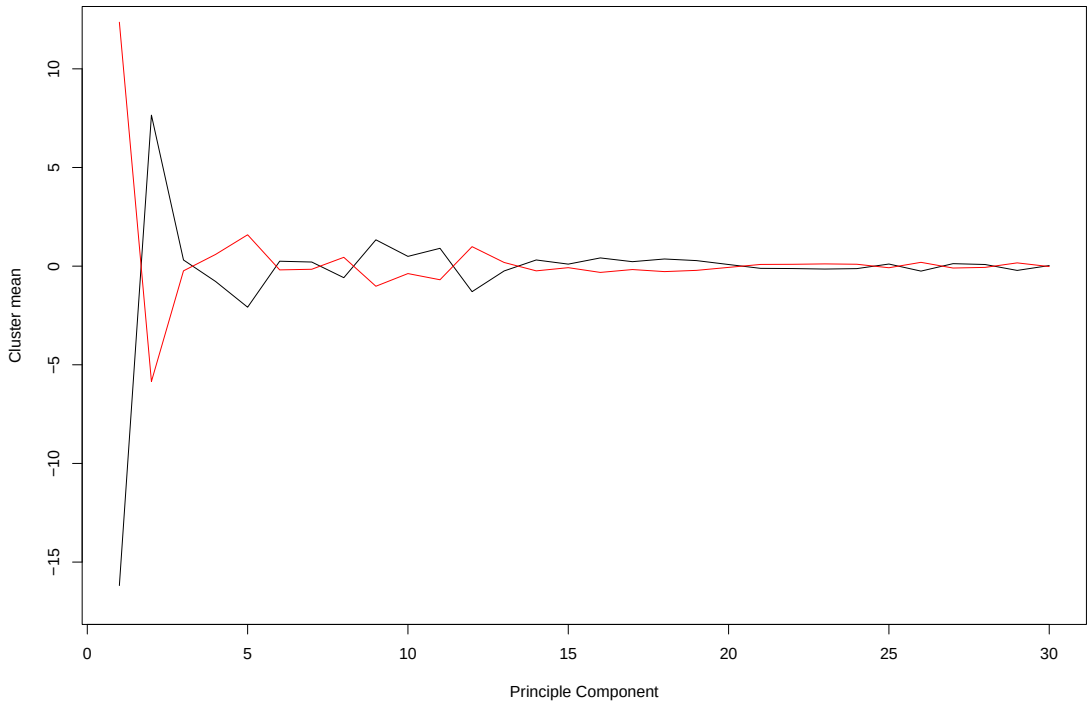


Figure 11: Mean profiles over first 30 principle components. (red=tumour cell)

3.2.6 Graph-based clustering

Seurat's Graph-based clustering is based on the methods outlined by Levine et al. (2015) [19]. It involves first generating a K-nearest neighbours (kNN) plot of the data using the euclidean distance based on PCA results. From this graph, similarity between cells is represented by their shared neighbours. The more shared neighbours, the more similar two cells are. The number of shared neighbours between two cells is

then converted to a weight between the two nodes (where each node on the graph represents a cell). The Louvain algorithm is then used to generate clusters. This algorithm optimises (maximises) the modularity (Q) of a graph partition which is essentially a measure of the quality of a partition and is given by:

$$Q = \frac{1}{2m}[\sum_{i,j} W_{ij} - \frac{S_i S_j}{2m}]\delta(c_i, c_j) \tag{4}$$

Where W_{ij} is the weight between nodes i and j (generated by the kNN graph calculations), S is the sum of all weights relating to a specific node, c is the community assignment of a node (cluster assignment) and $\delta(c_i, c_j) = 1$ if $c_i = c_j$ and 0 otherwise. m is just a normalisation constant.

This algorithm was applied to the data and the results can be seen in Figures 12 and 13. In Figure 12 the first two principal components have been plotted and these results are coloured by the clusters generated by *Seurat*. The tSNE plot in Figure 13 has also been coloured by the same clusters. These clusters group much better on the tSNE plot than in the PCA plot. This is probably because the clusters seem to group cancer types/ samples together and this is also what was observed with the tSNE results. There are however, only 9 clusters where there are 11 samples so the clustering is grouping some samples together which seem to have similar expression patterns. When having a closer look at the clusters, it can be seen that the algorithm also splits some cells that came from the same sample into different clusters. It does capture most of the samples in the data relatively well though.

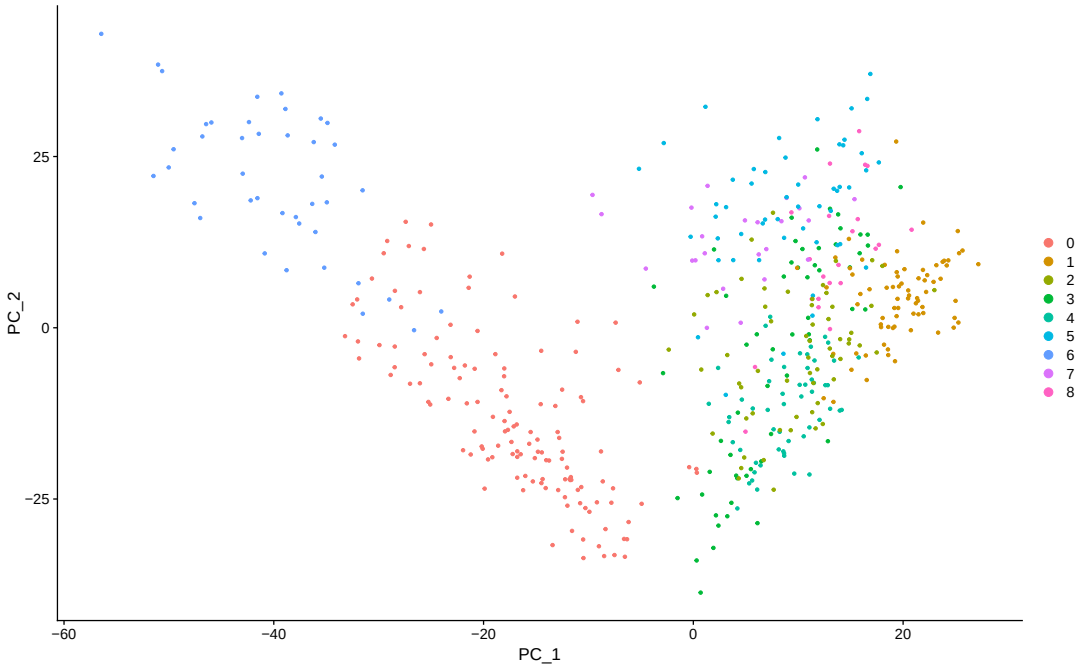


Figure 12: First 2 principal components of the PCA coloured by the graph-based clusters.

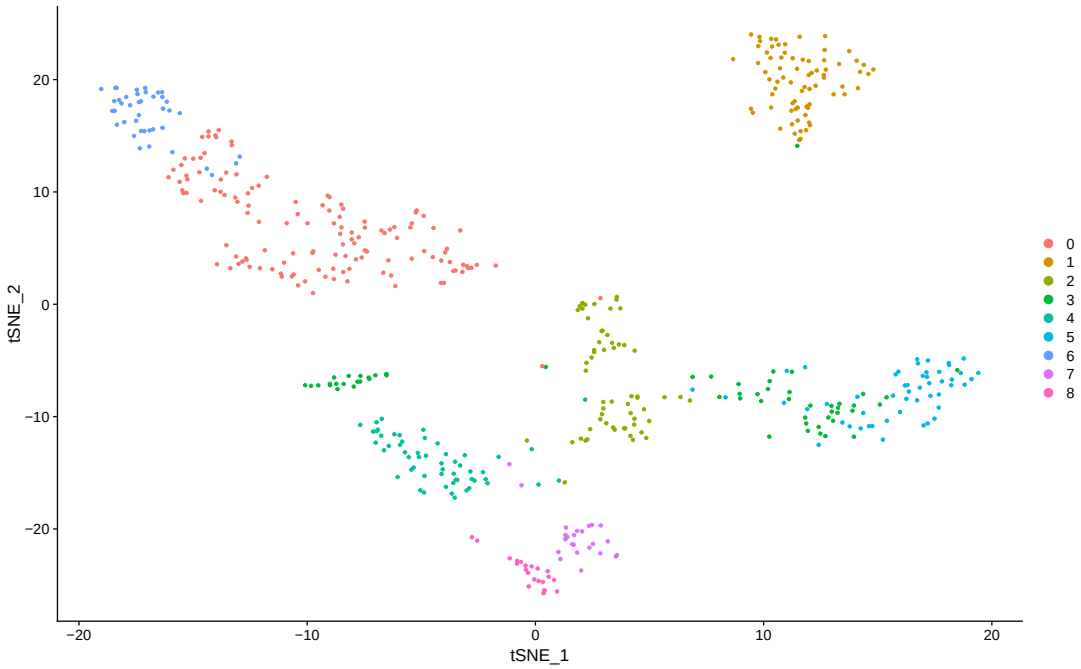


Figure 13: tSNE results coloured by the graph-based clusters.

3.3 Supervised techniques

Supervised machine-learning techniques are used in the field of single-cell transcriptomics. Many machine-learning models in the field of single-cell transcriptomics come pre-trained and are not always trained on data for the specific cell-types in a dataset. Some machine-learning models that require training based on a test dataset are presented below. These can be trained on any single-cell transcriptomic dataset and then used to predict on data containing the same cell-types. These machine-learning methods can be combined with a dimension reduction technique to make the data more manageable for the algorithms.

3.3.1 Support Vector Machines

Support Vector Machine (SVM) is a machine learning algorithm that is primarily used for data classification. It involves identifying a hyperplane that can separate the data into binary classes. For non-linear data Kernelized SVM is used to separate the data. In the same manner as with kernel PCA, the idea is to map the data to a higher dimension where the data is linearly separable and the SVM algorithm is able to separate the data. The kernel function is used to estimate the similarity between data points in the higher dimensional space and the SVM algorithm then identifies the hyperplane based on these similarities, without actually having to transform the data to the higher dimensional space.

The optimisation in a SVM is given by:

$$max_\alpha\ W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j K(x^{(i)}, x^{(j)}) \tag{5}$$

$$\text{s.t } \alpha_i \geq 0 \ i = 1, ..., m$$

$$\textstyle\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Where $K(x^{(i)}, x^{(j)})$ is the kernel function.

To implement classification via a support vector machine, the *kernlab* R package was used. The same kernel that was used for the kPCA was used here and the 'C'-constant of the regularization term in the Lagrange formulation was set to 1 (the default). Increasing C can improve accuracy on the training set but often leads to over-fitting. Higher values of C were also explored but the performance remained mostly the same. The data was split into training and test sets (with a 70:30 split) and the model was trained on the train set. The model was then used to predict classifications on the test set and the results of these predictions are shown in Table 4. The misclassification rate was roughly 6% which means that the model classifies the majority of the data points correctly. The false-positive rate is also higher than the false-negative rate. This means that when cells are tumour cells, they are likely to be identified correctly. However, when the cells are non-tumour cells, there is a small chance that they will be classified as tumour cells.

This dataset is not very large (only contains 515 cells) and the model still seems to classify the cells relatively well. With a larger dataset to train the model on, this approach could be very promising for identifying tumour cells in samples from genetic profiles of cells.

Table 4: Confusion table: SVM on test data.

	True label	
Prediction	Tumour	Non-tumour
Tumour	81	8
Non-tumour	1	64

3.3.2 Ensemble Algorithms

This section considers the application of ensemble machine learning algorithms with the aim of predicting the categories of non-tumour cells. Thus, for this analysis, the tumour cells were excluded from the data. In particular, it is shown how random forests and gradient boosting can be trained to partition the principle component feature space to separate the sub-types of non-tumour cells. These algorithms are trained on a subset of labeled data and used to predict cell-types on unseen data. Each algorithm was trained on a random subset of 60% of the non-tumour cells, and the final method is tested on the remaining 40% of cells.

The random forest algorithm builds a series of decision trees by using a bootstrap sample of the training data. Each decision tree is constructed by selecting a random subset of features, $mtry < f$, at each split. The final prediction is obtained by averaging the predictions of all trees in the forest. The parameterisation of random forests involves assessing the out-of-bag (OOB) error rate, which compares the classification to that of the unsampled observations. In contrast, gradient boosting algorithm builds an ensemble of decision trees by sequentially adding trees with slow learning rates and a relatively small number of partitions, with each new tree attempting to model the residuals from the previous tree. The process involves optimizing a loss function by maximising the accuracy between the predicted and true classes.

Both of these algorithms require the tuning of certain hyperparameters. The *randomForest* package was used for the random forest, and a grid search was thus performed to determine the optimal number of trees (B) and number of features for each split ($mtry$). The parameterisation which maximised the out-of-bag classification accuracy was $B = 91$ and $mtry = 92$ (OOB error = 18.3%) (see Appendix C: Figure 21).

The Extreme Gradient Boosting algorithm was implemented next via the *caret* package [6]. There are three tuning parameters to consider: the number of trees (B), the learning rate (η) and the interaction depth (d). One way in which gradient boosted trees differ from random forests is that a large B can lead to overfitting. Cross validation is thus used ($k = 5$) on the training data to determine the set of tuning parameters which maximises the CV accuracy. A grid search was performed, and the optimal parameterization was determined to be $B = 50$, $\eta = 0.01$, $d = 4$ (CV error = 24.1%). This can be seen in the top panel of Figure 22 in Appendix C.

The random forest and gradient boosting algorithms with their chosen parameterisations were used to generate predictions on the validation set. The random forest's predictions yielded the greatest accuracy, 84.6%, and the associated confusion matrix is shown in Table 5.

Table 5: Confusion table: Random forest on validation set

	True label			
Prediction	Bcell	Myeloid	Stromal	Tcell
Bcell	30	0	0	3
Myeloid	0	14	1	3
Stromal	0	0	7	1
Tcell	3	1	1	15

By evaluating the relative change in the objective function at each partition, the importance of each principle component in separating the cell types can be revealed. The top six variables in Figure 14 show that a selection of 6 principle components had a variable importance greater than 15%. To identify the clinical relevance of different genes in distinguishing these cell types, the loadings on these principle components could be examined.

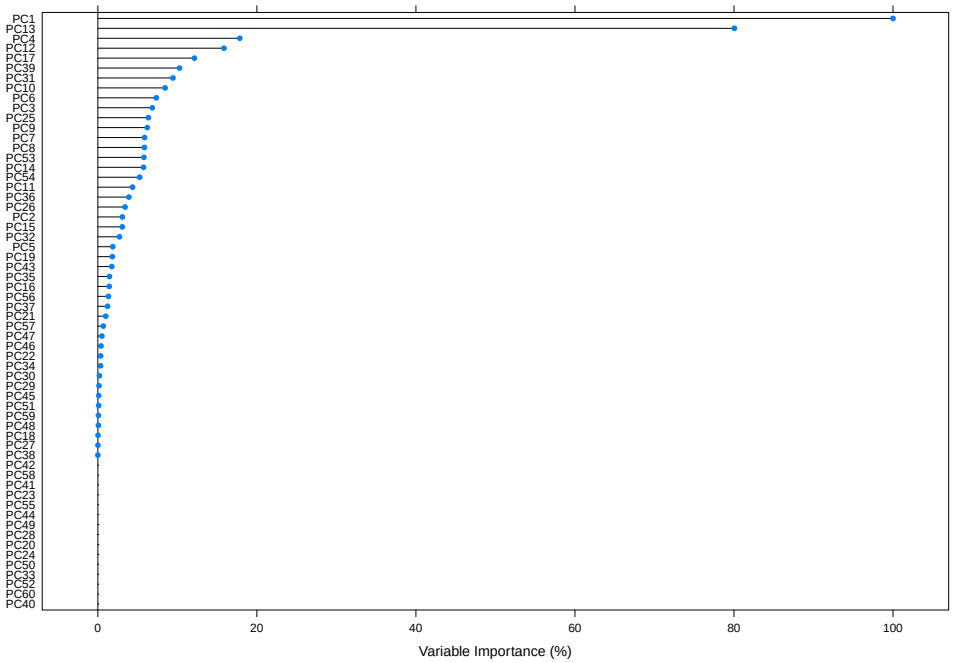


Figure 14: Variable Importance Plot: Random Forest

4 Discussion

In the above sections, the general approach to analysing single cell transcriptomics data was demonstrated. First, the motivations and methods behind the conventional data pre-processing were outlined. It was shown that ideally a raw counts matrix needs to be normalized and scaled, and some feature selection can be performed to remove uninformative genes.

The curse of dimensionality was shown to be a very relevant problem in this field. To reduce the feature space, three unsupervised dimension reduction techniques were demonstrated - each generating different results. Ordinary PCA was shown to produce an orthogonal reduced feature space which can effectively distinguish tumour from non-tumour cells, whilst retaining as much of the variability in the original feature space as possible. tSNE was shown to be an effective visualisation tool, while inaccurate in grouping cells by

tumour/non-tumour status. The potential for Kernal PCA as a non-linear dimension reduction technique was demonstrated, however the reduced feature space did not distinguish tumour from non-tumour cells as accurately as ordinary PCA.

Next, the application of three unsupervised clustering methods were demonstrated to generate groups of cells with similar gene expression profiles to different degrees. K-means was first demonstrated on both the original and reduced feature space, in an attempt to distinguish tumour from non-tumour cells. This is not conventionally done, and thus clustering on the principle component scores was performed. By comparing the clusters to the true labels, it was seen that the algorithm correctly clustered 93.6% of cells in the binary groupings. It was shown that the misclassification occurred as a result of these cells being more similar to the group to which they did not belong (in terms of principle component scores). This demonstrated the sensitivity of the clustering algorithm to the dimension reduction technique. Next, the application of hierarchical agglomerative algorithms was explored. The algorithm generated sensible clusters, with an overall accuracy of 91.6%. Through k-means and hierarchical clustering, it was shown that the cell types could be identified by exploring the mean profiles across their principle component scores, however this does require some clinical expertise. Lastly, *Seurat's* graph-based clustering algorithm was explored. This was shown to generate clusters that closely mirror the different samples from which the cells were taken.

While the true labels were present in this data, the challenges in annotating the cell clusters resulting from the unsupervised clustering algorithms were demonstrated. In the final section, it was thus shown how supervised algorithms could be used to classify cell types if trained on a data set of similar information. Support vector machines were able to correctly classify 94.2% of cells as tumour/non-tumour. The application of random forests was also explored, in classifying non-tumour cells. The algorithm was shown to correctly predict the cell-type of 84.6% of non-tumour cells. It was also shown how variable importance can be used to show those principle components, and thus genes, which are most important in distinguishing these cell types. It must be noted that the classification performance of both of these algorithms could likely be improved if trained on data with more cell types. At the same time, the out-of-sample performance of these algorithms depends on the similarity between the training data and the unseen data. This may present some challenges when there are between-batch effects in different samples, such as differences in sequencing depth.

It must be acknowledged here that, for the sake of brevity, there are many algorithms - dimension-reduction and (un)supervised clustering - which have not been explored in this text but may be useful in this analysis. For example, Uniform Manifold Approximation and Projection (UMAP) is an often-used alternative dimension reduction technique which has not been discussed here. While these algorithms may generate different results and have slightly different use-cases, their application here would all be very similar and would fit into the framework which has been outlined. Lastly, some techniques used in the annotation of unsupervised cell clusters was not explored in depth. For example, reference-based annotation was described but not demonstrated. This is just one example of downstream applications of the methods described here which could be the topic of another report of similar length.

5 Conclusion

This report demonstrates the use of multivariate analysis techniques to analyse single-cell transcriptomic datasets with the goal of identifying cell types/groupings. Many different statistical techniques are demonstrated and there are many more that are not demonstrated in this report. The choice on which techniques to use will ultimately depend on the data and it is recommended that a variety of techniques are used to ensure consistency of the results. This is because with datasets this large (and with this much inherent variability), the results can differ for different methods of analysis. Most of the methods demonstrated in this report succeeded in identifying the tumour and non-tumour cells from the single-cell expression data and the random forest was also able to identify the specific types of non-tumour cells relatively accurately. Hopefully, this report has highlighted the usefulness and importance of multivariate analysis in the field of single-cell transcriptomics.

References

[1] T. Abdelaal et al. "A comparison of automatic cell identification methods for single-cell RNA sequencing data". In: *Genome Biology* 20.1 (2019), p. 194. DOI: 10.1186/s13059-019-1795-z.

[2] T.S. Andrews and M. Hemberg. "Identifying cell populations with scRNASeq". In: *Molecular Aspects of Medicine* 59 (2018), pp. 114–122. ISSN: 0098-2997. DOI: https://doi.org/10.1016/j.mam.2017.07.002.. URL: https://www.sciencedirect.com/science/article/pii/S0098299717300493.

[3] T.S. Andrews, M. Hemberg, and V.Y. Kiselev. "Challenges in unsupervised clustering of single-cell RNA-seq data." In: *Nature Reviews Genetics* 20 (2019), pp. 273–282. ISSN: 0098-2997. DOI: https://doi.org/10.1038/s41576-018-0088-9.

[4] H. Bengtsson. *matrixStats: Functions that Apply to Rows and Columns of Matrices (and to Vectors)*. R package version 0.62.0. 2022. URL: https://CRAN.R-project.org/package=matrixStats.

[5] V.D. Blondel et al. "Fast Unfolding of Communities in Large Networks." In: *Journal of Statistical Mechanics Theory and Experiment.* 10 (2008). DOI: 10.1088/1742-5468/2008/10/P10008.

[6] T. Chen and C. Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[7] W. Chung et al. "Single-cell RNA-seq enables comprehensive tumour and immune cell profiling in primary breast cancer". In: *Nat Commun* 8 (2017), p. 15081. DOI: 10.1038/ncomms15081. URL: https://www.nature.com/articles/ncomms15081/.

[8] Microsoft Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*. R package version 1.0.17. 2022. URL: https://CRAN.R-project.org/package=doParallel.

[9] M. Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[10] S. Fortunato and M. Barthélemy. "Resolution limit in community detection." In: *Proceedings of the National Academy of Sciences* 104.1 (2007), pp. 36–41. DOI: https://doi.org/10.1073/pnas.0605965104.

[11] S. Freytag et al. "Cluster headache: comparing clustering tools for 10X single cell sequencing data." In: *bioRxiv* (2017). DOI: 10.1101/203752. URL: https://www.biorxiv.org/content/10.1101/203752v1.

[12] S. Freytag et al. "Comparison of clustering tools in R for medium-sized 10x Genomics single-cell RNA-sequencing data." In: *F1000 Research* 7 (2018), pp. 1297–. DOI: https://doi.org/10.12688/f1000research.15809.1.

[13] Y. Hao et al. "Integrated analysis of multimodal single-cell data". In: *Cell* (2021). DOI: 10.1016/j.cell.2021.04.048. URL: https://doi.org/10.1016/j.cell.2021.04.048.

[14] C.R. John et al. "M3C: A Monte Carlo reference-based consensus clustering algorithm". In: *bioRxiv* (2018). DOI: https://doi.org/10.1101/377002.

[15] A. Karatzoglou, A. Smola, and K. Hornik. *kernlab: Kernel-Based Machine Learning Lab*. R package version 0.9-32. 2023. URL: https://CRAN.R-project.org/package=kernlab.

[16] A. Kassambara and F. Mundt. *factoextra: Extract and Visualize the Results of Multivariate Data Analyses*. R package version 1.0.7. 2020. URL: https://CRAN.R-project.org/package=factoextra.

[17] J.H. Krijthe. *Rtsne: T-Distributed Stochastic Neighbor Embedding using Barnes-Hut Implementation*. R package version 0.16. 2015. URL: https://github.com/jkrijthe/Rtsne.

[18] M. Kuhn. *caret: Classification and Regression Training*. R package version 6.0-93. 2022. URL: https://CRAN.R-project.org/package=caret.

[19] J.H. Levine et al. "Data-Driven Phenotypic Dissection of AML Reveals Progenitor-like Cells that Correlate with Prognosis". In: *Cell* 164.1 (2015), pp. 184–197. DOI: 10.1016/j.cell.2015.05.047.

[20] A. Liaw and M. Wiener. "Classification and Regression by randomForest". In: *R News* 2.3 (2002), pp. 18–22. URL: https://CRAN.R-project.org/doc/Rnews/.

[21] M. Maechler et al. *cluster: Cluster Analysis Basics and Extensions*. R package version 2.1.4 — For new features, see the 'Changelog' file (in the package source). 2022. URL: https://CRAN.R-project.org/package=cluster.

[22] A. Mahalanabis et al. "Clustering single cells: a review of approaches on high-and low-depth single-cell RNA-seq data." In: *Briefings in Functional Genomics* 18.6 (2019), pp. 434–434. DOI: https://doi.org/10.1093/bfgp/ely001.

[23] A. Mahalanabis et al. "Evaluation of single-cell RNA-seq clustering algorithms on cancer tumor datasets." In: *Computational and Structural Biotechnology Journal* 20 (2022), pp. 6375–6387. DOI: https://doi.org/10.1016/j.csbj.2022.10.029.

[24] R. Nayak and Y. Hasija. "A hitchhiker's guide to single-cell transcriptomics and data analysis pipelines". In: *Genomics* 113.2 (2021), pp. 606–619. ISSN: 0888-7543. DOI: https://doi.org/10.1016/j.ygeno.2021.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0888754321000331.

[25] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2019. URL: https://www.R-project.org/.

[26] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

[27] H. Wickham et al. *dplyr: A Grammar of Data Manipulation*. R package version 1.0.10. 2022. URL: https://CRAN.R-project.org/package=dplyr.

[28] P. Zhao et al. "Evaluation of single-cell classifiers for single-cell RNA sequencing data sets". In: *Briefings in Bioinformatics* 21.5 (2020), pp. 1581–1595. DOI: 10.1093/bib/bbz096.

6 Appendix

Appendix A: Dimension Reduction Plots

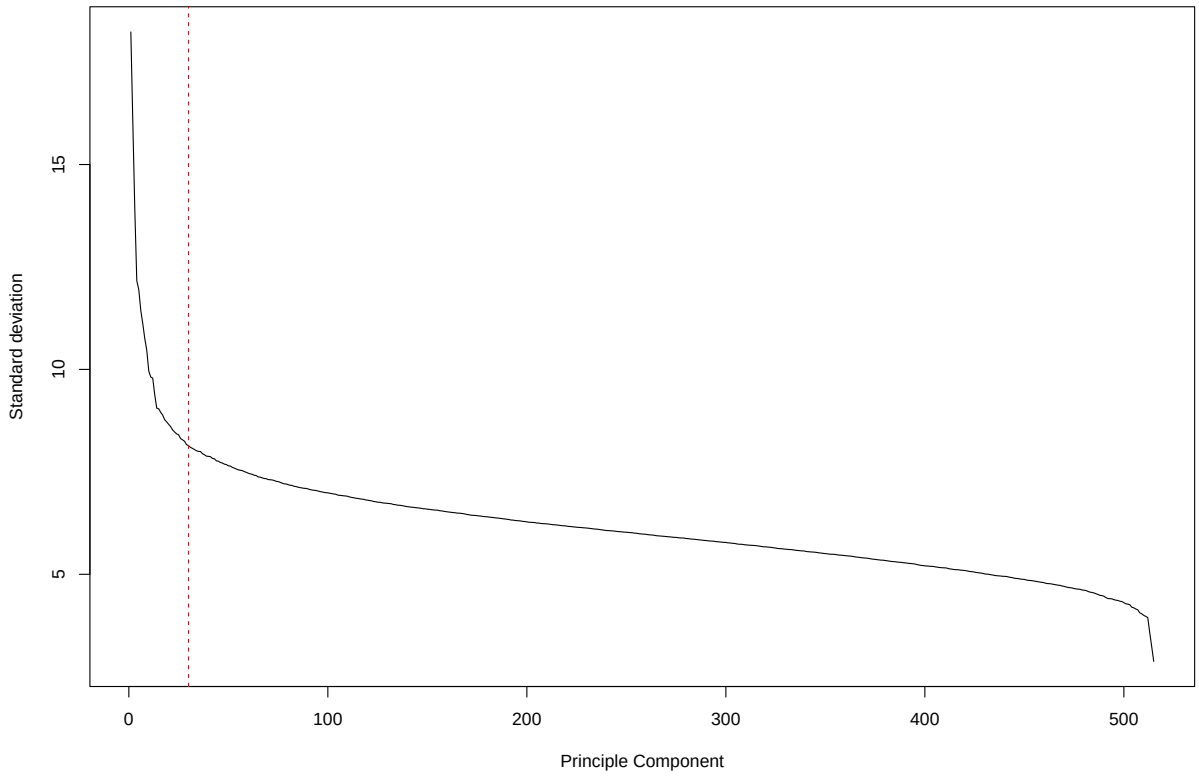


Figure 15: Standard deviation explained by each principle components

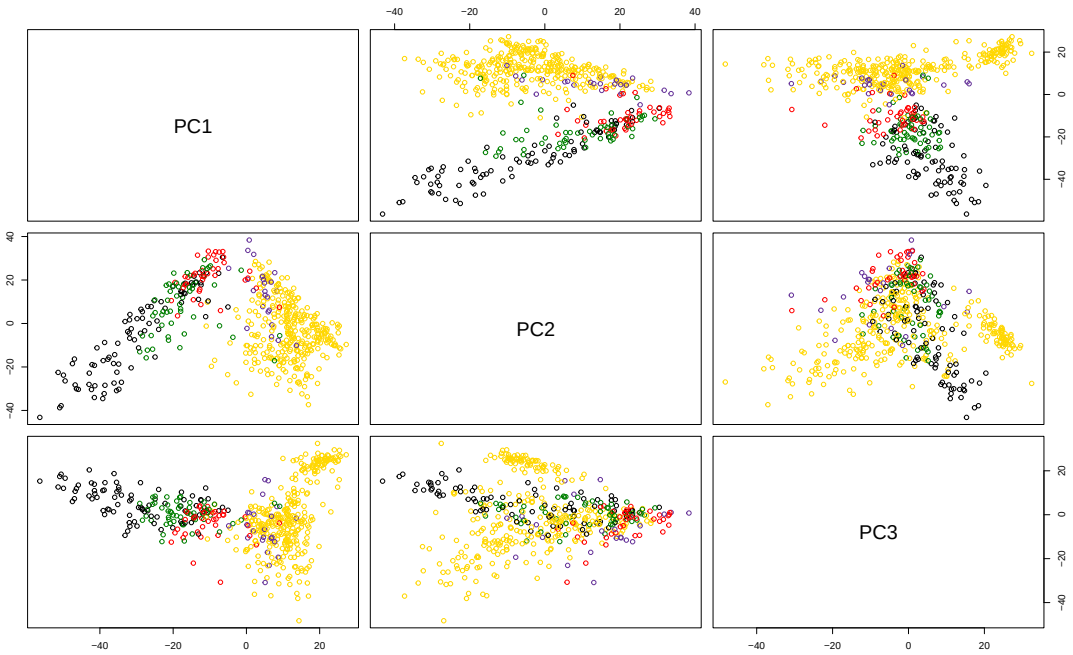


Figure 16: Pairs plot of the first 3 principal components coloured by the 5 different cell-types.

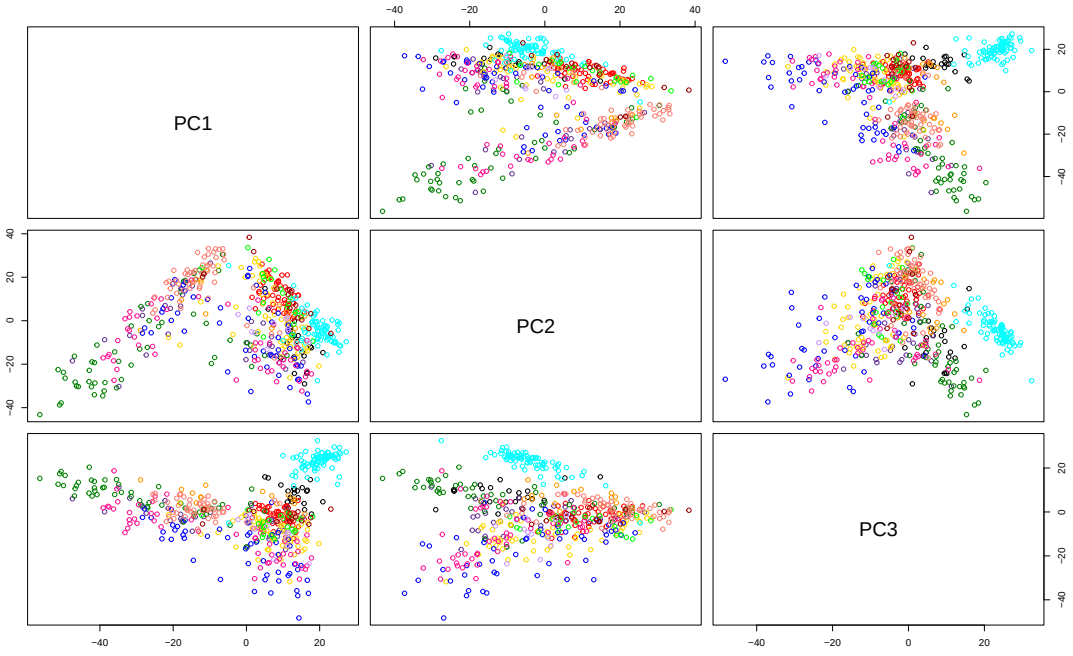


Figure 17: Pairs plot of the first 3 principal components coloured by cancer type/ sample.

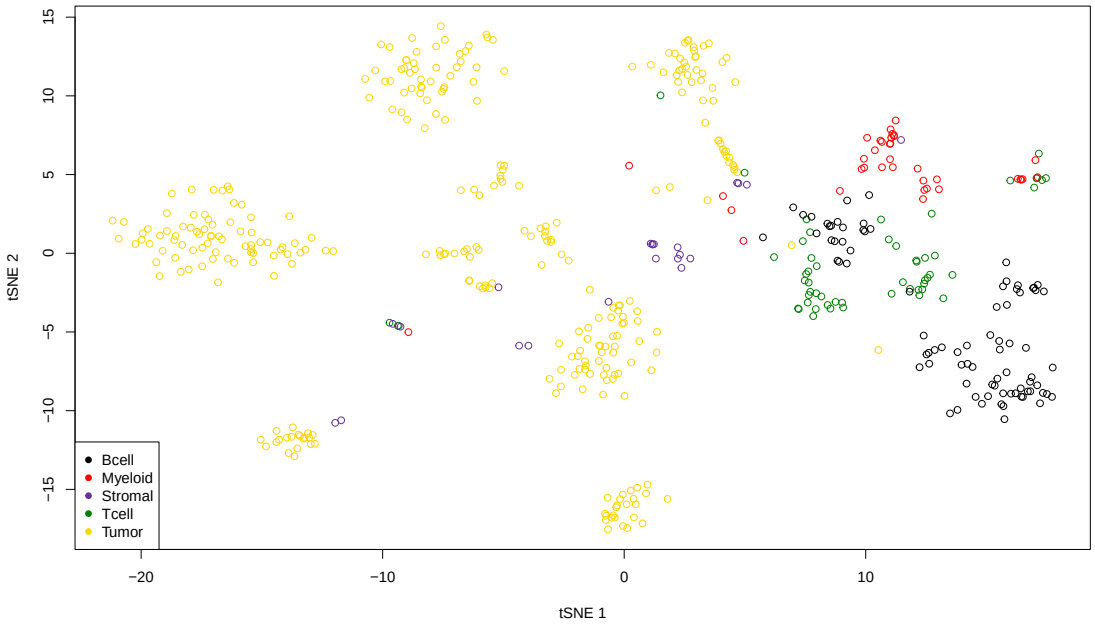
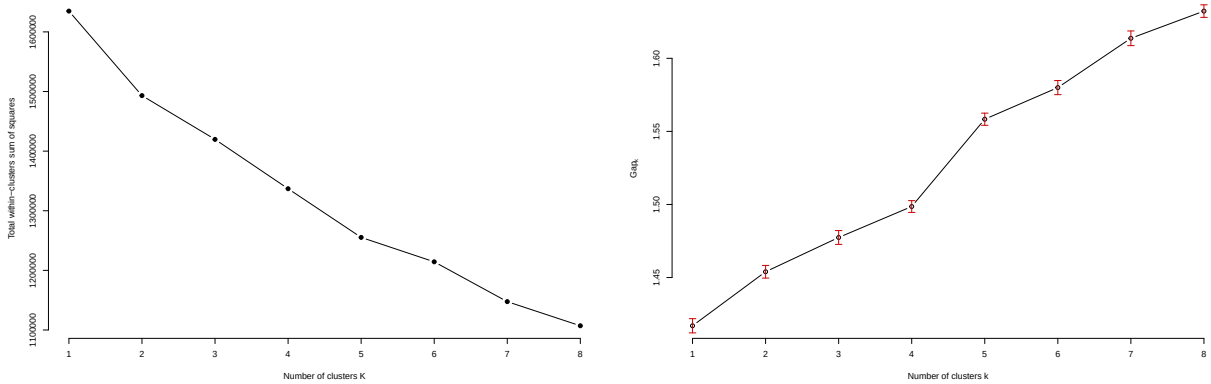


Figure 18: tSNE results with data points coloured by cell sub-types.

Appendix B: Determining K in unsupervised clustering



(a) WSS vs k

(b) GAP vs k

Figure 19: Statistics used to determine k for k-means clustering

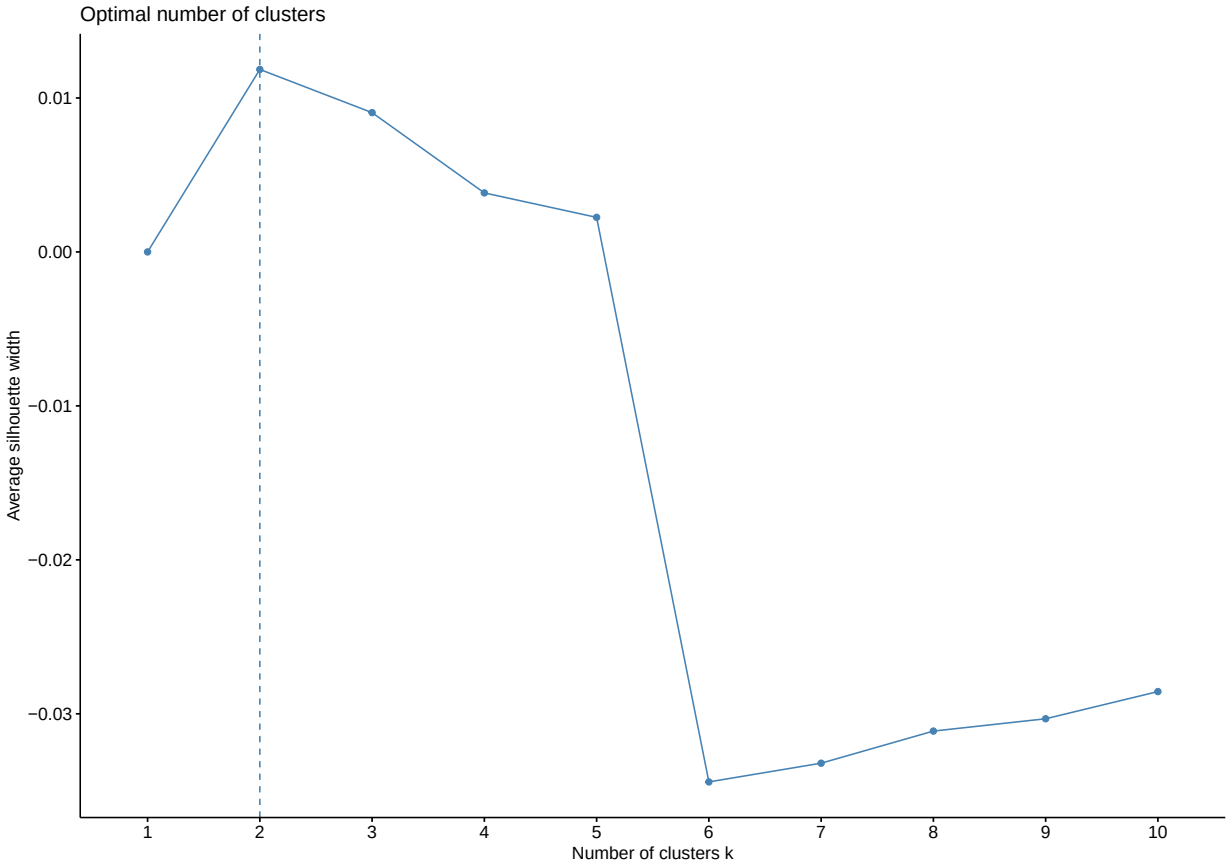


Figure 20: Average silhouette coefficient: hierarchical clustering with Ward's linkage

Appendix C: Ensemble methods

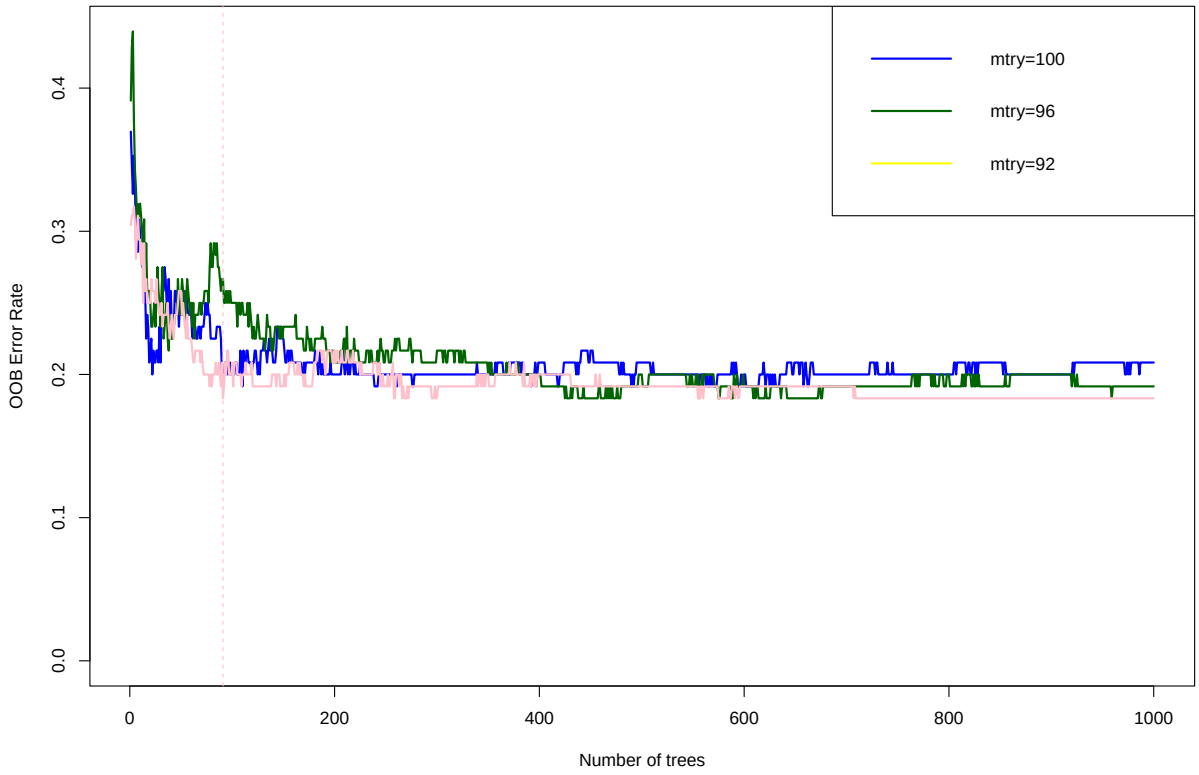


Figure 21: Random Forest: Out-of-bag error rates

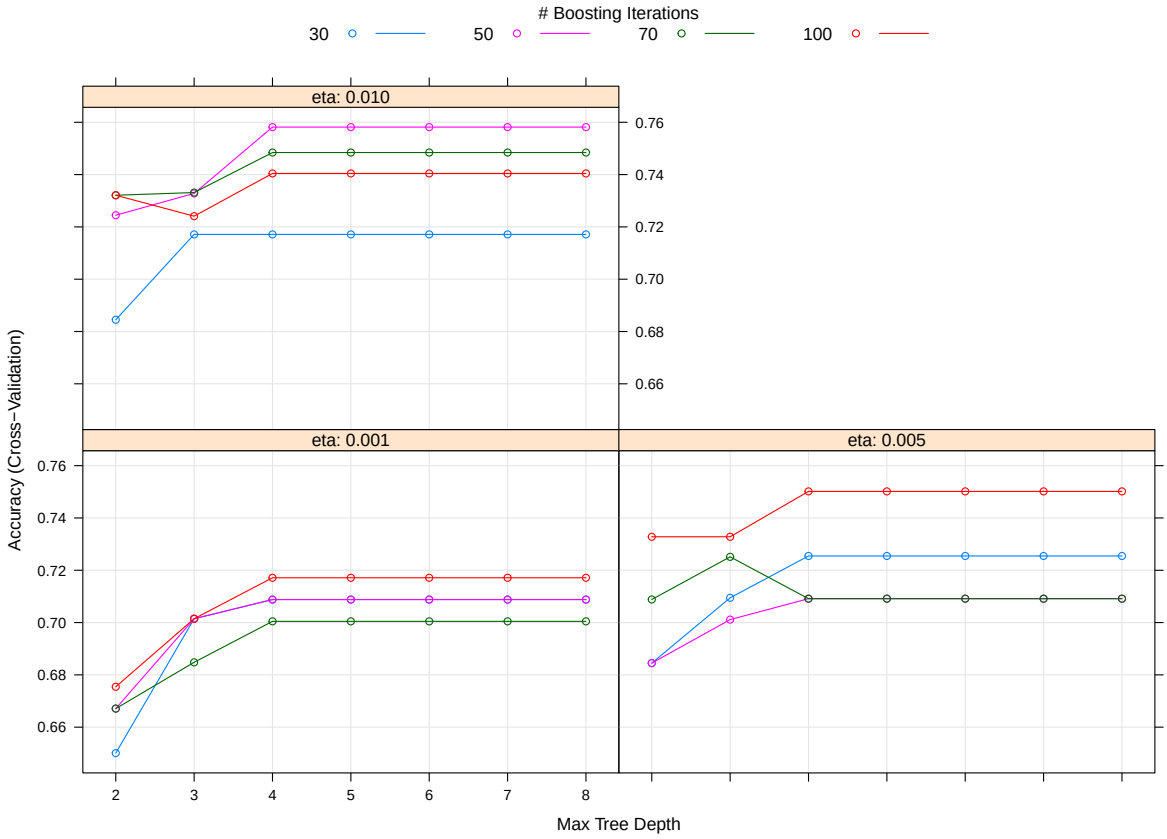


Figure 22: Gradient boosting: Grid search results

Appendix D: Code Listing

Data preparation

```
### DATA PREPROCESSING ###

# read in data on expression
bc_dat <- read.table(gzfile("GSE75688_GEO_processed_Breast_Cancer_raw_TPM_matrix.txt.gz")
    , header = TRUE)
# read in data on sample info
bc_info <- read.table(gzfile("GSE75688_final_sample_information.txt.gz"),header = TRUE)
str(bc_info)

# remove pooled samples and only keep single cell data from sample info
sc_info <- bc_info[which(bc_info$type == "SC"),]
# remove pooled samples and only keep single cell data from expression data
sc_dat <- bc_dat[-(57913:57915),-(4:17)]
# order sample info to match order of expression data (alphabetical)
sc_info <- sc_info[order(sc_info$sample),]
rownames(sc_info) <- 1:515

# the sample info only contains high quality samples where the expression matrix
# contains all samples so match the expression data samples with sample info
```

```
# by removing poor quality samples.
sc_match <- sc_dat[,colnames(sc_dat) %in% sc_info$sample]
sc_match <- cbind(sc_dat[,1:3], sc_match)
gene_names <- sc_match[,2]
rownames(sc_match) <- sc_match[,1]
sc_match <- sc_match[,-(1:3)]

## Pre-process data using Seurat
library(Seurat)
# create seurat object
seurat_dat <- CreateSeuratObject(counts = sc_match)
# reduce dataset to only 10000 most differentially expressed genes
var_gene <- FindVariableFeatures(seurat_dat, selection.method = 'vst', nfeatures = 35000)
# scale data
scale_sc_red <- ScaleData(object = var_gene)
# extract scaled and reduced expression matric from Seurat object
sc_pro_red <- GetAssayData(object = scale_sc_red, slot = "scale.data")
sc_pro_red <- as.data.frame(sc_pro_red)


red_gene_names <-gene_names[which(rownames(sc_match) %in% rownames(sc_pro_red))]
# create vector of sample names for coloring plots
samp <- as.factor(c(rep("BC01",22), rep("BC02",53), rep("BC03",33), rep("BC03LN", 53),
                  rep("BC04", 55), rep("BC05", 76), rep("BC06", 18), rep("BC07", 50),
                  rep("BC07LN", 52), rep("BC08", 22), rep("BC09", 55), rep("BC10", 15),
                  rep("BC11", 11)))

# define palette with 13 colours (for 13 samples)
pal <- c("black", "red", "#663399" , "#008000" , "#ffd700" , "#00ffff" , "#ff9900" ,
        "#0000ff" , "#ff1493" , "#00ff00", "#fa8072",
        "#990000", "#CC99FF")
palette(pal)
```

Dimension Reduction

```
### PCA ###

# perform PCA on data
PCA <- prcomp(t(sc_pro_red))

# plot first 3 PCs and colour by tumour and non-tumour
pairs(PCA$x[,1:3], col = as.factor(sc_info$index))
# plot first 3 PCs and colour by 5 cell types
pairs(PCA$x[,1:3], col = as.factor(sc_info$index3))
# plot first 3 PCs and colour by sample
pairs(PCA$x[,1:3], col = samp)

# plot proportion of variance explained by PC
plot(PCA$sdev/sum(PCA$sdev), type = "n",
     ylab = "Proportion variance explained",xlab = "Principal component")
lines(PCA$sdev/sum(PCA$sdev))
abline(v=30, col = "red")
```

```
sum(PCA$sdev[1:30]/sum(PCA$sdev)) # total prop var explained by first 30 PCs


### t-SNE ###

library(Rtsne)
# perform tSNE with perplexity of 30
tsne <- Rtsne(t(sc_pro_red), dims = 2, perplexity=30,
            verbose=TRUE, max_iter = 500)

plot(tsne$Y[,1], tsne$Y[,2], col = as.factor(sc_info$index),
    xlab = "tSNE 1", ylab = "tSNE 2")
legend("bottomleft",legend = c("non-tumor","tumor"), col = pal, pch = 16)
plot(tsne$Y[,1], tsne$Y[,2], col = as.factor(sc_info$index3),
    xlab = "tSNE 1", ylab = "tSNE 2")
legend("bottomleft",legend = sort(unique(sc_info$index3)), col = pal, pch = 16)
plot(tsne$Y[,1], tsne$Y[,2], col = samp, xlab = "tSNE 1", ylab = "tSNE 2")
legend("bottomleft",legend = unique(samp), col = pal, pch = 16)

### Kernel PCA ###

library(kernlab)

# Get datset with 10000 genes instead of 35000 because kernlab can't handle 35000
var_gene2 <- FindVariableFeatures(seurat_dat, selection.method = 'vst',
                                  nfeatures = 10000)
# scale data
scale_sc_red2 <- ScaleData(object = var_gene2)
# extract scaled and reduced expression matric from Seurat object
sc_pro_red2 <- GetAssayData(object = scale_sc_red2, slot = "scale.data")
sc_pro_red2 <- as.data.frame(sc_pro_red2)
t_sc_pro <- as.data.frame(t(sc_pro_red2))

# do kPCA using polydot
kPCA1 <- kpca(~., data = t_sc_pro, kernel = "polydot",
            kpar = list(degree = 1),features = 10)

plot(rotated(kPCA1), col = as.factor(sc_info$index))
plot(rotated(kPCA1), col = as.factor(sc_info$index3))
plot(rotated(kPCA1), col = samp)

# do kPCA using laplacedot
kPCA2 <- kpca(~., data = t_sc_pro, kernel = "laplacedot", kpar = list(sigma = 0.001),
            features = 10)
plot(rotated(kPCA2), col = as.factor(sc_info$index))
plot(rotated(kPCA2), col = as.factor(sc_info$index3))
plot(rotated(kPCA2), col = samp)

# do kPCA using anovadot
kPCA3 <- kpca(~., data = t_sc_pro, kernel = "anovadot", kpar = list(sigma = 0.001, degree
    = 2),
            features = 10)
plot(rotated(kPCA3), col = as.factor(sc_info$index))
```

```
plot(rotated(kPCA3), col = as.factor(sc_info$index3))
plot(rotated(kPCA3), col = samp)

# do kPCA using rbfdot
kPCA4 <- kpca(~., data = t_sc_pro, kernel = "rbfdot",
            kpar = list(sigma = 0.0001), features = 10)
plot(rotated(kPCA4), col = as.factor(sc_info$index),
     xlab = "PC1", ylab = "PC2")
plot(rotated(kPCA4), col = as.factor(sc_info$index3))
plot(rotated(kPCA4), col = samp)
```

Unsupervised Clustering

```
### Graph-Based (using Seurat) ###

# do PCA
S_obj <- RunPCA(object = scale_sc_red)
# calculate neighbours
S_obj <- FindNeighbors(S_obj, reduction = "pca", dims = 1:30)
# generate clusters
S_obj <- FindClusters(S_obj, resolution = 0.6)
# do tSNE
S_obj <- RunTSNE(object = S_obj)

# plot clusters on PCA graph
DimPlot(object = S_obj, reduction = "tsne")
# plot clusters on tSNE graph
DimPlot(object = S_obj, reduction = "pca")


### k-means (no dim reduction) ###

kmax <- 10
wss <- sapply(1:kmax,function(k){kmeans(t(sc_pro_red),k,nstart = 10)$tot.withinss})
plot(1:kmax,wss,type = 'b',xlab = 'Number of clusters',ylab = 'Total within-cluster sum
    of squares')

set.seed(2002)
km <- kmeans(t(sc_pro_red),2,nstart = 10)
misclass <- length(which((km$cluster-as.numeric(factor(sc_info$index, levels = c("Tumor
    ","nonTumor") , labels = c(1,2))))!=0))/515

clust_means <- km$centers
par(mar=c(7, 4, 4, 2))
plot(c(1,ncol(t(sc_pro_red))),range(clust_means), type = 'n',
     xaxt='n',ylab = 'Average scaled expression',xlab = '')
axis (side=1, 1:ncol(t(sc_pro_red)), red_gene_names, las=2)
colvec <- c("red","black")
lines (1:ncol(t(sc_pro_red)),clust_means[2,],col=colvec[2])
lines (1:ncol(t(sc_pro_red)),clust_means[1,],col=colvec[1])
par(mar=c(5, 4, 4, 2))

#========================== TRANSCRIPTOMICS CLUSTERING ============
```

```
rm(list=ls())
require(tidyverse)
require(Seurat)
library(matrixStats)
library(caret)
library(M3C)
library(factoextra)
#require(cowplot)
#require(scatr)
#require(scran)
#require(igraph)

dat = sc_pro_red

### investigate true labels
table(sc_info$index) #first level
table(sc_info[,3:4]) #second level
table(sc_info[,4:5]) #third level
table(sc_info[,c(3,5)])

#====================== DIMENSION REDUCTION ============
### PCA
pca.1 = M3C::pca(dat)
pca.1
pca.1$data
plot(pca.1$data[,1:3])
plot(pca.1$data[,1:3], col=as.factor(sc_info$index)) #coloured by tumor/non-tumor
plot(pca.1$data[,1:3], col=as.factor(sc_info$index2)) #coloured by type2
plot(pca.1$data[,1:3], col=as.factor(sc_info$index3)) #coloured by type3

#check- should be same as prcomp
tst = prcomp(x=t(dat), center = F, scale. = F)
plot(tst$sdev, type="l", xlab = "Principle Component", ylab="Standard deviation") +
    abline(v=30, lty=2, col="red") # "elbow" plot
plot(tst$x[,1:3])
propvar = (cumsum(tst$sdev^2))/sum(tst$sdev^2) #prop var explained
plot(propvar, type="l", xlab = "Principle Component", ylab="Proportion Variance Explained
    ") + abline(v=30, lty=2, col="red")
propvar[30]


#====================== CLUSTERING ============
#=======determine k=====
#ELBOW METHOD
# within SS for k=2:20
k.max <- 8 # Max no. clusters
set.seed(200)
WSS <- sapply(1:k.max,
              function(k){kmeans(pca.1$data[,1:30], k, nstart=10)$tot.withinss})
plot(1:k.max, WSS, type="b", pch = 19, frame = FALSE, xlab="Number of clusters K", ylab="
    Total within-clusters sum of squares")
```

```
#GAP
library(cluster)
set.seed(200)
gap_stat <- clusGap(pca.1$data[,1:30], kmeans, nstart=10, K.max=8, B=5)
plot(gap_stat, frame = FALSE, xlab = "Number of clusters k", main = "")


###--------------- K-MEANS on PCA

### K=2
#run k-means on first X pca scores
set.seed(200)
km.pca1 = kmeans(pca.1$data[,1:30], centers=2, nstart = 10)

#compare k-means clustering to actual labels
table(km.pca1$cluster)
table(sc_info$index)
pred= as.factor(km.pca1$cluster)
table(pred)
true= sc_info$index
true = ifelse(true=="Tumor", 1, 2)
true=as.factor(true)
table(km.pca1$cluster)
table(true)

cfm1 = confusionMatrix(pred, true)
cfm1

plot(pca.1$data[,1:3], col=as.factor(-1*km.pca1$cluster))
plot(pca.1$data[,1:3], col=as.factor(sc_info$index))


#find mean expressions per cluster
#gene expressions
index.tum = which(km.pca1$cluster==1)
index.nontum = which(km.pca1$cluster!=1)
mean.tum = rowMeans(dat[,index.tum])
mean.nontum = rowMeans(dat[,index.nontum])
plot(mean.nontum, type='l') + lines(1:35000, mean.tum, col="red")

#PCA scores
clust_means <- km.pca1$centers
plot(clust_means[1,], type="l", xlab="Principle Component", ylab="Cluster mean", ylim=c
    (-23, 11), col="red") + lines(1:30, clust_means[2,], col="black")

###-------------- HEIRARCHICHAL
###----determine K
temp = dat[,sample(1:ncol(dat), 0.5*ncol(dat), replace = F)] #random subset for
    computational efficiency and user-sanity maintenance
fviz_nbclust(t(temp), FUNcluster = hcut, method = "wss")
fviz_nbclust(t(temp), FUN = hcut, method = "silhouette")
```

```
### run clustering algorithm
# create dissimilarity matrix - euclidean
d.eucl = dist(t(dat), method="euclidean")

#perform clustering with ward linkage
set.seed(200)
hc.eucl.ward = hclust(d.eucl, method="ward.D2")

plot(hc.eucl.ward)
#Cutting the cluster tree to make
# 2 groups
hc.cut2 <- cutree(hc.eucl.ward,k =2)
# 5 groups
hc.cut5 = cutree(hc.eucl.ward, k=5)

#compare HC clustering to actual labels
hc.cut2 = as.factor(hc.cut2)
table(hc.cut2)
table(true)
str(true); str(hc.cut2)

cfm3 = confusionMatrix(hc.cut2, true)
cfm3 #worse than kmeans
plot(pca.1$data[,1:3], col=hc.cut2)
plot(pca.1$data[,1:3], col=true)

#compare expression profiles of clusters
# Load necessary packages
library(ggplot2)
library(dplyr)
library(pheatmap)
library(gplots)

# Create a dataframe with labels
df <- data.frame(t(dat), hc.cut2) #with expression matrix
df.pc <- data.frame(pca.1$data, hc.cut2) #with PCs

# Rename column with labels to "Type"
colnames(df)[ncol(df)] <- "Type"
colnames(df.pc)[ncol(df.pc)] <- "Type"

#generate PC means for each cluster
hc.tum = df.pc[which(df.pc$Type==1),]
hc.nontum = df.pc[which(df.pc$Type==2),]
pcmean.tum = colMeans(hc.tum[,-ncol(hc.tum)])
pcmean.nontum = colMeans(hc.nontum[,-ncol(hc.nontum)])

# 1. Mean profiles
plot(pcmean.nontum[1:30], type="l", xlab="Principle Component", ylab="Cluster mean", ylim
    =c(-17, 12)) + lines(1:30, pcmean.tum[1:30], col="red")
```

```
# 4. Density plots
ggplot(df.pc, aes(x=df.pc[,1], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
ggplot(df.pc, aes(x=df.pc[,2], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
ggplot(df.pc, aes(x=df.pc[,3], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
ggplot(df.pc, aes(x=df.pc[,4], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
ggplot(df.pc, aes(x=df.pc[,5], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
ggplot(df.pc, aes(x=df.pc[,6], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
ggplot(df.pc, aes(x=df.pc[,7], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
ggplot(df.pc, aes(x=df.pc[,8], fill=Type)) +
  geom_density(alpha=.5) +
  labs(title="Density plot", x="Expression", y="Density", fill="Type")
```

Support Vector Machine

```
### SVM ###
library(kernlab)

class <- as.factor(sc_info$index)
ksvm_dat <- cbind(t_sc_pro,class)
ksvm_dat <- data.frame(ksvm_dat)

# split data into training and test set
index <- sample(1:515, size =360, replace = FALSE)
train <- ksvm_dat[index,]
test <- ksvm_dat[-index,]

# get support vector machine (using same parameters from kernel PCA for kernel)
svp <- ksvm(class~., data = train, scaled = FALSE, type = "C-svc",
           kernel="rbf", kpar = list(sigma = 0.0001), C = 1)
svp

# predict on test set
pred <- predict(svp, test[,-10001])

# get confusion matrix
library(caret)
conf <- confusionMatrix(data = pred, reference = as.factor(sc_info[-index,3]))
```

Ensemble Methods

```
#============================ TRANSCRIPTOMICS - ENSEMBLE METHODS ============
rm(list=ls())
require(tidyverse)
require(Seurat)
library(matrixStats)
library(caret)
library(M3C)
library(factoextra)
library(dplyr)
library(caret)
library(xgboost)
library(randomForest)
### ================ DATA MANAGEMENT ======
setwd("~/OneDrive - University of Cape Town/2023/MSc Biostatistics/Coursework Semester 1/
    Multivariate/Assignment/Datat/Breast Cancer ")
# read in data on expression
bc_dat <- read.table(gzfile("GSE75688_GEO_processed_Breast_Cancer_raw_TPM_matrix.txt.gz")
    , header = TRUE)
# read in data on sample info
bc_info <- read.table(gzfile("GSE75688_final_sample_information.txt.gz"),header = TRUE)
str(bc_info)

# remove pooled samples and only keep single cell data from sample info
sc_info <- bc_info[which(bc_info$type == "SC"),]
# remove pooled samples and only keep single cell data from expression data
sc_dat <- bc_dat[-(57913:57915),-(4:17)]

# REMOVE TUMOUR CELLS
sc_nontum_info <- sc_info[which(sc_info$index != "Tumor"),]

# order sample info to match order of expression data (alphabetical)
sc_nontum_info <- sc_nontum_info[order(sc_nontum_info$sample),]
rownames(sc_nontum_info) <- 1:nrow(sc_nontum_info)

# the sample info only contains high quality samples where the expression matrix
# contains all samples so match the expression data samples with sample info
# by removing poor quality samples.
sc_nontum_match <- sc_dat[,colnames(sc_dat) %in% sc_nontum_info$sample]
sc_nontum_match <- cbind(sc_dat[,1:3], sc_nontum_match)
rownames(sc_nontum_match) <- sc_nontum_match[,1]
sc_nontum_match <- sc_nontum_match[,-(1:3)]

## Pre-process data using Seurat
library(Seurat)
# create seurat object
seurat_dat <- CreateSeuratObject(counts = sc_nontum_match)
# reduce dataset to only 10000 most differentially expressed genes
var_gene <- FindVariableFeatures(seurat_dat, selection.method = 'vst', nfeatures = 20000)
# scale data
scale_sc_red <- ScaleData(object = var_gene)

# extract scaled and reduced expression matric from Seurat object
sc_pro_red <- GetAssayData(object = scale_sc_red, slot = "scale.data")
```

```
sc_pro_red <- as.data.frame(sc_pro_red)

dat = sc_pro_red


###============= RANDOM FORESTS ========
pca.1 = M3C::pca(dat)
dat.pc = pca.1$data[,1:120]
i2 = as.factor(sc_nontum_info$index2)
i3 = as.factor(sc_nontum_info$index3)
dat.pc = cbind(dat.pc, i3)


#split into train and validation sets
set.seed(2022)
train_index <- createDataPartition(y = dat.pc$i3, p = 0.6, list = FALSE)
dat.train <- dat.pc[train_index,]
dat.valid <- dat.pc[-train_index,]

#============================== RANDOM FORESTS ==============================
#tuning params = B, mtry

# Bagging - mtry = 18
set.seed(2022)
 bag_dat1 <- randomForest(i3 ~ ., data = dat.train,
                          mtry = ncol(dat.train) -2,
                          ntree = 1000,
                          importance = F,
                          na.action = na.exclude,
                          do.trace = 100)
# Bagging - mtry = 14
set.seed(2022)
 bag_dat2 <- randomForest(i3 ~ ., data = dat.train,
                       mtry = ncol(dat.train) - 6,
                       ntree = 1000,
                       importance = F,
                       na.action = na.exclude,
                       do.trace = 100)

# Bagging - mtry = 10
set.seed(2022)
 bag_dat3 <- randomForest(i3 ~ ., data = dat.train,
                       mtry = ncol(dat.train) - 9,
                       ntree = 1000,
                       importance = F,
                       na.action = na.exclude,
                       do.trace = 100)


# CONFUSION
bag_dat1$confusion
bag_dat2$confusion
bag_dat3$confusion
```

```
which.min(bag_dat1$err.rate[,1]); min(bag_dat1$err.rate[,1]) #minimum err rate and
    corresponding ntree
which.min(bag_dat2$err.rate[,1]); min(bag_dat2$err.rate[,1]) #minimum err rate and
    corresponding ntree
which.min(bag_dat3$err.rate[,1]); min(bag_dat3$err.rate[,1]) #minimum err rate and
    corresponding ntree


## Compare OOB Errors
par(mfrow = c(1,1))
plot(bag_dat1$err.rate[,1], type = 'l', xlab = 'Number of trees', ylab = 'OOB Error Rate
    ',
     col = 'blue', lwd = 2, ylim = c(0, max(bag_dat1$err.rate[,1], bag_dat2$err.rate[,1])
         ))
lines(bag_dat2$err.rate[,1], col = 'darkgreen', lwd = 2)
lines(bag_dat3$err.rate[,1], col = 'pink', lwd = 2)
abline(v=which.min(bag_dat3$err.rate[,1]), lty=2, col="pink")
legend('topright', legend = c('mtry=100', 'mtry=96', 'mtry=92'),
       col = c('blue', 'darkgreen', 'yellow'), lwd = 2, lty = c('solid', 'solid', 'solid')
           )

#Run optimal forest
nt = which.min(bag_dat2$err.rate[,1])
mt = bag_dat2$mtry
set.seed(2022)
bag_dat <- randomForest(i3 ~ ., data = dat.train,
                        mtry = mt,
                        ntree = nt,
                        importance = TRUE,
                        na.action = na.exclude,
                        do.trace = 100)
plot(bag_dat)

#model performance - INSAMPLE
bag_dat$confusion
min(bag_dat$err.rate[,1])
plot(bag_dat$err.rate[,1], type="l")

#model performance - out of sample
bag_dat.pred = predict(bag_dat, dat.valid)
table(bag_dat.pred)
confusionMatrix(bag_dat.pred, dat.valid$i3)

#============================== GRADIENT BOOSTING ==============================

#Parallelisation initialisation
library(doParallel)
cores <- detectCores() - 2 #Don't use all your cores
cl <- makePSOCKcluster(cores)
registerDoParallel(cl)

#grid search to determine hyperparams
```

```
xgb_grid1 <- expand.grid(nrounds = c( 30, 50, 70, 100), #B
                         max_depth = c(2:8), #d
                         eta = c(0.01, 0.005, 0.001), #lambda
                         gamma = 0.001, #mindev
                         colsample_bytree = 1, #proportion random features per tree
                         min_child_weight = 1, #also controls tree depth
                         subsample = 1 #bootstrap proportion
)
ctrl <- trainControl(method = 'cv', number = 5, verboseIter = T)

set.seed(2022)
 xgb_dat1 <- train(i3 ~ ., data = dat.train,
                   method = 'xgbTree',
                   trControl = ctrl,
                   verbose = F,
                   tuneGrid = xgb_grid1, allowParallel=TRUE)
 stopCluster(cl)
# save(xgb_dat1, file = 'xgb_dat1.1.Rdata')
# load('xgb_dat1.1.Rdata')

#compare xgb configurations
plot(xgb_dat1)
xgb_dat1$bestTune

#model performance of best configuration
xgb_dat1$results["66",]

#compare RF and XGB
# INSAMPLE
bag_dat
xgb_dat1$results["66",]
#accuracy of XGB is slightly better

# OUTOFSAMPLE
xgb.pred = predict(xgb_dat1, dat.valid)
table(xgb.pred)
confusionMatrix(xgb.pred, dat.valid$i3)
confusionMatrix(bag_dat.pred, dat.valid$i3)
table(bag_dat.pred)
#OOS accuracy is better for RF

#====================== VARIABLE IMPORTANCE =========
impToPlot.rf <- importance(bag_dat, scale=FALSE)

varImpPlot(bag_dat, scale = TRUE, type = 1, main="")
#gradient boosted tree

plot(varImp(xgb_dat1, scale = TRUE), main="", xlab="Variable Importance (%)")
```