Master in High Performance Computing

Clustering Strategy for Selection of Relevant Genes in Single Cell Transcriptomics

Supervisor(s):
Matteo Marsili,
Alejandro Rodriguez Garcia,
Alberto Sartori

Candidate:
Florentino Gomes de Oliveira Silva

5th edition
2018–2019

[0]List of Figureslof

Contents

1 Introduction **5**

2 Data clustering and information entropy **8**

2.1 Partition clustering 9

2.1.1 k-means clustering 9

2.1.2 k-medoids clustering 13

2.2 Hierarchical agglomerative clustering 14

2.3 Information entropy 17

3 Selection of relevant genes **19**

3.1 The 'Zeisel' dataset 19

3.2 Partitioning the cell space 22

3.3 Gene expression on clusters 26

3.4 Multi-partition entropy signature 27

4 Results **29**

A Known markers genes figures **32**

A.1 Multi partition relevance curves 32

Bibliography **36**

Abstract

Different unsupervised learning methods have been applied to single cell RNA sequencing datasets, aiming to unveil similarities and correlations between cells and groups of cells. In this thesis, it will be presented a novel theoretical framework based on information entropy to select most informative genes. In order to achieve this goal it was necessary to study data clustering methods that not only could output a meaningful partition of the high-dimension space of the cell dataset, but also that have well-defined clustering parameters. In addition, it focus on methods that perform under low run time complexity and a good scalability profile. As result, it was found a group of marker genes that preserves the clustering structure they are embedded, which biological relevance still under investigation.

Chapter 1

Introduction

One of the goals in modern cancer research is to determine the exact composition of a tumor in order to identify potential treatments that target only specific cells. For long it was believed that a tumor was composed by an homogeneous population of cells which has been shown in the succeeding years a false assumption. In fact, a tumor is a collection of sub-populations of rich heterogeneous cells[41]. Another group of cells that display a high heterogeneity factor are 'brain cells'[38]. Brain cells can either be a neuron and non-neuron cells. The former are cells that communicate with each other by sending electrical impulses through ionic channels. The latter are organized in a non-trivial fashion in the several compartments of the brain. In the case of these two groups (brain and tumor cells), samples of cells were used to analyse the average of gene expression of a given sample, a technique which is called 'bulk analysis'. However, because of biological variability in individual cells, bulk analysis masks the individual fingerprint of each sub-population in the sample. In general, the concept of 'cell type' refers to groups of cells that share the same features, like shape, functionality and location. Nonetheless, the notion of individual variability among cells of same type requested the additional concept of 'cell state'. At some point, cells of a certain type can flip their functionalities temporarily to fulfill a specific mechanism and eventually flip back to their 'ground state'. This heterogeneity can be captured at the molecular level thanks to the technique called single-cell RNA sequencing[2, 22].

The central dogma of molecular biology, stated by Francis Crick[10], explains the flow of genetic information and can be summarized as: DNA (genome) makes RNA (transcriptome), RNA makes proteins (proteome). On one hand, analysis of DNA cannot identify if a gene is active ('ON') or inactive ('OFF'). On the other hand, analysis of proteins are rather difficult due to their high reactiviness. For these reasons, the transcriptome is the optimal molecular fingerprint of a cell.

Output data from single-cell RNA sequencing experiments have soared in the past decade[37], from some hundreds of cells to hundreds of thousand, a rapidly increasing trend also followed by the number of publications. By 2015, next-generation sequencing (NGS)

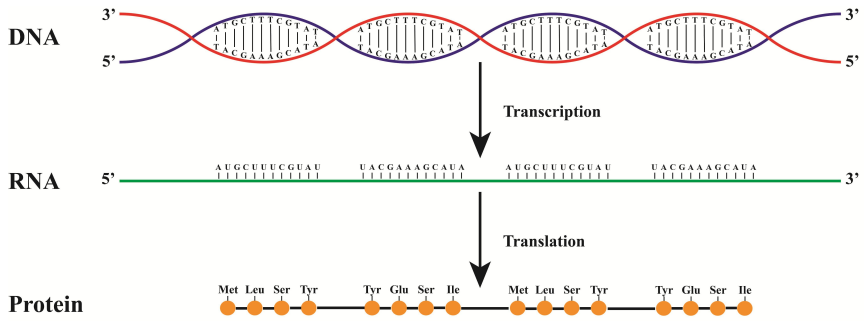

Figure 1.1 – The central dogma of molecular biology, stated by Francis Crick[10].

datasets, which includes single-cell RNA-seq, were already considered a big data problem due to their scale and complexity, generating petabytes (10^{15} bytes) of data[44]. Big data is commonly characterized by its 4 'V's: Volume, Velocity, Variety and Veracity. In the context of Volume, NGS experiments is one of the largest domain of data acquisition[35], with single-cell RNA-seq generating high-volume, high-dimensional expression data of heterogeneous cell populations. In order to process data of this magnitude, from acquisition to statistics, high-performance computing methodology is required and enter the domain of the second 'V', Velocity. The Variety in single-cell RNA-seq raises challenges on how to properly normalize the data, which is an issue in the field[42]. At last, RNA sequencing is a method divided in many steps and at each of these the data can contaminated by noise coming from the experimental procedure and the machinery (technical noise) and the underlying biological process itself (biological noise). This requires a careful quality control, which is the Veracity issue in big data.

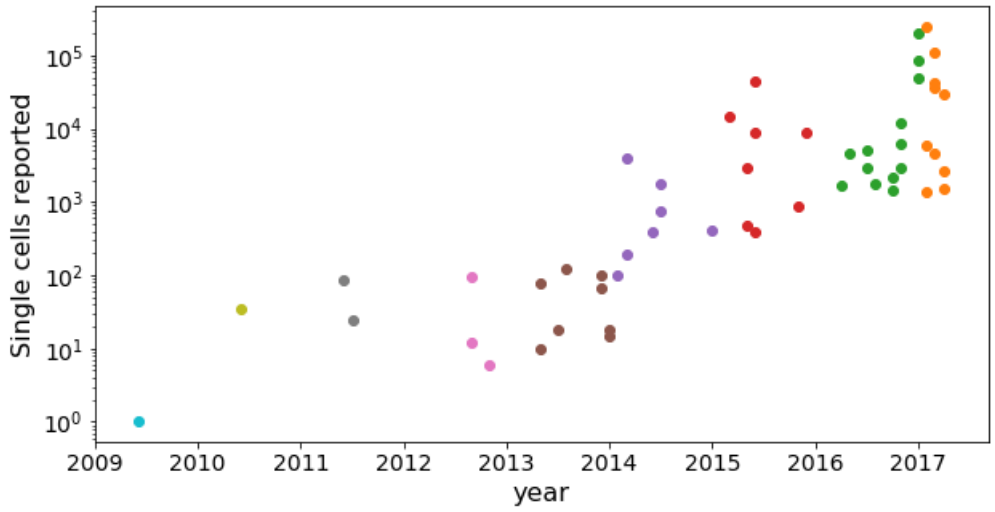

Figure 1.2 – Single-cell RNA sequencing data-size trend[37] from 2009 to 2017.

Different unsupervised learning methods have been applied to single-cell RNA-seq dataset, aiming to unveil similarities and correlations between cells and groups of cells. Theses approaches includes bi-clustering[45], clustering the data twice or three times with

some dimension reduction method in between (ICA or PCA)[31], and more recently using deep neural networks[40].

In this thesis, it will be presented a novel theoretical framework based on information entropy to select most informative genes. In order to achieve this goal it was necessary to study data clustering methods that not only could output a meaningful partition of the high-dimension space of the cell dataset, but also that have well-defined clustering parameters. In addition, it focus on methods that perform under low run time complexity and a good scalability profile. In chapter 2, the clustering methods to be investigated will be introduced, as well as the entropy-based framework that will be cornerstone of this work. Then in chapter 3 the strategies adopted for multi-partitioning the cell space will be explained, why some fails and others succeed. At last, chapter 4 is dedicated to present the main results and further research interests.

Chapter 2

Data clustering and information entropy

Machine learning techniques have been often divided in three paradigms: supervised, unsupervised and reinforcement learning[36]. Supervised learning aims to find the best map from a data space (the samples space) to another data space (the labels space), whereas in unsupervised learning it is assumed that one data space is missing (the labels space) and it must be 'inferred' as accurately as possible[46]. Reinforcement learning is drastically different since the two data spaces (the 'agent' space and the 'environment') interact dynamically and the goal is find the most rewarding outcome from this back and forth interaction.

Unsupervised learning methods are applied in order to perceive common features that can separate the data in meaningful groups. The technique that runs through all the data set looking for regularities and defining different groups based on similarities among data elements is called clustering. Each subgroup of the data set is determined by a similarity aspect is a called a cluster. Cluster analysis is widely used on research areas that rely on data-driven modeling, from biology to economics. In this chapter, two clustering approaches will be presented and then applied throughout this work. The concept of partition clustering is introduced in section 2.1, whereas section 2.2 regards to hierarchical agglomerative clustering.

At section 2.3, some definitions from information entropy will be remarked. Shannon's entropy has been proven extremely powerful as a measurement of 'surprise' in a given stochastically generated data set. Moreover, basic concepts of the critical variable selection (CVS) framework will be discussed since it defines the fundamental quantities to be measured on the single-cell transcriptomics dataset.

2.1 Partition clustering

Given a data set composed of m objects defined on a n-dimensional real space $\mathcal{R}^n$, suppose a search for k groups of this data that share common features according to some similarity measure. This is the main idea behind partition clustering. In other words, given a number k of clusters and a set of criteria that will define the similarities sought, the algorithm will divide the space in k groups and label the elements belonging to each c partition accordingly, $c = \{1, 2, ..., k\}$. This method is sometimes referred to as competitive learning algorithm [25] since each group/partition is striving to label the data points. Two partitioning methods will be discussed: k-means and k-medoids. For simplicity, examples will be based on the Iris dataset[14], which is composed by 150 samples evenly distributed of 3 different species of Iris flower. Each sample carries a measurement of 4 different features of this species: petal length, petal width, sepal length, sepal width, all measured in centimeters. From the point of view of the method, the dataset is composed of 150 vectors on a n-dimensional space with $n = 4$.

2.1.1 k-means clustering

In k-means algorithms the centroid is the object responsible for claiming the data point membership to each partition. Each centroid is a point on a n-dimension real space $\mathcal{R}^n$ whose coordinates are given by the means of the s_α ($\leq m$) coordinates of the points belonging a particular cluster α. In other words, the centroid is the center of mass $\hat{r}_\alpha$ of cluster α with elements cluster elements $r_{\alpha,i}$, $i = \{1, ..., s_\alpha\}$,

$$\hat{r}_\alpha = \frac{1}{s_\alpha} \sum_{i=1}^{s_\alpha} r_{\alpha,i} \,. \tag{2.1}$$

Although other real space metrics can be assumed, the distance is generally defined as the euclidean distance $d(x, y)$,

$$d^2(x, y) = \sum_j (x_j - y_j)^2 \,, \tag{2.2}$$

where x_j and y_j are the coordinates of x and y, respectively. The k-means algorithm allows different ways to choose the first centroids, which will be called seeds and represented by $\alpha_{t=0}$ with $\alpha_t = \{1, ..., k\}$ at any time step t. The different procedures for the election of seeds will be discussed later on. Once the seeds for each centroid $\alpha_{t=0}$ are initialized, each data point is assigned to the nearest seed and then forming the first partitions. Afterwards, each new centroid $\alpha_{t=1}$ is evaluated using the data points on that partition $\alpha_{t=0}$ and then reassigning the data points according to the new centroids $\alpha_{t=1}$. This procedure is known as Lloyd's algorithm [23] and the steps can be summarized as following:

1. Select k seeds, which are points in the n-dimensional space $\mathcal{R}^n$.
2. Assign each data point to its nearest centroid.

3. Recalculate the centroids as the average of all data points within each cluster.
4. Re-assign data points to their nearest new centroids.
5. Continue steps 3 and 4 until the method either converges (centroids were captured by some local minima), or the maximum number of iterations is reached.

Suppose that a new point is added to a dataset that has already been labeled, in this case the incomer point will be assigned to its nearest cluster and the centroids reevaluated in order to accommodate the new configuration. The constraint on the data set for assessing if the centroid has reached an optimal configuration is called the cost function Φ_{cost}. For k-means, the cost function is the sum of the 'dissimilarity' measured on each centroid. As remarked on the step 5, this update goes on until the method converges. This will The k-means algorithm always converges to a fixed point [25], *one* of the many possible local minima. In other words, after a number of iterations the fluctuations in the position of the centroids are minimized, the centroids have been captured by *one* optimal position. As an example, the projections of the Iris dataset for $k = 3$ clusters are shown Figure 2.1.

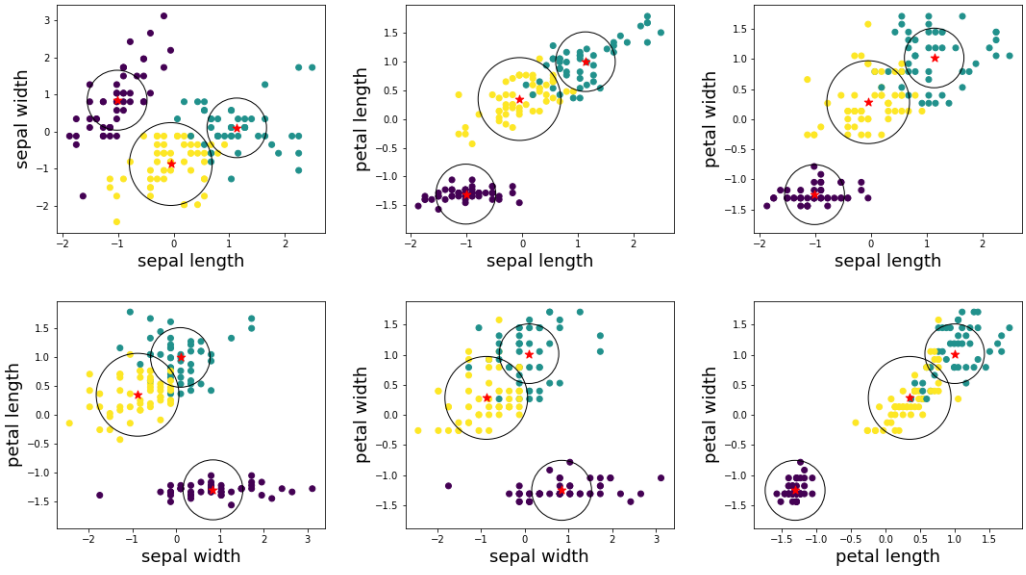

Figure 2.1 – The six projections of the Iris dataset as a results of k-means clustering with $k = 3$. The data points are the coloured *dots*, each colour indicates a different cluster. The *red stars* indicates the centroids and the *black circles* their gyration radius.

The dissimilarity is also referred as the inertia. As known from percolation theory [9], the radius of gyration squared is defined as the average square distance of each point

$r_{\alpha,i}$ within a cluster α to its centroid $\hat{r}_\alpha$, $\alpha = \{1, ..., k\}$,

$$R_\alpha^2 = \frac{1}{s_\alpha} \sum_{i=1}^{s_\alpha} (r_{\alpha,i} - \hat{r}_\alpha)^2 \,. \tag{2.3}$$

The radius of gyration is a measurement of compactness of the cluster. As a matter of comparison, it is as if the cluster mass was homogeneously distributed on a ring of radius R_α. From dynamics of rigid bodies [29], the momentum of inertia of the homogeneous ring is given by $I = mR_\alpha^2$, which implies referring to the cost function Φ_{cost} simply as (total) inertia. From this perspective, the cost function Φ_{cost} is given by the sum of each clusters α inertia (Equation 2.4). Henceforth, the term 'inertia' will refer exclusively to the total inertial, Φ_{cost}.

$$\Phi_{cost} = \sum_{\alpha=1}^{k} R_\alpha^2 \tag{2.4}$$

As the goal is minimize Φ_{cost}, it also means that algorithm seeks for k clusters as compact as possible, given a certain initialization. As the number k of clusters increases, more compact the clusters will be, and consequently decreasing the inertia. Dimensions with higher variance tend to unbalance the distance and consequently bias the inertia. For a fare trade among the dimensions, the following transformation is applied to the data points

$$\bar{r}_{i,j} = \frac{r_{i,j} - \mu_j}{\sigma_j} \quad , \tag{2.5}$$

where μ_j stands for the mean value over the dimension j and σ_j its standard deviation, with $j = \{1, ..., n\}$. This procedure is called standardization of the dataset. As shown in Figure 2.2, the original data points $r_{i,j}$ reach lower minima than their standardized counterparts $\bar{r}_{i,j}$. Nevertheless, this not a good practice since the inertia is biased by the dimensions with larger fluctuations.

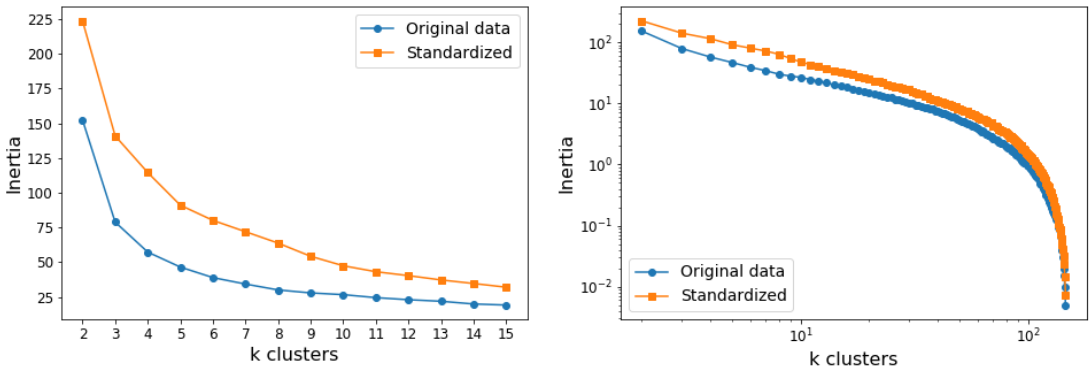

Figure 2.2 – Inertia as function of number of clusters, $\Phi_{cost}(k)$, for k-means clustering the original data points (*blue dots*) and the standardized data(*orange squares*).

This work is focused on implementations that display a better performance, both in terms of time complexity and scalability. The exact solution for k-means is claimed to have time complexity $\mathcal{O}(m^{kn+1})$[20], where m is the number of elements to be clustered, n is the dimension of the space, and k is the number of clusters. Despite that, most implementations are based Lloyd's algorithm[23] which has runtime complexity $\mathcal{O}(kmn)$ per iteration[19]. This is the case for k-means implemented as part of the Scikit-Learn (`sklearn`) cluster package [30] and at the Nvidia RapidsAI machine learning (`cuML`) package [39].

Although `sklearn` and `cuML` are bounded under the same complexity, their approaches for initializing the first centroids differ a great deal. As the method is sensitive to its initial conditions, the random initialization can yield badly positioned centroids. By random initialization, one should understand as randomly choosing k points in the n-dimensional space and setting them as first centroids. This allows two seeds to be spotted on the same cluster, eventually being captured by local minima that may not be consistent with the data distribution. This kind of bias is approached by `k-means++`[3], found at `sklearn` package. The general idea behind it is the following: given $c = 3$ clusters, the first centroid seed c_1 is placed randomly in space; the second centroid c_2 is not chosen randomly, instead its position is output by a distribution $\phi(c_1)$ that takes into account the position c_1 to place c_2 far from it; that third centroid c_3 will be placed far from both c_1 and c_2 due to the updated joint distribution $\phi(c_1, c_2)$. `k-means++` deliver better results than a random initialization, but its drawback is related to the update of the $\phi(c_i)$ distribution. As the distribution $\phi(c_i)$ most be updated for each seed c_i, this approach is not scalable[5], which is the reason why this implementation still widely used for problems that present (relative) low dimensionality and search for small number of clusters, $k \ll m$, but becomes troublesome when searching for high-dimensional k clusters with $k \approx m$.

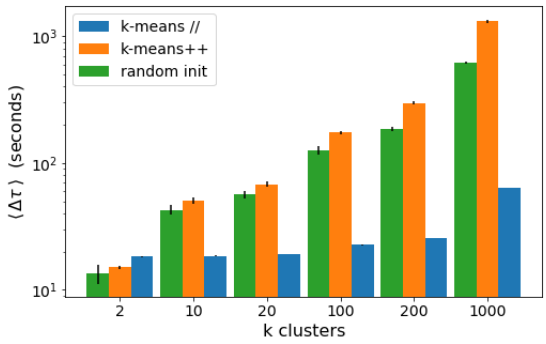

Figure 2.3 – Performance of different `k-means` implementations. Averaged time elapsed $\langle \Delta\tau \rangle$ over 10 realizations each, with same convergence parameters and oversampling factor $l = 8$. Data set dimensions: $n \approx 20,000$, $m \approx 3,000$. The `sklearn` implemtentations benchmarked in a Intel i5-4440 3.10GHz CPU, whereas `cuML` was benchmarked in a Nvidia Quadro-GP100.

The improved version of `k-means++` was named scalable `k-means++` and often referred simply as k-means parallel (`k-means//`)[5]. It defines a oversampling factor l and instead of initializing c seeds it starts with c_l seeds, $c_l = l \times c$. Broadly speaking, once this c_l seeds are initialized, the algorithm ranks them by a given cost function and then select the c better performing (in the sense of its algorithm) to be the centroids. This process is not only scalable, but also can be parallelized. `k-means//` is implemented at Nvidia `cuML` for GPUs with computer capability greater or equal to 6.0 [39]. Comparative plot of performance at Figure 2.3.

k-means in especially sensivite to outliers. An outlier is a data point that deviate significantly from other points of the group/cluster it belongs[17]. However, the outlier affects severely the position of the centroid, thus compromising the quality of the final clustering configuration. This issue can be suppressed by the adoption of k-medoids methods.

2.1.2 k-medoids clustering

Instead of representing each cluster using a centroid, k-medoids method main object is identified by its medoid. The medoid is a data point which has least total distance to the other members of its cluster. In other words, the medoid is chosen by the same criteria as the centroid, the minimization of the the cost function Φ_{cost} (Equation 2.4). By constraining the centre to the manifold where the data points dwell, other distance metric can assumed apart from the Euclidean distance. In addition, k-medoids can be more robust to outliers than k-means. Nevertheless, k-medoids like k-means is sensitive to initial conditions, which can be problematic on large high-dimensional datasets. The most used algorithm for k-medoids the PAM algorithm[21], which steps are the following:

1. Select k data points to be assinged as the first medoids.
2. Assign each data point to its nearest medoid.
3. Select a new data point and swap it with its medoid.
4. Reassign every point to its nearest medoid.
5. Calculate the inertia, if it smaller cost is smaller, keep the new point as a medoid.
6. Repeat steps 3-5 until the medoids don't change.

Due to the medoids constrain to the data manifold, the inertia for an initial trial considering $k = 2$ is lower than its counterpart in k-means. Moreover, as k is incremented the inertia decays slower than in k-means, revealing the manifold constrain on updating the positions (Figure 2.4).

Both k-means and k-medoids rely on measuring the distance matrix. k-means distance matrix is calculated for each new centroid, while in the case on k-medoids the distance matrix is from point to point in all the dataset and shall be calculate only once.

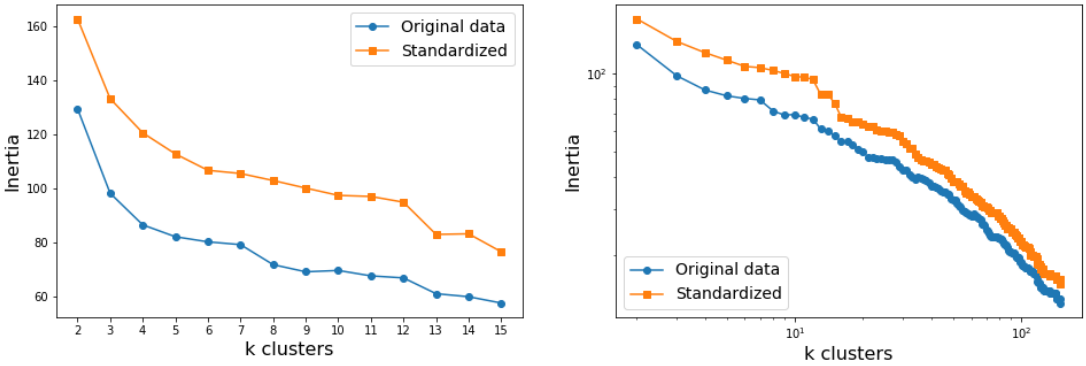

Figure 2.4 – Inertia as function of number of clusters, $\Phi_{cost}(k)$, for k-medoids clustering the original data points (*blue dots*) and the standardized data(*orange squares*).

k-medoids runtime complexity per iteration is $\mathcal{O}(k(m-k)^2)$ [26], due the possible flips. In a recent work[32], PAM algorithms performance were reviewed and improved towards big data applications.

2.2 Hierarchical agglomerative clustering

So far, the control parameter k was an essential input information for the methods. This does not hold true for hierarchical agglomerative clustering (HAC). For the HAC approach, each data point starts as a cluster itself. Given a certain metric as criteria, the clusters are linked together. The procedure goes on merging two cluster at a time until all clusters are combined into one single cluster. The algorithm outputs is a tree with all the connection (links) create. The method proceeds as follows:

1. Define each data point as a cluster.
2. Calculate the pairwise distances D between every clusters.
3. Combine the two clusters that have the shortest distance D, reducing the number of clusters by one.
4. Steps 2 and 3 are repeated until all clusters have been merged into one cluster which contains all data points.

The distances between clusters in point 2 are trivially to compute while the clusters have a single component. However, there is more than one way of computing this distance when the clusters have more than one component and it is called the linkage between clusters. Each linkage defines a different HAC methods. The linkages are listed below, in order to simply the notation, d indicates the Euclidean distance as in Equation 2.2.

- Single linkage. The distance D_{sing} between clusters u and v is the minimum distance among all distances between the clusters elements u_i and v_j,

$$D_{sing}(u,v) = min\{d(u_i,v_j)\} \quad . \tag{2.6}$$

- Complete linkage. The distance D_{comp} between clusters u and v is the maximum distance among all distances between the clusters elements u_i and v_j,

$$D_{comp}(u,v) = max\{d(u_i,v_j)\} \quad . \tag{2.7}$$

- Average linkage. The distance D_{aver} is the average distance all distances between the clusters elements u_i and v_j,

$$D_{aver}(u,v) = \langle\, d(u_i,v_j)\,\rangle \quad . \tag{2.8}$$

- Weighted linkage. The distance D_{weig} between clusters u and v is mean of the distance between v and the 'parent' clusters of u: s and t,

$$D_{weig}(u,v) = \frac{d(s,v) + d(t,v)}{2} \quad . \tag{2.9}$$

- Centroid Distance. The distance D_{cent} between the centroids of clusters u and v,

$$D_{cent}(u,v) = d(u_{cm}, v_{cm}) \quad . \tag{2.10}$$

- Ward linkage. The distance D_{ward} is based on the Ward variance minimization algorithm [7]. Considering s and t the 'parent' clusters of cluster u, $|.|$ the cardinality operator, and $T = |s| + |t| + |v|$,

$$D^2_{ward}(u,v) = \frac{|v|+|s|}{T} D^2_{ward}(v,s) + \frac{|v|+|t|}{T} D^2_{ward}(v,t) - \frac{|v|}{T} D^2_{ward}(s,t) \; . \tag{2.11}$$

The output tree can be visualized with a dendrogram. For example, a dendrogram for the Iris dataset is plotted at Figure 2.5. The dendrogram highlights how the clusters where combined into one single clusters, starting from data points and moving bottom up. The dendrogram height dimension indicates the value used as criteria to join the clusters.

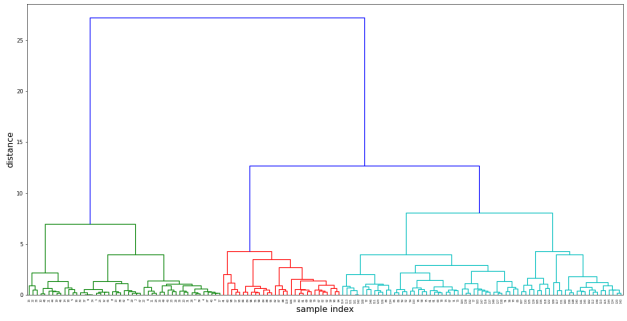

Figure 2.5 – Dendrogram for the Iris dataset showing 3 clusters (*green*,*red* and *cyan* branches).

Hierarchical agglomerative clustering is implemented in Scipy library[43] and has all the distances D presented. The pairwise generated dendrogram be approached as a partitioning of the dataset just like k-means and k-medoids. For example, at Scipy hierarchical clustering package there is method (`fcluster`) that finds a cluster given a criterion, which can be the maximum number of clusters t. In that way, it can be performed a partition of the data space similarly to partioning methods, with t playing the same role as k (Figure 2.6).

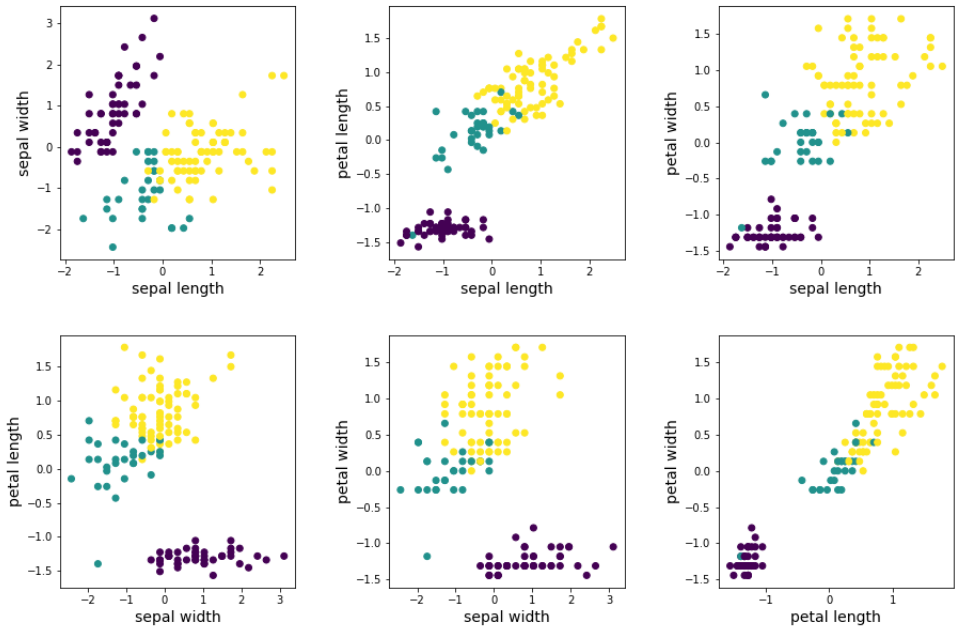

Figure 2.6 – Projections of the Iris dataset for 3 clusters generated by HAC.

One possible drawback of applying HAC in order to find partitions is the fact that its runtime complexity is $\mathcal{O}(m^3)$. That is greater that $\mathcal{O}(k(m-k)^2)$ for k-medoids and $\mathcal{O}(nkm)$ for k-means. Despite that suppose n has the same order of magnitude of m, $m \approx n$, and that the goal is run all possible values of $k = \{1, ..., m\}$. This turns k-means unfeasible. k-medoids can have a chance to compete if carefully rearranged. The superiority of HAC is due the fact the it runs all the linkage at once. The partioning of cluster that happens a posteriori can be costy, however the main point is that HAC is not sensitive to initial conditions, property that will be explored later.

2.3 Information entropy

The concept of entropy as introduced by Shannon[34] is measurement of predictability of a stream of data [33] and has found application in many research areas. As as example, in transforming files using a given compression method, measuring the entropy gives the minimum amount of the bits of information needed to compress the file without information loss. In other words, entropy measures the average uncertainty in a given probability distribution [6], a measure of variability[13] of the outcomes of the process the generates the data. On one hand, outcomes with low probability do not contribute much to the entropy for being rare and outcomes with zero probability has no contribution, since they will not occur. On the other hand, the higher is the probability of an outcome, the lower is its contribution to the entropy. According to Shannon, if an outcome s occurs with probability $p(s)$, where $\sum_{s\in S} p(s) = 1$, then the uncertainty can be calculated as $-\log p(s)$. Thus, the entropy $H(S)$ can be calculated as

$$H(S) = -\langle \log_2 p(s) \rangle_{p(s)} = -\sum_s p(s) \log_2 p(s)\,, \tag{2.12}$$

and henceforth called resolution.

Suppose that a given outcome is triggered by another process with well-defined control parameters which, for simplicity, will be called as 'perturbation' and its control parameter as 'amplitude'. The outcomes that are informative about this given perturbation should be more unpredictable when the perturbation arises from different amplitudes than when the same components of the amplitude trigger similar perturbations This is an example for introducing the concept of mutual information[13]. The mutual information is decrease in the uncertainty of the outcome x once the amplitudes on the perturbation $y \in Y$ are known,

$$I(X;Y) = H(X) - H(X|Y)\,, \tag{2.13}$$

where $H(X|Y)$ is the condition entropy of X given Y,

$$H(X|Y) = -\sum_x \sum_y p(x,y) \log_2 p(x|y)\,. \tag{2.14}$$

Consider a sample of N data points $\hat{S} = (s_1, ..., s_N)$, where s_i are independent and identically distributed by a variable drawn from a distribution $p(s)$ that is unknown. This distribution will be referred as the 'generative model'. The amount of information that the sample $\hat{S}$ contains about its generative model is called the relevance of the sample and is given by

$$H[K] = -\sum_{k=1}^{\infty} \frac{km_k}{N} \log_2 \frac{km_k}{N}\,, \tag{2.15}$$

where m_k is the number of unique outcomes that occurred k times in the sample. The relevance is then the entropy associated with the occurrence of different outcomes k in the sample $\hat{S}$.

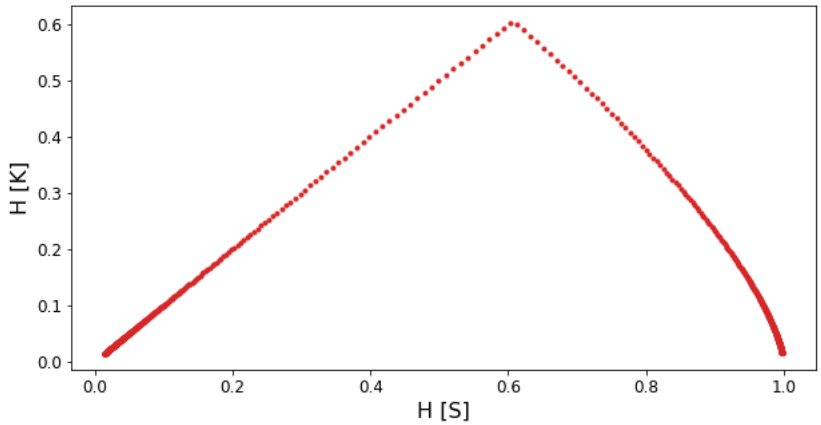

Figure 2.7 – Relevance-resolution ($H[K]$,$H[S]$) curve.

Recently, Cubero *et al.*[11] argued that maximally informative samples are those that maximize the relevance at a given resolution. The theoretical curve for the relation of $H[K]$ and $H[S]$ i given in Figure 2.7. The maximum informative samples exhibit *statistical criticality* and its signature is given by the relation[18, 27, 28]

$$m_k \propto k^{-\mu-1}\,. \tag{2.16}$$

This framework of maximizing relevance at a given resolution has been successfully used to characterize informative neurons [12] and relevant positions in proteins[16]. Moreover it will be the main theoretical approach to guide the measurements and ideas in this work , henceforth referred here as 'critical variable selection' (CVS) framework.

Chapter 3

Selection of relevant genes

The main goal of the present work is to apply critical variable selection (section 2.3) to the single-cell RNA sequencing dataset. As was demonstrated for neurons, the main idea is to group cells into subsets given a certain partitioning criteria. For the case if the neurons, the natural criteria was defined by the time scale, in which the spike train is partitioned in pre-set time bins and the measurement of relevance in terms of resolution was called multi-scale relevance. For the case of single cell RNA-seq, the partitions will be defined by clustering the cells in the dataset, throughout all possible values of k. Instead of the time scale, the relevance is a considered for all possible partitions and resolutions, a multi partition relevance.

3.1 The 'Zeisel' dataset

The single-cell RNA-seq dataset to be explored was published by Zeisel *et al.*[45] and will be referred as 'Zeisel dataset'. This dataset contains 3005 samples of different cell types in the somatosensory cortex and in the CA1 region of the hippocampus of a mouse (*Mus musculus*) brain (Figure 3.1).

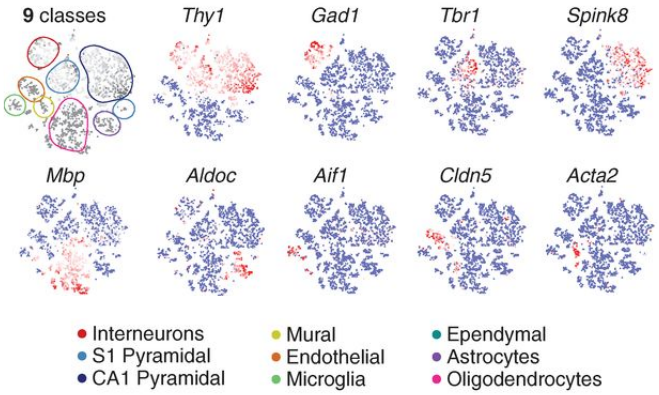

Figure 3.1 – Clustering structure published by Zeisel *et al.*[45] on the dataset.

Each sample is a cell of 9 different classes, see Figure 3.1. The total number of genes captured for these cells is 19972. By applying a biclustering algorithm on this data, that is clustering cells after having the genes clustered, Zeisel *et al.* could identify 47 subclasses of cells. This dataset can be found online at the Gene Expression Omnibus (GEO)[15] which is a public functional genomics data repository and at a repository maintained by the Hemberg Group on quantitative models of gene expression at Sanger Institute [4].

The 'Zeisel' dataset is a record of digital gene expression (DGE). A DGE is a table of counts of how many times a given gene was identified on a certain cell. In this case, it was used UMI (unique molecular identifier) method and confirmed by FISH (flourescent in situ hybridization) method.

Cell type	Number of cells
Interneurons	290
S1 Pyramidal	390
CA1 Pyramidal	948
Oligodendrocytes	820
Microglia	98
Endothelial	175
Astrocytes	198
Ependymal	26
Mural	60

Table 1 – The 3005 cells on 'Zeisel' dataset distributed among the 9 types of cells.

Many genes are exclusively expressed in some specific cell type, this genes are called marker genes, since they are the first to identify a target cell type. Each cell type has a different expression profile, just like a unique fingerprint of the cell. In addition, for each cell profile a large number of genes are not expressed in the DGE table. The main factors for that is, firstly, the gene is simply not part of that cell functionality, and secondly because current technology still not capturing a lot of genes. As consequence, the original dataset is a sparse matrix. This sparse matrix changes to a dense one since the standardization procedure is required for preventing the data analysis to be dominated by genes with large variation – just like in the example with the Iris dataset (Equation 2.5). The standardization of the original data points is given by the transformation below,

$$z^*_{i,j} = \frac{z_{i,j} - \mu_j}{\sigma_j} \quad , \tag{3.1}$$

where $z_{i,j}$ the count of gene j in cell i. The mean counts on gene j and the standard deviation over all i cells are given by μ_j and σ_j, respectively.

At each measurement, the number of genes detected on one cell may differ (and most likely will) from the number in another cell of the same cell type. This demands a normalization by the total number of counts within a cell, which gives each gene count as a fraction of the total expression of that cell. The common procedure in bioinformatics is

to multiply this fraction by 1 million and use the counts per million (cpm) scale[24]. The 'counts per million' transformation is given by Equation 3.2,

$$\bar{z}_{i,j} = \frac{z_{i,j} \cdot 10^6}{\sum_i z_{i,j}} \quad , \tag{3.2}$$

with i index for cells and j for genes. As expected, the $\bar{z}_{i,j}$ values must also be standardized (Equation 3.3),

$$\bar{z}^*_{i,j} = \frac{\bar{z}_{i,j} - \bar{\mu}_j}{\bar{\sigma}_j} \quad . \tag{3.3}$$

The marker gene 'Thy1' is found in 3 types of cells: Interneurons, S1 Pyramidal, and CA1 Pyramidal. The sum of these cells is greater that half of total number of cells, which makes 'Thy1' a good candidate to exemplify how the transformations in Equation 3.2 and Equation 3.3 affect the counts distribution throughout the cells. The histograms are given in Figure 3.2.

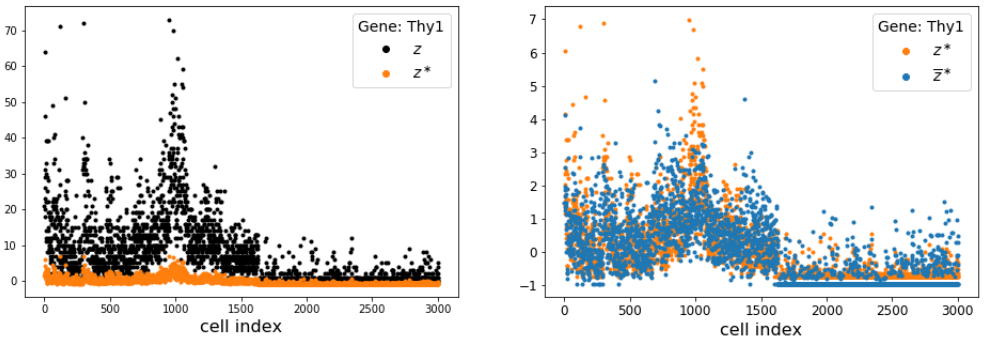

Figure 3.2 – 'Thy1' marker gene expression profile z under standardization z^* and counts per million transformation followed by standardization, $\bar{z}^*$.

Other marker genes display more localized distributions since they relate to shorter sequences of cells, as can be seen in Figure 3.3. This is the most significant feature on classifying a gene as a marker gene: how precisely it helps tagging a particular cell type.

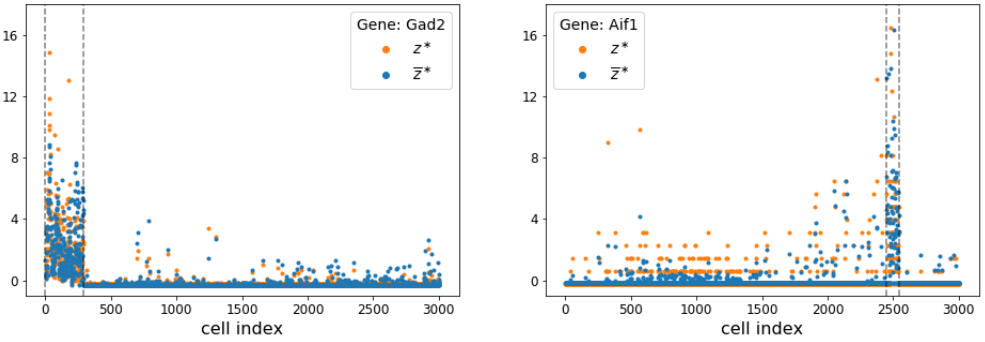

Figure 3.3 – Gene expression profiles z^* and $\bar{z}^*$ of marker genes 'Gad2' and 'Aif1'.

3.2 Partitioning the cell space

As the aim is to measure $H[K]$ and $H[S]$ on partitioned configurations of the cell space and having genes as dimensions of this space, approach with partition clustering comes naturally. For k-means, it has been already discussed the best performance comes with `k-means//` (Figure 2.3). In addition, the performance of `k-means//` was measure on the 'Zeisel' dataset in two different GPU models (Figure 3.4). One in a Quadro-GP100 (16Gb) at ICTP cluster computer Argo, and the other in a Tesla-P100 (16Gb) at Google Cloud online platform. Although these two GPU have very similar technologies embedded, for example both are Pascal architecture with 3584 CUDA cores, the Quadro-GP100 performs slightly better. A performance test was also run in a Quadro-P400, also Pascal architecture but with only 256 CUDA cores which explain a poor performance, far below the first two. The file that contains the dataset has size 121 MBytes, which presents no challenge for neither of the gpus tested. Henceforth, all k-means clustering on 'Zeisel' dataset cited are `k-means//` and run at Quadro-GP100 gpus.

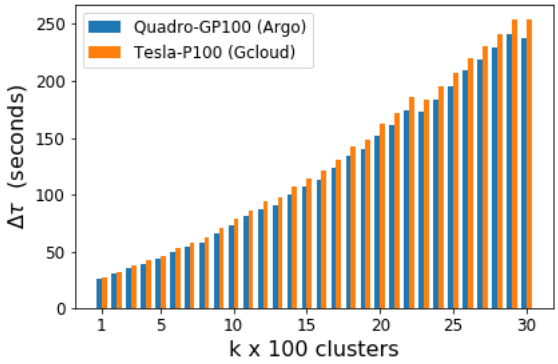

Figure 3.4 – Single gpu benchmark for clustering 'Zeisel' dataset using `k-means//`.

After the clustering is done for all values of $k_c = \{1, ..., 3005\}$ (k_c is adopted trying to clarify the notation), the output is a table of dimensions (3005,3005) indicating the number of cells and the maximum number of k_C partitions, respectively. The cluster labels are limited by the values of k_C. For example, for $k_c = 4$, the set of labels is $c = \{1, 2, 3, 4\}$. The labels table is used then to rearrange the expression profile table, in a way that the cells with same label are grouped together.

Although the counts for each genes at a particular cell is an integer, it will be consider that the at each cell i the gene j can assume only two states $g_{i,j} = \{0, 1\}$. State $g_{i,j} = 0$ corresponds to $z_{i,j} = 0$, the gene is 'off' (inactive, not expressed) at cell (i, j). State $g_{i,j} = 1$ corresponds to $z_{i,j} \geq 1$, meaning that a gene that is 'on' or active at cell (i, j), this mapping 'binarizes' the data (Equation 3.4).

$$\begin{cases} z_{i,j} = 0 \Rightarrow g_{i,j} = 0 \\ z_{i,j} \geq 1 \Rightarrow g_{i,j} = 1 \end{cases} \tag{3.4}$$

These two states $g_{i,j}$ will be explored in more detail in section 3.3. For now, suppose that a hypothetical gene named 'Ref.' (short for reference) has the special property of being expressed in all the 3005 cells, that is $g_i(\text{'Ref.'}) = 1$ for all i. The fact that this gene is always 'on', it will be the reference of how information was stored by the clustering process itself.

The concepts and definitions introduced at section 2.3 will be adapted for the data space of single cell RNA seq, since the goal is to search for maximally informative genes. Moving to the framework of CVS, the set of expression counts $g_{i,j} = 1$ at each cluster is given by,

$$S = \{k_1, ..., k_s, ..., k_C\}\,. \tag{3.5}$$

In the case of the reference gene 'Ref.', the k_s values S are equal to the size of each cluster. The number of total expression counts for each gene,

$$M = \sum_i g_{i,j}\,, \tag{3.6}$$

which for 'Ref.' will equal to the total number of cells N_{cell}. At last, m_k is the number of clusters of size k on a given space partition k_C. Then,

$$M = \sum_s^C k_s = \sum_k^\infty k m_k\,, \tag{3.7}$$

where m_k is the 'occurrence' factor of size k. At last, the resolution and relevance are the following,

$$\begin{cases} H[S] = -\sum_s^C \dfrac{k_s}{M} \log \dfrac{k_s}{M} \\ \\ H[K] = -\sum_k^\infty \dfrac{k m_k}{M} \log \dfrac{k m_k}{M}\,. \end{cases} \tag{3.8}$$

Given the table with the labels, the gene is 'processed' (further details in section 3.3) and the values of $H[K]$ and $H[S]$ is calculated for each $k_C = \{1, ..., N_{cell}\}$. The result for k-means clustering is given at Figure 3.5. The points in curve for $H[K]$,$H[S]$ display large fluctuation. The main reason for that is the sensitivity of k-means to initial conditions. In other words, each k_c initialization is a captured by a local minimum that is likely to be different of the local minima of its neighbor points in this curve, $k_c - 1$ and $k_c + 1$. This behaviour is also clearly noticed from the the plots of $H[K]$ and $H[S]$ in terms of k_c, Figure 3.6.

Despite running the algorithm in a powerful GPU, each k_c-means clustering spent from 20 seconds ($k_c = 1$) to 230 seconds ($k_c = 3005$), resulting in a total run time of 55 hours and making this clustering strategy impractical.

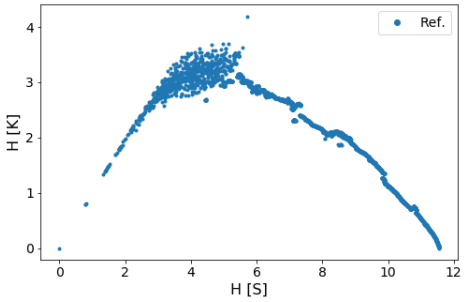

Figure 3.5 – `k-means//` solution for $H[K]$,$H[S]$.

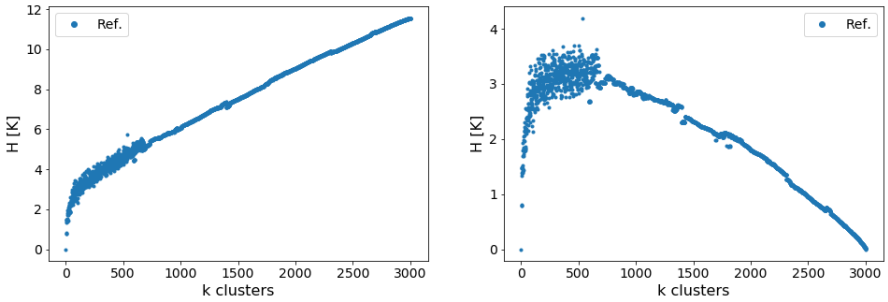

Figure 3.6 – `k-means//` solution for $H[K]$ and $H[S]$ for k clusters.

Moving away from k-means, the second strategy for partitioning the cell space is based on k-medoids. As known, k-medoids PAM algorithm performs better for both time complexity and memory usage, although the sensitivity to initial conditions still a problem. In order to overcome the initialization issue, the complete k_c-medoids run (for all values of k_c) was adapted, inspired by the bottom up approach found in hierarchical agglomerative clustering. The 'next' $k_c - 1$ has as initial conditions input a subset of the previous k_c resulting medoids.

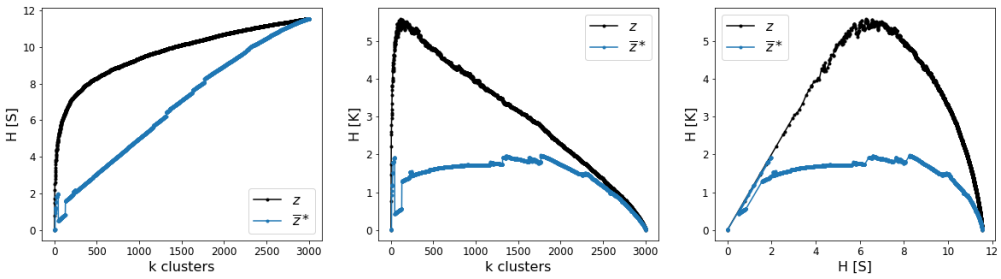

Figure 3.7 – k-medoids with 'perturbative decremental' search for z and $\bar{z}^*$.

The algorithm begins with a `k-medoids++` initialization for $k_c^{max} = 3005$, which will lead to a global mininum, the trivial solution. Beside the labels output, the algorithm now yields the $k_c - 1$ medoids with highest inertia values, meaning that the lowest inertia

medoid is removed. The $k_c - 1$ data points are used as initialization in the next search for $k_c - 1$ medoids. In some sense it performs a 'directed' or 'perturbative decremental' search as the $k_c - 1$ medoids will re-arrenge themselves based on the previous configuration. Proceeding on the CVS framework, the results for the curves of $H[K]$ and $H[S]$ can be found in Figure 3.7. The outcomes of this strategy are smoother curves compared to the previous approach, with fluctuations considerably reduced. Nevertheless, the results for $\bar{z}^*_{i,j}$ are far below the curves for z.

The third strategy is using hierarchical agglomerative clustering by testing the 6 linkages defined in section 2.2 to create the whole dendrogram and then proceeding with cluster identification. The results for z and $\bar{z}^*_{i,j}$ in (Figure 3.8)

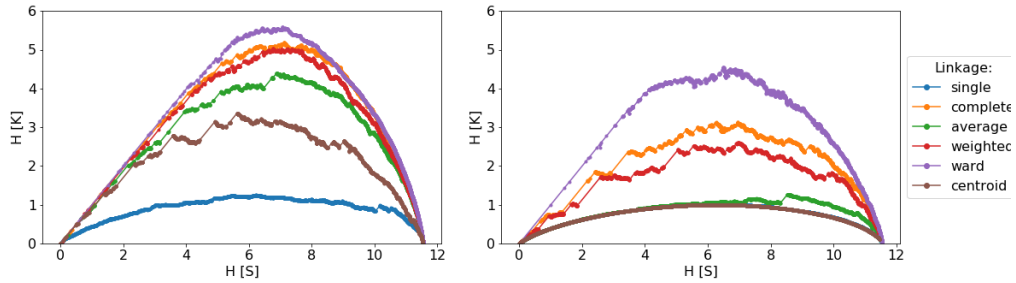

Figure 3.8 – Curves of $H[K]$,$H[S]$ for z (on the left) and $\bar{z}^*_{i,j}$ (on the right) for hierarchical agglomerative clustering with different linkage methods.

The ward linkage seem to present the best performance in terms of maximizing the area of $H[K]$,$H[S]$ for $\bar{z}^*_{i,j}$. In the solution for z, it can be seen implied that some dimensions are dominating over other in the measurement, resulting in a biased result similar to the curve for z in Figure 3.7. This can be also verified by plotting the result of HAC-ward for all dataset z, $z^*_{i,j}$, $\bar{z}_{i,j}$,and $\bar{z}^*_{i,j}$ in Figure 3.9.

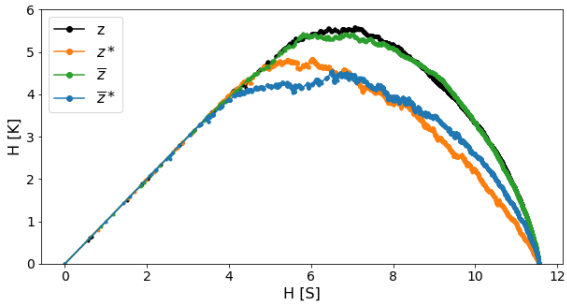

Figure 3.9 – Hierarchical agglomerative clustering with 'Ward' linkage for all z dataset transformations z,z^*

The Zeisel dataset is a relatively small file, it is only 120 Mbytes. It takes in average of 70 seconds to run the build the whole dendrogram (`linkage` method) and another 32

seconds to search for all the clusters (`fcluster` method). As the HAC has complexity $\mathcal{O}(m^3)$, it rises the concern about performance of this algorithm in a larger dataset. With that in mind, it was run a trial on a much larger dataset both in dimension and in number of samples, the 'Campbell' dataset[8] This dataset contains the expression of 26774 genes (which is the same order o magnitude of 'Zeisel') and 21086 cells of the mouse hippocampus, resulting in a file of 1.21 GBytes. The run time for trial was about 3 hours and 20 minutes, the same order of magnitude of running k-medoids for the 'Zeisel' dataset. In Figure 3.10, it can be noticed that for 'Campbell' dataset the run time of `fcluster` method is greater than `linkage` method, which must be investigated more carefully in future works.

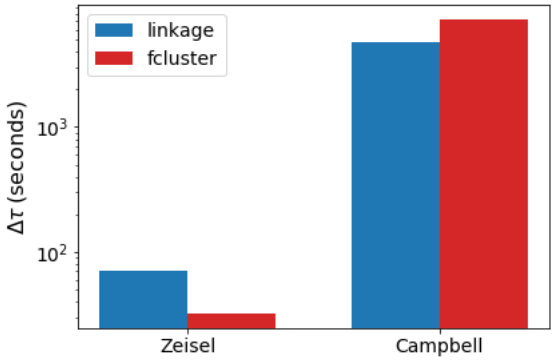

Figure 3.10 – Performance for HAC methods `linkage` and `fcluster` for 'Zeisel' and 'Campbell' datasets.

3.3 Gene expression on clusters

Before measuring $H[K]$ and $H[S]$, gene expression data must be organized according to the cluster labels produced by the clustering method. A first and naïve implementation of the code that tracks and separate the group of cells can take up to 20 minutes for each gene, being insurmountable problem in a dataset with $\approx$ 20 thousands genes. The first version of the algorithm that will be considered for optimization spent around 2 minutes run time. This still not reasonable considering the amount of genes to process, so four optimizations were implemented over this code. The first two are serial approaches and the last two use multi-thread parallelization strategy. A Intel i5-4440 CPU (3.10GHz) was used for benchmarking these implementations and average was taken over the set composed by the 'Reference', the nine knwon marker genes and more ten genes chosen randomly. Firstly, the optimization `Opt.1` takes advantage of algorithms implemented in Numpy package (cite). The second optimization `Opt.2` change the structure of the search for labels and expression $g_{i,j}$ by zipping this two arrays (the array of the genes $g_{i,j}$ values and the array of a given cluster number k) into a dictionary. The former optimization perform the genes partitioning in around 30 seconds, as for the latter performs in 15 seconds. At last, it was implemented a shared memory parallelization approach for `Opt.3`

and `Opt.4` using the `Ray` package for python, with `Opt.3` and `Opt.4` spanning 2 and 4 threads, respectively. The output of this process is the k_s, k and m_k parameters required to evaluate both resolution and relevance.

Opt. level	Average	Std. dev.
Opt. 0	125.41	1.12
Opt. 1	29.72	1.41
Opt. 2	14.40	0.96
Opt. 3	6.83	0.61
Opt. 4	4.03	0.33

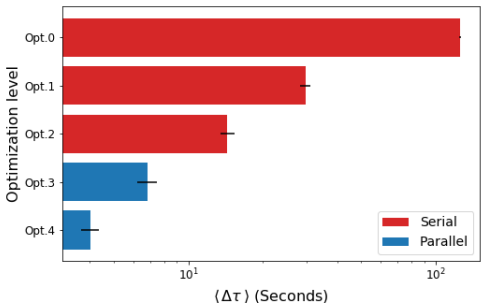

Figure 3.11 – Performance of the gene processing algorithm.

3.4 Multi-partition entropy signature

Following the concepts introduced in the previous sections, the $H[K]$,$H[S]$ curves for the 9 known marker genes can found on **??**. To compare the results for different genes, it is performed the integral (area under the curve) of the curves $H[K]$,$H[S]$ each gene. However, this measurement it ill-defined, since each gene will present a different resolution. To overcome this caveat, the curves are normalized by its maximum resolution,

$$\begin{cases} \hat{H}[S] = H[S]/H[S]_{\max} = H[S]/(\log_2 M) \\ \\ \hat{H}[K] = H[K]/H[S]_{\max} = H[K]/(\log_2 M)\,. \end{cases} \tag{3.9}$$

The result is the normalized curves in Figure 3.12.

The CVS framework when applied to neurons[12] had the relevance named 'multi-scale relevance', since the resolution and relevance were measured on different time scales. At the present approach, the cell space is being clustered multiple times with control parameter k and generating distinct partitions. Adapting the previous terminology, the area of each $H[K]$,$H[S]$ will be called multi-partition relevance (MPR). Results for MPR measurement for all genes will be presented in chapter 4.

Apart from relevance and resolution, another measurement of interest is the the mutual information between the gene expression G and the clustering structure S and it is given by

$$\begin{aligned} I[G;S] &= H[G] - H[G|S] \\ &= -\sum_s \frac{k_s}{N}\left[\frac{k_{s,g}}{k_s}\log_2\left(\frac{k_{s,g}}{k_s}\right) - \left(1-\frac{k_{s,g}}{k_s}\right)\log_2\left(1-\frac{k_{s,g}}{k_s}\right)\right], \end{aligned} \tag{3.10}$$

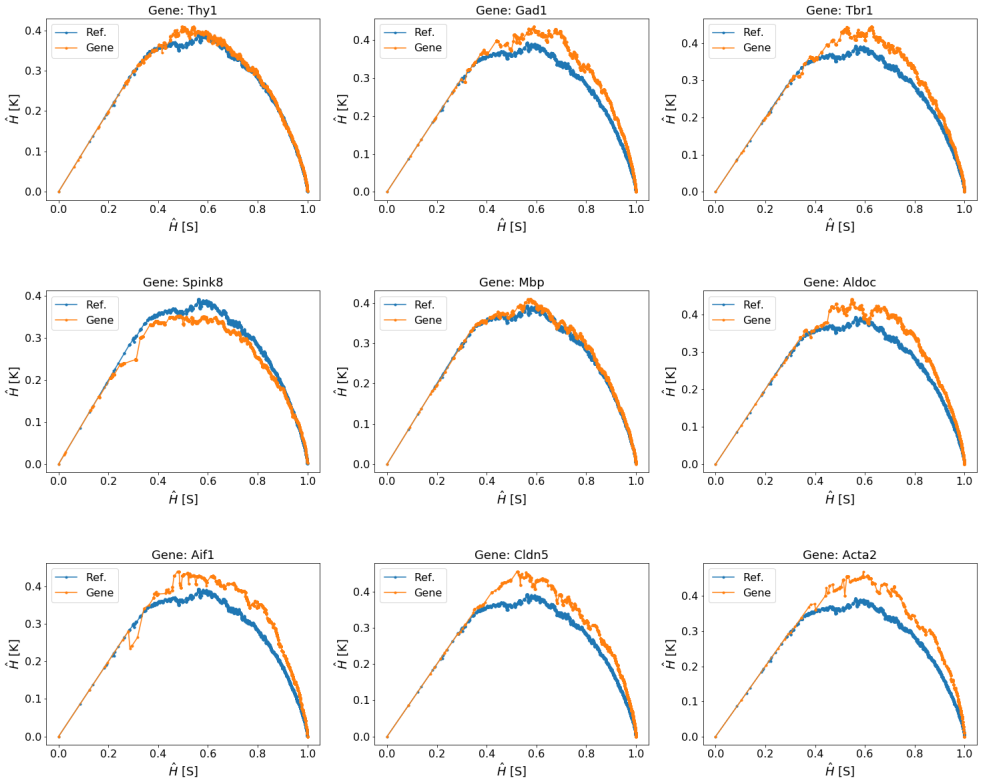

Figure 3.12 – Normalized curves of $H[K]$,$H[S]$ for the 9 known marker genes.

where k_s is the size of cluster s (total number of cells, either at state 'ON' or 'OFF'), $k_{s,g}$ is the number of times gene g is 'ON' at cluster s. Then, $k_{s,g}/k_s$ is the fraction of cells in s where gene g is 'ON' and N is the total number of cells, independent of their states. By measuring $I[G;S]$ it is possible to know how much of the underlying clustering structure each gene carries in their on cluster structure. The lower the value of $I[G;S]$, more information it contains from the clustering structure. For example, for the hypothetical reference gene 'Ref.', $I[G;S] = 0$ as expected by definition. Similar to multi-partition relevance, in the case of multi-partition mutual information shall be normalized by the maximum resolution as well (Equation 3.11),

$$\hat{I}[G;S] = I[G;S]/H[S]_{\max} = I[G;S]/(\log_2 M)\,. \tag{3.11}$$

Chapter 4

Results

In this chapter, it will be presented the results of multi partitioning relevance (MPR) and multi partition mutual information (MPMI) for all genes in Zeisel dataset. As each gene total expression counts M_g vary from one another and $\log_2 M_g$ is the maximum resolution for a given gene, $\log_2 M$ is the natural scale to project the behavior of both MPR and MPMI.

The multi partition relevance is the 'area under the curve' for the relation of normalized relevance and resolution, $\hat{H}[K]$ and $\hat{H}[S]$ presented in section 2.3. As a result, it can be seen that the variation this measurement among the genes reduces as M increases (Figure 4.1). In addition, the known marker genes follow this trend with only one exception the gene 'Spink8'. The reason for that will be explored in future work, but for now it can be noticed that Spink8 has a very low resolution and the largest average gap distance from the the clustering resolution curve Appendix A.

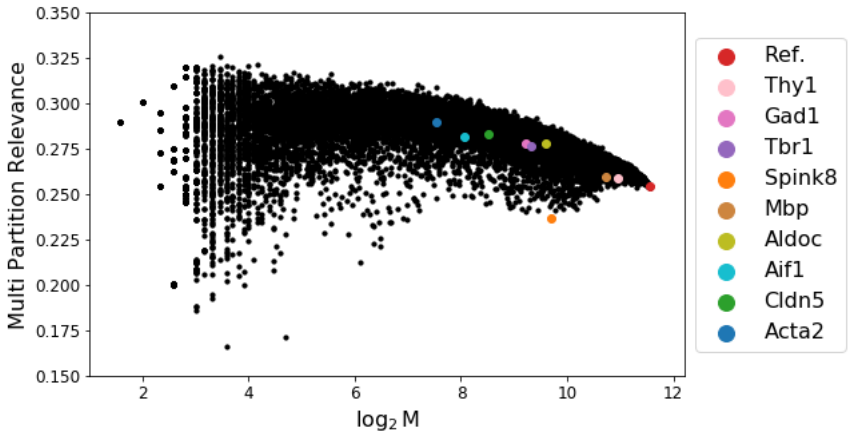

Figure 4.1 – Multi partition Relevance for all genes in the 'Zeisel' dataset.

For the multi partition mutual information, 'Spink8' gene do not follow the trend as well. Nonetheless it carries a lot of information from the clustering structure – the lower

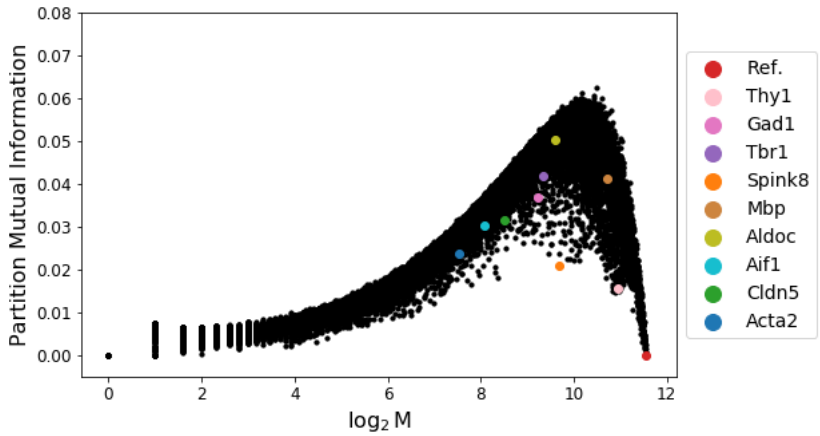

Figure 4.2 – Multi partition mutual information for all genes in the 'Zeisel' dataset.

the value of the mutual information, the highest is the cluster structure correlated to the 'Reference' gene. For example, as explained in section 3.1, gene 'Thy1' is expressed in three classes of cells that together represents more than half of the cell space size. This property is evident as 'Thy1' gene is the known marker gene that has lowest value for MPMI (Figure 4.2).

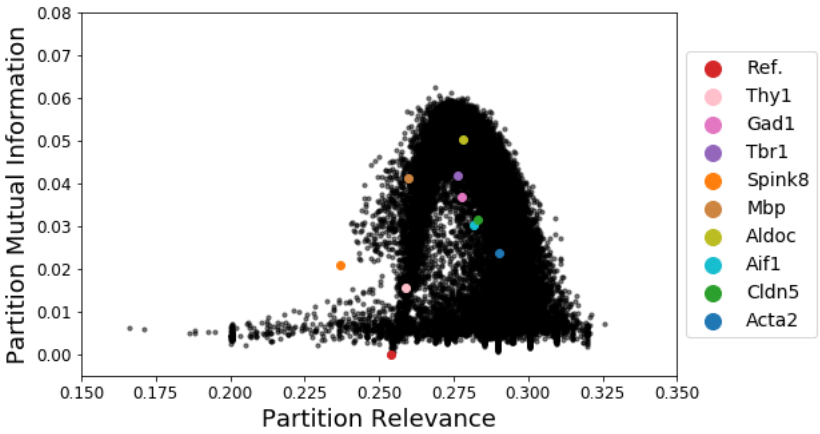

Figure 4.3 – Multi partition mutual information and multi partition relevance for all genes in the 'Zeisel' dataset.

When MPMI and MPR are plotted together, we get a cloud of points in the window $y = \{0, .25\}$, $x = \{0.02, 0.06\}$ that do not follow the main trend. The only known marker gene that dwells in this region is 'Spink8' gene. Searching on Allen Brain databases for the genes on this region, it was found that most of them are very specific marker genes (see Figure 4.4).

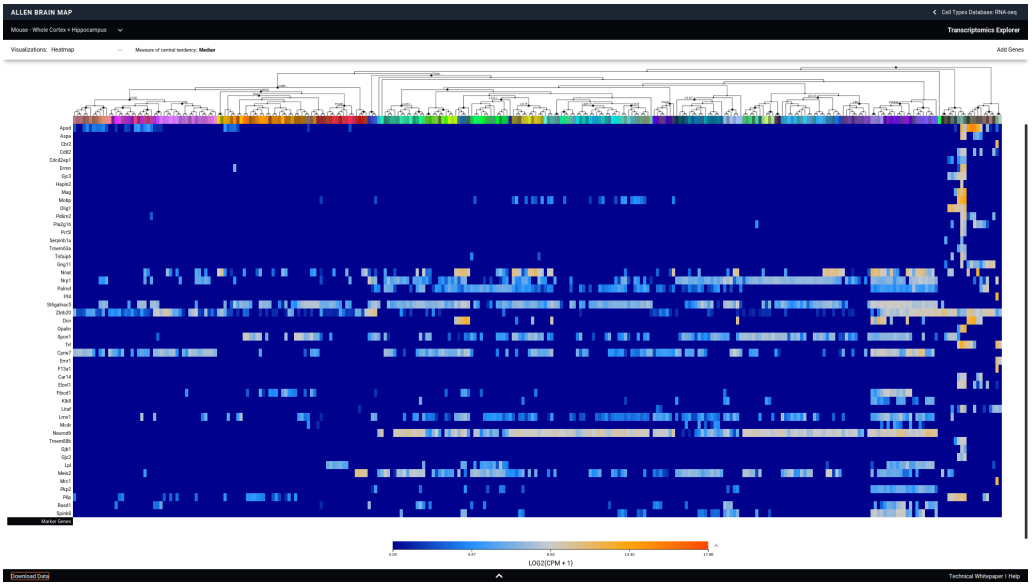

Figure 4.4 – A screenshot of expression profile of genes from Allen Brain Map.

Moreover, these genes preserve the clustering structure embedded. In collaboration with Ryan Cubero, it was done a preliminar UMAP analysis, as can be seen in Figure 4.5. The genes considered in this analysis are the ones with $\log_2 M \geq 7$. If it is true that the low MPR genes preserve the clustering structure in 'B' in Figure 4.5, this should be seen under a UMAP dimension reduction in Figure 4.5 'C'.

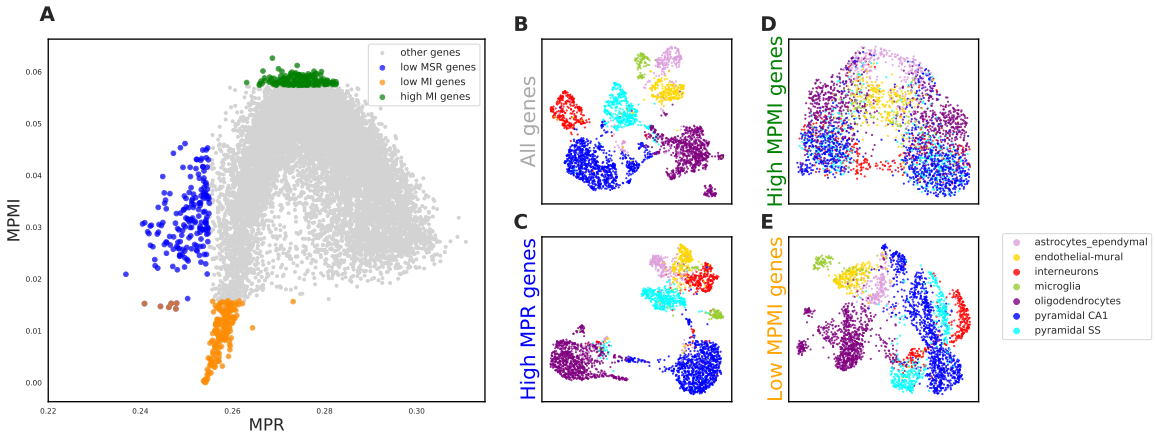

Figure 4.5 – UMAP dimension reduction for different regions of the MPR-MPMI space.

This was an incipient work both from the perspective of biology, physics and high-performance computing. There are many questions answer before further conclusions. The main question might be on the MPR curves, since in the current approach the marker genes are not those that have high MPR curves. In fact, results focus on low MPR curves, which contradicts the theory of maximally informative samples. Further research should focus larger datasets and check if the current results are consistent on them.

Appendix A

Known markers genes figures

A.1 Multi partition relevance curves

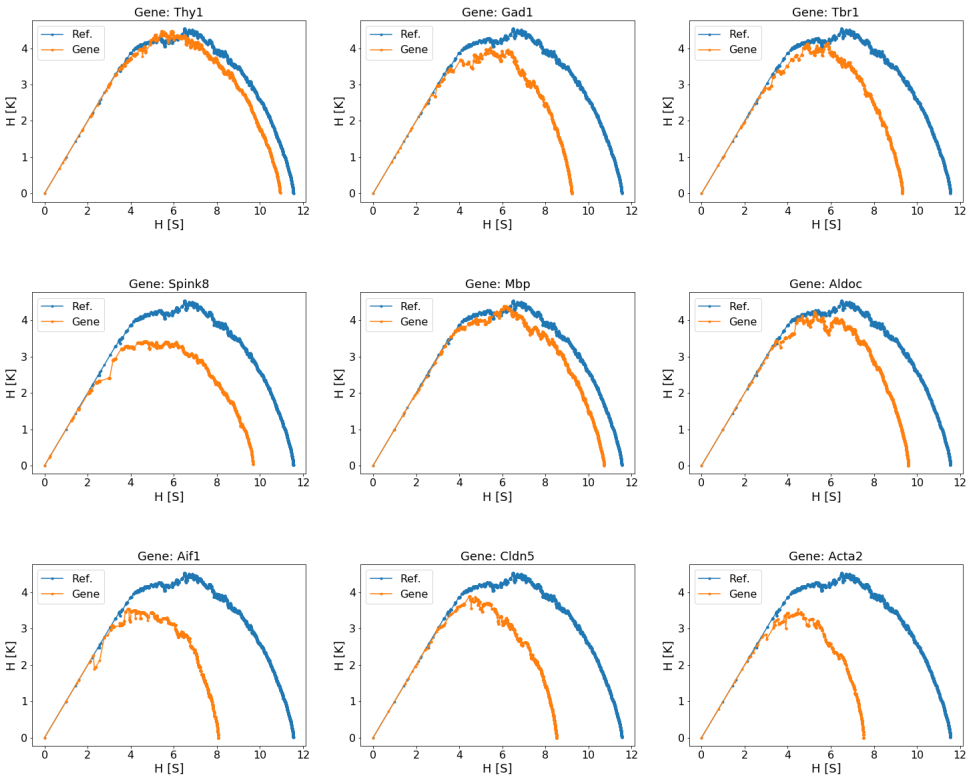

Figure A.1 – $H[K]$,$H[S]$ curves for all known marker genes.

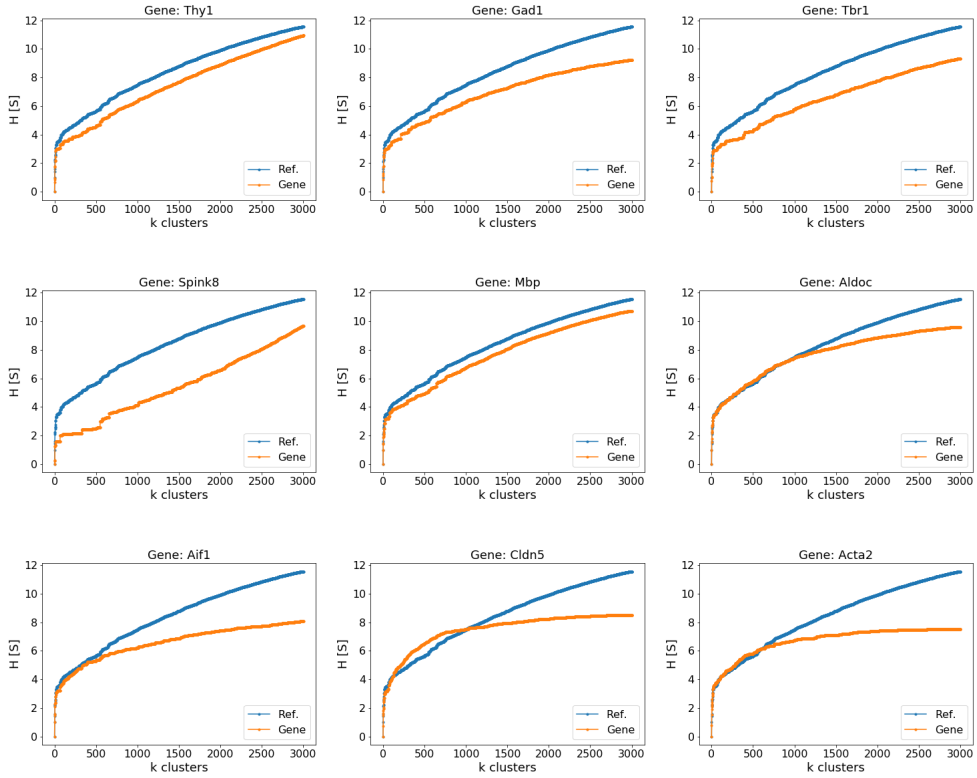

Figure A.2 – $H[S]$ curves for all known marker genes.

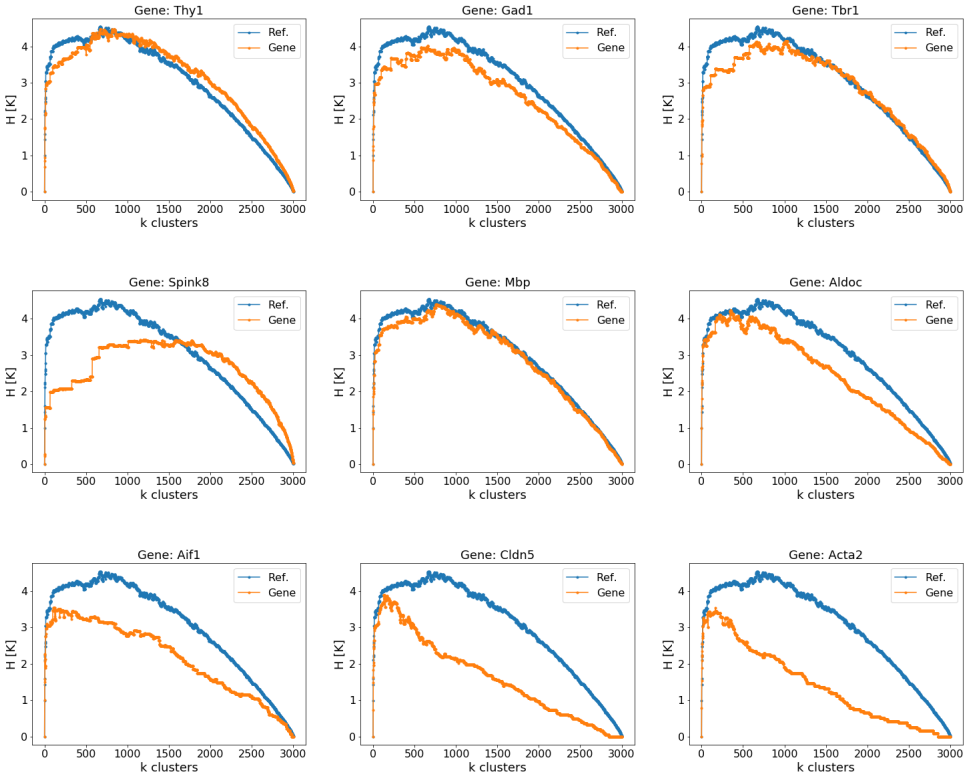

Figure A.3 – $H[K]$ curves for all known marker genes.

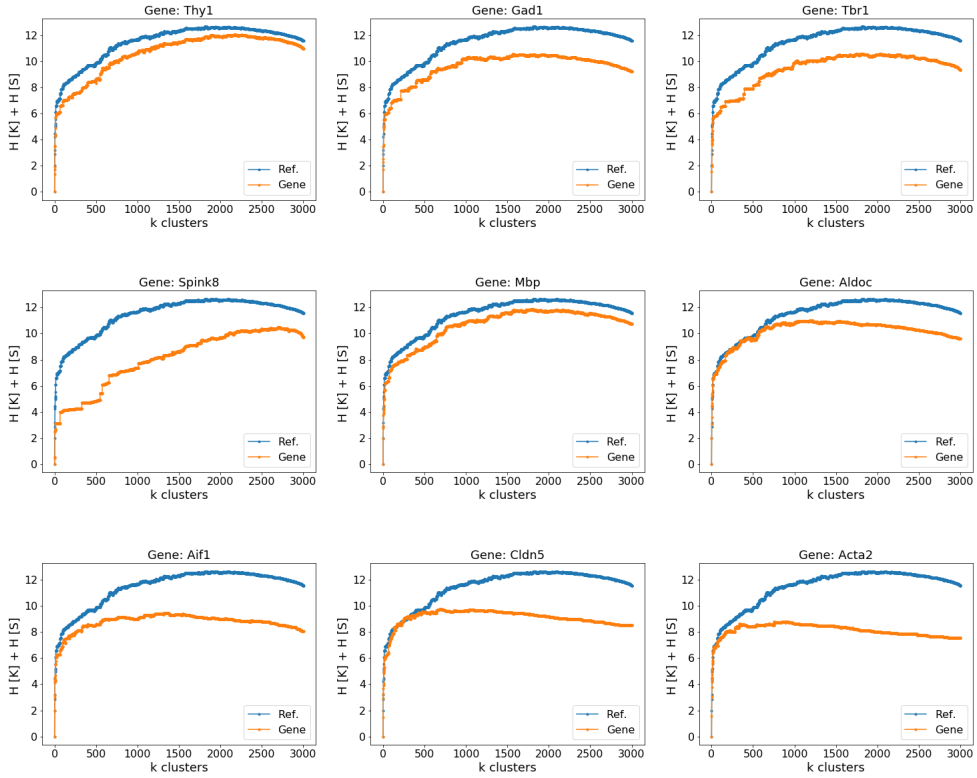

Figure A.4 – Trade-off of $H[K]$ and $H[S]$ curves for all known marker genes.

Bibliography

1 No citation in the text.

2 Aisha A AlJanahi, Mark Danielsen, and Cynthia E Dunbar. An introduction to the analysis of single-cell rna-sequencing data. *Molecular Therapy-Methods & Clinical Development*, 10:189–196, 2018. Cited on page 5.

3 David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. Cited on page 12.

4 Hemberg Group at the Sanger Institute. Repository of the quantitative models of gene expression group. Cited on page 20.

5 Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, March 2012. 2 citations in pages 12 and 13.

6 David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012. Cited on page 17.

7 Roger K Blashfield. Mixture model tests of cluster analysis: Accuracy of four agglomerative hierarchical methods. *Psychological Bulletin*, 83(3):377, 1976. Cited on page 15.

8 John N Campbell, Evan Z Macosko, Henning Fenselau, Tune H Pers, Anna Lyubetskaya, Danielle Tenen, Melissa Goldman, Anne MJ Verstegen, Jon M Resch, Steven A McCarroll, et al. A molecular census of arcuate hypothalamus and median eminence cell types. *Nature neuroscience*, 20(3):484, 2017. Cited on page 26.

9 K. Christensen and N. R. Moloney. *Complexity and Criticality*. Advanced physics texts. Imperial College Press, 2005. Cited on page 10.

10 Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970. 2 citations in pages 5 and 6.

11 Ryan John Cubero, Junghyo Jo, Matteo Marsili, Yasser Roudi, and Juyong Song. Statistical criticality arises in most informative representations. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(6):063402, 2019. Cited on page 18.

12 Ryan John Cubero, Matteo Marsili, and Yasser Roudi. Multiscale relevance and informative encoding in neuronal spike trains. *Journal of Computational Neuroscience*, pages 1–18, 2018. 2 citations in pages 18 and 27.

13 Peter Dayan, LF Abbott, et al. Theoretical neuroscience: computational and mathematical modeling of neural systems. *Journal of Cognitive Neuroscience*, 15(1):154–155, 2003. Cited on page 17.

14 Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936. Cited on page 9.

15 National Center for Biotechnology Information. Gene expression omnibus. Cited on page 20.

16 Silvia Grigolon, Silvio Franz, and Matteo Marsili. Identifying relevant positions in proteins by critical variable selection. *Molecular BioSystems*, 12(7):2147–2158, 2016. Cited on page 18.

17 Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969. Cited on page 13.

18 Ariel Haimovici and Matteo Marsili. Criticality of mostly informative samples: a bayesian model selection approach. *Journal of Statistical Mechanics: Theory and Experiment*, 2015(10):P10013, 2015. Cited on page 18.

19 J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. Cited on page 12.

20 Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract). In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, page 332–339, New York, NY, USA, 1994. Association for Computing Machinery. Cited on page 12.

21 L Kaufman, PJ Rousseeuw, and Y Dodge. Clustering by means of medoids in statistical data analysis based on the, 1987. Cited on page 13.

22 Aleksandra A Kolodziejczyk, Jong Kyoung Kim, Valentine Svensson, John C Marioni, and Sarah A Teichmann. The technology and biology of single-cell rna sequencing. *Molecular cell*, 58(4):610–620, 2015. Cited on page 5.

23 Stuart Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982. 2 citations in pages 9 and 12.

24 Malte D Luecken and Fabian J Theis. Current best practices in single-cell rna-seq analysis: a tutorial. *Molecular systems biology*, 15(6), 2019. Cited on page 21.

25 D.J.C. MacKay, D.J.C.M. Kay, and Cambridge University Press. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. 2 citations in pages 9 and 10.

26 Francesco E. Maranzana. On the location of supply points to minimize transportation costs. *IBM Systems Journal*, 2(2):129–135, 1963. Cited on page 14.

27 Matteo Marsili, Iacopo Mastromatteo, and Yasser Roudi. On sampling and modeling complex systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2013(09):P09003, 2013. Cited on page 18.

28 Thierry Mora and William Bialek. Are biological systems poised at criticality? *Journal of Statistical Physics*, 144(2):268–302, 2011. Cited on page 18.

29 H.M. Nussenzveig. *Curso de física básica: Mecânica (vol.1)*. 1. Blucher, 2013. Cited on page 11.

30 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. Cited on page 12.

31 Arpiar Saunders, Evan Z Macosko, Alec Wysoker, Melissa Goldman, Fenna M Krienen, Heather de Rivera, Elizabeth Bien, Matthew Baum, Laura Bortolin, Shuyu Wang, et al. Molecular diversity and specializations among the cells of the adult mouse brain. *Cell*, 174(4):1015–1030, 2018. Cited on page 7.

32 Erich Schubert and Peter J Rousseeuw. Faster k-medoids clustering: improving the pam, clara, and clarans algorithms. In *International Conference on Similarity Search and Applications*, pages 171–187. Springer, 2019. Cited on page 14.

33 James Sethna et al. *Statistical mechanics: entropy, order parameters, and complexity*, volume 14. Oxford University Press, 2006. Cited on page 17.

34 Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948. Cited on page 17.

35 Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015. Cited on page 6.

36 Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998. Cited on page 8.

37 Valentine Svensson, Roser Vento-Tormo, and Sarah A Teichmann. Exponential scaling of single-cell rna-seq in the past decade. *Nature protocols*, 13(4):599–604, 2018. 2 citations in pages 5 and 6.

38 Bosiljka Tasic, Boaz P Levi, and Vilas Menon. Single-cell transcriptomic characterization of vertebrate brain composition, development, and function. In *Decoding Neural Circuit Structure and Function*, pages 437–468. Springer, 2017. Cited on page 5.

39 RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018. 2 citations in pages 12 and 13.

40 Tian Tian, Ji Wan, Qi Song, and Zhi Wei. Clustering single-cell rna-seq data with a model-based deep learning approach. *Nature Machine Intelligence*, 1(4):191–198, 2019. Cited on page 7.

41 Itay Tirosh, Benjamin Izar, Sanjay M Prakadan, Marc H Wadsworth, Daniel Treacy, John J Trombetta, Asaf Rotem, Christopher Rodman, Christine Lian, George Murphy, et al. Dissecting the multicellular ecosystem of metastatic melanoma by single-cell rna-seq. *Science*, 352(6282):189–196, 2016. Cited on page 5.

42 Catalina A Vallejos, Davide Risso, Antonio Scialdone, Sandrine Dudoit, and John C Marioni. Normalizing single-cell rna sequencing data: challenges and opportunities. *Nature methods*, 14(6):565, 2017. Cited on page 6.

43 Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, page arXiv:1907.10121, Jul 2019. Cited on page 16.

44 Pingjian Yu and Wei Lin. Single-cell transcriptome study as big data. *Genomics, proteomics & bioinformatics*, 14(1):21–30, 2016. Cited on page 6.

45 Amit Zeisel, Ana B Muñoz-Manchado, Simone Codeluppi, Peter Lönnerberg, Gioele La Manno, Anna Juréus, Sueli Marques, Hermany Munguba, Liqun He, Christer Betsholtz, et al. Cell types in the mouse cortex and hippocampus revealed by single-cell rna-seq. *Science*, 347(6226):1138–1142, 2015. 2 citations in pages 6 and 19.

46 Yan-Qing Zhang and Jagath Chandana Rajapakse. *Machine learning in bioinformatics*, volume 4. Wiley Online Library, 2009. Cited on page 8.