University of Amsterdam

Masters Thesis

Using probabilistic methods for hierarchical visualization of single-cell RNA-seq data

Author:
Tobias Beers

Supervisor:
Perry Moerland

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Science

in the

Computational Science Lab
Informatics Institute

August 2020

Declaration of Authorship

I, Tobias Beers, declare that this thesis, entitled 'Using probabilistic methods for hierarchical visualization of single-cell RNA-seq data' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Amsterdam.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date: 10 July 2020

"Statistics is the grammar of science."

Karl Pearson

UNIVERSITY OF AMSTERDAM

Abstract

Faculty of Science
Informatics Institute

Master of Science in Computational Science

Using probabilistic methods for hierarchical visualization of single-cell RNA-seq data

by Tobias Beers

Single cell RNA sequencing (scRNA-seq) has progressed the biological sciences by allowing researchers to obtain a rough estimate of the number of transcripts in a cell for every gene. Modern advancements in the field of scRNA-seq have enabled the reading of thousands of genes in thousands to even millions of cells. Techniques like scRNA-seq therefore return high dimensional data-sets, which are hard to visualize in only two dimensions. The analysis and visualization of such data have been performed by linear (PCA, PPCA) and later non-linear (t-SNE, UMAP) dimensionality reduction techniques. In this report, we examine the role of hierarchical mixtures of probabilistic PCAs (HmPPCAs) for the visualization of scRNA-seq data. At the same time, we explore the use of probabilistic programming to define and solve probabilistic models easily, without the need to derive the necessary equations to solve them manually.

The HmPPCAs model was implemented in the probabilistic programming language Stan and evaluated on 2 scRNA-seq data-sets from the literature and 5 simulated data-sets of various dimensionalities. The model is solved using NUTS (an extension to Hamiltonian Monte Carlo) and ADVI (an extension to variational inference) to get a better grasp of two different methods used in probabilistic programming. We then compared the HmPPCAs results with results obtained using other popular techniques like t-SNE and UMAP. The plots obtained through all methods were tested for their cell type separability.

The results show that ADVI provides statistical modellers with a fast way to approach solutions without sacrificing much of the performance. Although HmPPCAs did not achieve the same visualization performance as UMAP and t-SNE, the results show that extending linear dimensionality reduction techniques with hierarchy can show structures that are not noticeable when using a single non-hierarchical dimensionality reduction.

Acknowledgements

I want to thank Perry for helping and understanding and dedicating so much of his time to me and this project.

I also thank Joel, for guiding me through all the negatives, and I thank Ivander for bringing all the positives. I love these people!

I thank the people who helped me for making my life easier and I thank the difficult moments in my life for teaching me how to grow.

I feel very grateful towards the plants and trees on this planet, some of which who were the best listeners and who always seemed to know the right thing to do.

I thank my mom, for listening, talking and being cool!

And I thank the universe for giving birth to this all!

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ADVI	**A**utomatic **D**ifferentiation **V**ariational **I**nference
BIC	**B**ayesian **I**nformation **C**riterion
CSL	**C**omputational **S**cience **L**ab
ELBO	**E**vidence **L**ower **BO**und
EM	**E**xpectation-**M**aximization
GMM	**G**aussian **M**ixture **M**odel
HMC	**H**amiltonian **M**onte **C**arlo
HmPPCAs	**H**ierarchical **M**o**PPCAs**
HSPC	**h**ematopoietic **s**tem and **p**rogenitor **c**ells
LT_SHC	long-**t**erm **h**ematopoietic **s**tem **c**ells
MC	**M**onte **C**arlo
ML	**M**aximum **L**ikelihood
MoPPCAs	**M**ixture **o**f **PPCAs**
NUTS	**N**o **U**-**T**urn **S**ampler
PCA	**P**rincipal **C**omponent **A**nalysis
PPCA	**P**robabilistic **PCA**
scRNA-seq	**S**ingle **C**ell **RNA** **S**equencing
t-SNE	**t**-**D**istributed **S**tochastic **N**eighbour **E**mbedding
UMAP	**U**niform **M**anifold **A**pproximation and **P**rojection
UvA	**U**niversiteit **v**an **A**msterdam
VB	**V**ariational **B**ayes
VI	**V**ariational **I**nference

Symbols

$\boldsymbol{x}$	d-dimensional observed data-point
$\boldsymbol{X}$	$n \times d$ matrix describing the observed data-set
$\boldsymbol{z}$	m-dimensional latent data-point
$\boldsymbol{Z}$	$n \times m$ matrix describing the latent data-set
$\boldsymbol{\mu}$	d-Dimensional mean of $\boldsymbol{x}$
σ	Standard deviation of $\boldsymbol{x}$
$\boldsymbol{\Sigma}$	Covariance matrix of $\boldsymbol{X}$
n	Number of data-points in data-set
d	Number of dimensions of one variable
K	Number of mixture components
π_k	Mixture coefficients

1 Introduction

Gene expression is the process where genes on the DNA are transcribed and used to create specific products. It is the process that sets the synthesis of proteins from the DNA in motion and it is fundamental to all biological mechanisms. When a gene is expressed, its DNA sequence is copied to a piece of mRNA, which can then be translated into a protein. Genes that are highly expressed are being transcribed to mRNA more often than genes that are lowly expressed, so the number of mRNA transcripts for these genes is relatively high. Given the biological and medical significance of this process, it is studied broadly and techniques for quantitative measurement have been developed over the last decades. RNA sequencing (**RNA-seq**) measures the relative level of expression of genes by quantifying the number of mRNA transcripts. This approach measures the average levels of gene expression across a population of cells, which possibly consists of different cell types. This is a limitation to many experiments because results are not specific to one individual cell. Single-cell RNA-sequencing (scRNA-seq) [1] has given the means to analyse the expression patterns of large numbers of genes from specifically chosen cells. First, a sample of specifically chosen cells is collected. The cells undergo a lysis step that permeates their membrane but leaves the RNA intact. The mRNA is then reverse-transcribed to its complementary cDNA. The newly acquired cDNA is amplified, through polymerase chain reaction (PCR) or in vitro amplification. Finally, this cDNA is sequenced and the resulting sequence reads are mapped back to the genome, pieces of cDNA that match with a gene are counted.

This technique has seen a rise in popularity [2]. ScRNA-seq was first pioneered in the late 2000's [3]. The new invention of scRNA-seq allowed researchers to study individual cell types and to compare different cell types. For example, cells from different embryonic stages might be compared to study the activity of genes during embryonic development [4]. Another example would be the detection of malignant tumor cells in otherwise healthy cell populations [5].

For the comparison of transcriptomes of different cell types, it is often desirable to visualize the gene expression levels of individual cells in a simple yet informative way. It

may not be hard to visualize data in two dimensions, for instance, one could generate a scatter-plot of two genes, where each point corresponds to one cell. Simple correlations should then easily become visible. For three dimensions or genes, a 3D plot can be generated similarly. For data of more dimensions, it is possible to produce multiple plots for every combination of two dimensions. However, scRNA-seq deals with data that may exceed thousands of genes, which is hard to visualize in a way that preserves most information. Dimensionality reduction has therefore become an important topic in scRNA-seq data analysis. Techniques like principal component analysis (**PCA**) [6], probabilistic PCA (**PPCA**) [7], t-distributed stochastic neighbour embedding (**t-SNE**) [8] and uniform manifold approximation and projection (**UMAP**) [9, 10] (among others) deal with dimensionality reduction and try to transform high-dimensional data to 2 or 3 dimensional data in the most adequate ways. For instance, PCA projects the data onto lesser dimensions, known as the principal components, in such a manner that the variance of the projected data is kept at a maximum. PPCA attempts to find latent random variables that explain the observed data in a probabilistic way. These techniques are linear. T-SNE and UMAP are non-linear techniques. These techniques put the data-points in the low dimensional space by specifying an attractive force between points that are closely related in the data space and a repellent force between other points.

Another way to induce non-linearity is by adding hierarchical layers. For example, a hierarchical mixture of PPCAs (**HmPPCAs**) [11] is able to divide the data into different clusters and returns their individual low-dimensional representation. It has been argued that each of these clusters might have different optimal representations in the low-dimensional space. While one cluster can be best described by two principal components, another cluster might be better represented by two different principal components. Representing every cluster by the same principal components might be optimal for the whole data-set, but not for the separate clusters. This model has been used for the analysis of scRNA-seq data [12]. However, this model was introduced as an interactive model, which still requires the user to select the number of different clusters and their approximate locations in the low-dimensional space. User input may be susceptible to human subjectivity and inconsistency. It also requires the user to wait for the parameter estimation to finish so that they can initiate the next level. Here, we develop a model that works in a fully automated way and does not require extra user-input.

Another difference to our model is how it is solved. Some of the mentioned techniques, like PCA and PPCA, have a closed-form solution. Alternatively, it is possible to use an Expectation-Maximization (**EM**) algorithm (see Appendix A.3), which is guaranteed to converge to a local optimum, with lower computational cost than the closed-form solution. The HmPPCAs model as proposed originally was solved using an EM-algorithm as well [11, 12]. However, another approach to find a solution to the HmPPCAs is

through probabilistic programming. Probabilistic programming entails programming a probabilistic model and having the underlying distributions of the parameters found automatically. Platforms for statistical optimization, like TensorFlow Probability [13] and Stan [14] have been released as an alternative way to approach solutions to statistical problems. These are Bayesian probabilistic programming languages. Using tools like these has the benefit that the user can specify and manipulate statistical models without having to derive and solve the necessary equations. This increases the range of models to be studied and enables the user to easily refine parts of the model without having to rewrite the entire solution. These tools do not only give the users a solution to their model, but they can also give more information about the solution as they compute (an approximation of) the entire probability function of the parameters of the model given the input data. The models created in this report will be created within the Stan environment using Python as the primary programming language. Stan is able to automate the inference of probabilistic models in two ways. The first is by sampling through the No U-Turn Sampler (**NUTS**), which is an extension to the Hamiltonian Monte Carlo method. The other is an algorithm called automatic differentiation variational inference (**ADVI**). Both these methods will be discussed in Section 2.2

Since we will mainly extend the HmPPCAs model, Section 2 starts with the theory behind PPCA and hmPPCAs models. We then continue by discussing Stan and the sampling methods used to solve our models. We then conclude this section with a description of the algorithm used for this project. Section 3 describes the setup of the model and the experiments to evaluate our model and to compare it with other models. The results of these experiments will be discussed in Section 4, and we conclude this report with a brief discussion in Section 5.

2 | Approach

In this Section, we discuss our model and the theory behind it. First, we discuss latent variable models, PPCA and HmPPCA which form the theoretical basis of our model in Section 2.1. More information on the Gaussian distribution, maximum likelihood estimation, the EM-algorithm and mixture models can be found in Appendix A. We then move on to Section 2.2, in which we explore the statistical platform Stan and how Stan is used to infer solutions about statistical models. Having discussed both the theory behind our model and our method of inference, we move on to Section 2.3, in which our hierarchical model is described in full detail.

2.1 Formulating the hierarchical mixture model

2.1.1 Latent variable models and PPCA

Suppose our data consists of d-dimensional vectors $\boldsymbol{x}_i \in \mathbb{R}^d$. Now we could treat these variables as separate identities, but it is well possible that the values of some of those random variables are directly influenced by one or more latent variables. For instance, we might measure levels of expression of d genes. It is then possible that a latent variable affects the expression levels of multiple genes. Since it is hard to visualize more than 3 dimensions (the expression of more than 3 genes in this example), it would be ideal to visualise the underlying latent variables instead, as they exhibit fewer dimensions. One technique to achieve this is the probabilistic PCA or PPCA.

Suppose the latent variables are m-dimensional vectors $\boldsymbol{z}_i$ of latent variables which are drawn from a zero-centered Gaussian distribution $\mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I}_m)$, with $\boldsymbol{I}_m$ being the $m \times m$ identity matrix and $\boldsymbol{0}$ the zero-vector of length m. In our model, $\boldsymbol{x}$ is given by

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu} + \epsilon \tag{2.1}$$

Here, $\boldsymbol{\mu}$ is a d-dimensional vector describing the mean. ϵ describes an d-dimensional noise term from a Gaussian distribution $\mathcal{N}(\epsilon|\mathbf{0}, \sigma^2\boldsymbol{I_d})$. $\boldsymbol{W}$ is a $d \times m$ matrix mapping $\boldsymbol{z}$ to $\boldsymbol{x}$ and its entries are known as the factor loadings. This means that $\boldsymbol{x}|\boldsymbol{z}$ follows the distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2\boldsymbol{I})$. The likelihood function of a single data-point $\boldsymbol{x}_i$ is described by $\int p(\boldsymbol{x}_i|\boldsymbol{z}_i, \boldsymbol{W}, \boldsymbol{\mu}, \boldsymbol{\sigma^2})p(\boldsymbol{z}_i)d\boldsymbol{z}_i$. For a data-set consisting of n vectors $\boldsymbol{x}_i$, we can take the product of the likelihood of all individual $\boldsymbol{x}_i$ to find the likelihood of the whole data-set.

Now suppose we want to find a matrix $\boldsymbol{Z}$ describing the latent data based on the matrix of observed data $\boldsymbol{X}$, along with the maximum likelihood estimates of $\boldsymbol{\mu}$, σ^2 and $\boldsymbol{W}$. A closed-form solution has been derived by Tipping and Bishop [7]. There is also an iterative expectation-maximization (**EM**) algorithm to solve this problem (see Appendix A.3 for an elaboration on the EM algorithm).

For the E-step, we are interested in the log likelihood of the complete data-set (including latent variables). The joint probability of an observed and latent data-point i is $p(\boldsymbol{x}_i, \boldsymbol{z}_i) = p(\boldsymbol{x}_i|\boldsymbol{z}_i)p(\boldsymbol{z}_i)$. The complete-data likelihood is then given by $p(\boldsymbol{X}, \boldsymbol{Z}) = \prod_{i=1}^{n} p(\boldsymbol{x}_i|\boldsymbol{z}_i)p(\boldsymbol{z}_i)$. If we translate this into the expected complete-data log likelihood, we end up with

$$\begin{aligned} E[\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{W}, \sigma^2)] = \\ \sum_{i=1}^{n} \ln \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \boldsymbol{\sigma^2}\boldsymbol{I}) + \sum_{i=1}^{n} \ln \mathcal{N}(\boldsymbol{z}_i|\mathbf{0}, \boldsymbol{I}) = \\ -\sum_{i=1}^{n} \Big\{ \frac{d}{2} \ln 2\pi\sigma^2 + \frac{1}{2}\mathrm{Tr}(E[\boldsymbol{z}_i\boldsymbol{z}_i{}^T]) + \frac{1}{2\sigma^2}||\boldsymbol{x}_i - \boldsymbol{\mu}||^2 - \frac{1}{\sigma^2}E[\boldsymbol{z}_i]^T\boldsymbol{W}^T(\boldsymbol{x}_i - \boldsymbol{\mu}) \\ + \frac{1}{2\sigma^2}\mathrm{Tr}(E[\boldsymbol{z}_i\boldsymbol{z}_i{}^T]\boldsymbol{W}^T\boldsymbol{W}) + \frac{m}{2}\ln 2\pi \Big\}, \end{aligned} \tag{2.2}$$

where the trace (Tr) of a matrix is defined as the sum of its diagonal elements.

We would also like to know what the expected value is for every latent variable $\boldsymbol{z}_i$. For this, it is best to look at the differences between each data-point $\boldsymbol{x}_i$ and the mean of all data-points $\overline{\boldsymbol{x}}$, so that the effect of $\boldsymbol{\mu}$ is taken into account. We then use the transformation matrix $\boldsymbol{W}$ and the matrix $\boldsymbol{M} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} + \sigma^2\boldsymbol{I}$ to get to the expected values of $\boldsymbol{z}_i$.

$$E[\boldsymbol{z}_i] = \boldsymbol{M}^{-1}\boldsymbol{W}^T(\boldsymbol{x}_i - \overline{\boldsymbol{x}}) \tag{2.3}$$

$$E[\boldsymbol{z}_i \boldsymbol{z}_i{}^T] = \sigma^2 \boldsymbol{M}^{-1} + E[\boldsymbol{z}_i] E[\boldsymbol{z}_i]^T \tag{2.4}$$

For the M-step, we differentiate equation 2.2 with respect to its parameters to find the maximum likelihood values for these parameters. We already know that $\boldsymbol{\mu}_{ML}$ is given by the sample mean of $\boldsymbol{X}$. $\boldsymbol{W}$ can be obtained by 2.5 and then σ^2 can be obtained by 2.6 using the newly obtained values for $\boldsymbol{W}$.

$$\boldsymbol{W} = \left[\sum_{i=1}^{n} (\boldsymbol{x}_i - \overline{\boldsymbol{x}}) E[\boldsymbol{z}_i]^T \right] \left[\sum_{i=1}^{n} E[\boldsymbol{z}_i \boldsymbol{z}_i{}^T] \right]^{-1} \tag{2.5}$$

$$\sigma^2 = \frac{1}{dn} \sum_{i=1}^{n} ||\boldsymbol{x}_i - \overline{\boldsymbol{x}}||^2 - 2E[\boldsymbol{z}_i]^T \boldsymbol{W}^T (\boldsymbol{x}_i - \overline{\boldsymbol{x}}) + \mathrm{Tr}(E[\boldsymbol{z}_i \boldsymbol{z}_i{}^T] \boldsymbol{W}^T \boldsymbol{W}) \tag{2.6}$$

For more details on the derivation of these equations, we wish to refer to Bishop (2006) [15].

2.1.2 The hierarchical mixture of PPCAs

For our purposes, we need to discuss one more model that combines the previous knowledge: a mixture of PPCAs (**MoPPCAs**) [16]. Assume we have different groups of cell types which exhibit different patterns of gene expression. Then not only can we find different mixture components in our data, each mixture component may also be explained by a particular underlying latent data-set of lower dimensionality. This gives us the distribution $p(\boldsymbol{x}_i|\boldsymbol{\mu}, \boldsymbol{\sigma}^2, \boldsymbol{\pi}, \boldsymbol{W}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{W}_k, \boldsymbol{\mu}_k, \sigma_k^2)$, where each mixture component k is a latent variable model with its own mean $\boldsymbol{\mu}_k$, standard deviation σ_k and factor loadings matrix $\boldsymbol{W}_k$.

We can solve such a model, similarly to the Gaussian mixture model (**GMM**) (see Appendix A.3), with an EM algorithm. A key ingredient of a MoPPCAs model is knowing which data-points belong to which mixture component. Therefore, we need to compute the probabilities γ_i^k of each data-point $\boldsymbol{x}_i$ being generated by each mixture component k first. We refer to this term γ_i^k as the *responsibility* term. Using Bayes' theorem, we derive γ_i^k as follows.

$$\gamma_i^k = \frac{\pi_k p(\boldsymbol{x}_i)}{\sum_{j=1}^{K} \pi_j p(\boldsymbol{x}_i)} = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_i|\theta_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_i|\theta_j)} \tag{2.7}$$

With this, we can specify the complete log-likelihood.

$$\ln p(\boldsymbol{X}, \boldsymbol{Z}|\theta) = \sum_{i=1}^{N}\sum_{j=1}^{K} \gamma_i^k \ln \pi_k \mathcal{N}(\boldsymbol{x}_i|\theta_k)\mathcal{N}(\boldsymbol{z}_i|\boldsymbol{0}, \boldsymbol{I}) \tag{2.8}$$

Note that we have explicitly defined $\boldsymbol{z}$ to come from a standard normal distribution. Setting the derivatives with respect to the individual parameters to 0 gives us the new estimates for the parameters.

This method can be extended to a hierarchical mixture of PPCAs (**hmPPCAs**) [11]. We have discussed a model where data was generated by different components, each with their own latent space. In a hmPPCAs, each of the components may be composed of even more underlying mixture components. Where a MoPPCAs divides the top-level data into clusters and finds a representation of each cluster in a latent space, a hmPPCAs evaluates whether sub-clusters can be found within the second-level clusters. Figure 2.1 demonstrates an example of this, where the data can first be separated in a red and a yellow-and-blue cluster. On the next level, the yellow-and-blue cluster is further separated into two yellow and blue sub-clusters. Logically, these sub-clusters could be divided even further by adding more levels of depth to the model. The probability density function describing a hmPPCAs with two levels is:

$$p(\boldsymbol{x}) = \sum_{i=1}^{K}\sum_{j\in\mathcal{G}_i} \pi_i \pi_{j|i} \mathcal{N}(\boldsymbol{x}|\theta_j) \tag{2.9}$$

In this case, $\mathcal{G}_i$ is the set of second-level sub-clusters j that are contained within top-level cluster i. The complete-data log-likelihood function of this model is similar to that of a MoPPCAs model.

$$\sum_{i=1}^{n}\sum_{j=1}^{K} \gamma_i^j \sum_{m=1}^{|\mathcal{G}_i|} \gamma_i^{m|j} \ln \pi_{m|j} p(\boldsymbol{x}_i, \boldsymbol{z}_i) \tag{2.10}$$

2.2 Stan, NUTS and ADVI

All models described in this report have not been fitted analytically or by use of the EM algorithm as described above. Instead, all statistical models have been programmed in Stan so that they could be solved numerically. Stan is a platform for statistical modelling. An example of a PPCA model as defined in Stan is given in Appendix C. Given a statistical model, Stan tries to find the posterior distribution of all unknown

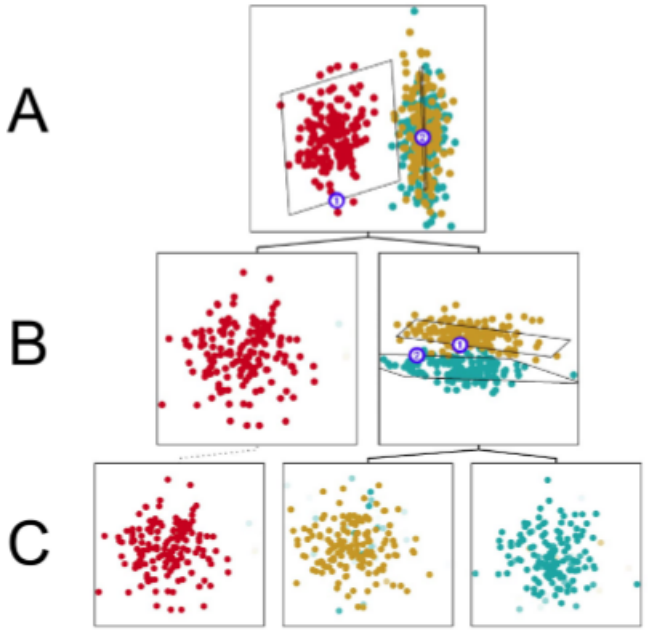


Figure 2.1: Example of a hierarchical cluster structure. The top-level shows the complete data-set in its latent space (A). The data is then clustered into a red and a yellow-and-blue cluster, and the latent space of each of those clusters is given (B). Doing so reveals structure within the yellow-and-blue cluster that was not visible before. This allows the cluster to be divided even further into separate yellow and blue sub-clusters (C). This figure has been copied from Bishop & Tipping (1998) [11].

'parameters'. These parameters might be parameters in the classical sense, e.g. the mean and standard deviation of a Gaussian given the observed data. Alternatively, they might be missing variables, e.g. the latent variables explaining the observed variables. Using Stan has several advantages. It allows us to specify a model, or just its log-likelihood function, and then Stan attempts to find the optimal solution without requiring to derive the equations to find the maximum-likelihood solution to the model. This is especially useful when manipulating a statistical model, as we can adjust the model without needing to rewrite the (algorithmic) solution. Secondly, Stan makes use of samplers which not only give a point estimate for the parameter values but the entire posterior distribution of each unknown parameter. This gives us information about the uncertainty of the found solution. Stan implements two main algorithms for this. Most notably, Stan makes use of the No U-turn sampler (NUTS) (discussed in Section 2.2.1). This is an extension of Hamiltonian Monte Carlo sampling. NUTS has the benefit over Hamiltonian Monte Carlo that it automatically determines the number of steps needed for convergence, saving computation time. The other, more recently implemented algorithm to determine the posterior of the parameters, is the automatic differentiation variational inference (ADVI) algorithm (see Section 2.2.2). This is an algorithm based upon variational inference. It selects a distribution from a variational family that is

as similar as possible to the posterior. This might be less exact, but it saves a lot of computational cost without affecting the accuracy of the outcome too much. These algorithms were both utilized and compared in this project.

Stan does not stand alone, but rather, it needs to be implemented in another environment. For this project, programming was done by use of Python, in which Stan was implemented through a package called PyStan [17]. The general workflow is as follows. Data is read in through the Python environment and a statistical model is specified in Stan. Then the model is compiled. Using the data as input, Stan is used to infer the parameters and outputs them to the Python environment. This data is then finally analyzed using Python.

2.2.1 Monte Carlo sampling and NUTS

Monte Carlo (**MC**) methods are a branch of sampling methods. They are forms of Markov chains, and generally entail a random trajectory through the sample space, directed by the underlying probability distribution of interest. An easy and well-known example is the Metropolis-Hastings algorithm [18, 19], where a fictive particle jumps randomly to new positions within the sample space. Based on the ratio of the probabilities of the old and new position, a new position can be accepted as a sample. If it is rejected, a jump to a new position is made. The samples drawn using this algorithm are guaranteed to converge to the underlying distribution when sampling long enough.

A drawback of this algorithm is that the number of samples necessary for convergence is sub-optimal, as the random walks through the sample space are inefficient. For this reason, other algorithms, such as Hamiltonian Monte Carlo (**HMC**), were proposed [20]. When using HMC, the fictive particle behaves as in a Hamiltonian dynamics simulation. A particle has a position in the sample space and a velocity. For a pre-determined number of steps of fixed step size, the particle moves in a random manner through the sample space, at the end of which the position is either accepted or rejected as a sample. This algorithm produces much more efficient random walks and is therefore less costly computationally. However, this algorithm has its own drawbacks. HMC requires manually setting a step-size parameter ϵ and a number of steps L. Sub-optimal values for these parameters can result in unnecessary computational cost and inaccurate results. Particularly, setting L too high will result in the particle slowly returning to its original position, which requires not only unnecessary computation but also leads to slower convergence. Tuning these parameters usually requires some degree of expertise and a few tuning runs of the algorithm. Some adaptations to HMC have been proposed to automatically tune ϵ [21], but not L.

The No-U-Turn Sampler [22] (**NUTS**) is an extension to HMC. NUTS is most notable for its ability to determine the optimal number of steps L while running. When a particle moves through the sample space to generate a sample, the number of steps needs to be high enough for a decent random walk to happen, but not too high as the fictive particle will eventually loop back and move back to its original position. The point where the fictive particle is the furthest away from its initial position is characterized by sharp turns in the path of the particle. Stopping at these 'U-turns' is where the algorithm has drawn its name from. Additionally, NUTS tunes ϵ automatically whilst running the algorithm.

NUTS has added a feature that checks when the sampling trajectory is at its maximum length and then uses this criterion to stop. The concrete criterion depends on the derivative of half the squared distance from the original to the new sample. If this measure is close to zero, the squared (and thus the absolute) distance between the two samples is not changing anymore and the trajectory is probably at its point of return. The series of samples is desired to be a Markov chain. One necessary property of such Markov chains is known as time-reversibility, i.e. the validity of the series of samples is not dependent on whether the samples are read from front to back or back to front. To maintain the time-reversibility of the Markov Chain, the trajectory is run both forward and backward in time. The algorithm builds a 'tree' to examine the structure of the trajectory and the algorithm halts with a probability that is negatively proportional to the derivative of half the squared distance at either end of the trajectory. Secondly, NUTS tunes ϵ on-the-fly, based on a method named 'dual averaging' [21]. This adaptation has been slightly improved to include more parameters. Apart from adapting ϵ while running, NUTS does a quick search for a reasonable value of ϵ to initialize the algorithm with.

2.2.2 Variational inference and ADVI

Variational inference (**VI**) is a method of approximating a posterior probability distribution $p(\boldsymbol{\theta}|\boldsymbol{x})$ for the parameters $\boldsymbol{\theta}$ given data $\boldsymbol{x}$ that is difficult to determine. It works by chossing a familiy of probability distributions $q(\boldsymbol{\zeta})$ and then minimizing the dissimilarity between P and Q. To measure this dissimilarity, the Kullback-Leibler divergence of P and Q ($D_{KL}(Q||P)$) is used. This is an asymmetric similarity measure defined as $D_{KL}(Q||P) = \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\boldsymbol{x})}$. This can be rewritten as $D_{KL}(Q||P) = \log p(\boldsymbol{x}) - \mathcal{L}(Q)$, where $\mathcal{L}(Q) = -E_{\boldsymbol{\theta}}[\log q(\boldsymbol{\theta}) - \log p(\boldsymbol{\theta}, \boldsymbol{x})]$. The first term $\log p(\boldsymbol{x})$ is known as the evidence. The last term $\mathcal{L}(Q)$ is the evidence lower bound or **ELBO**. Its name is derived from the fact that the evidence is the sum of the ELBO and the KL divergence; since the KL divergence is non-negative, the ELBO is the lower bound for the evidence. Often, finding the optimal distribution $q(\boldsymbol{\zeta})$ though KL-divergence minimization is difficult

or impossible. However, the evidence is a function that does not rely on $q(\boldsymbol{x})$ and is therefore kept constant as $q(\boldsymbol{x})$ is optimized. Minimizing the KL divergence is therefore equivalent to maximizing the ELBO, as is done in VI.

ADVI [23] is an algorithm that implements VI and automates the process. Traditionally, performing a VI algorithm required that the researcher had specified a VI algorithm beforehand, which included the selection of a family of variational distributions, the objective function to optimize and the computation of its gradients. This was not only an effort to the researcher, but it was also an obstacle to the automatization of the process. ADVI differs in that it transforms the given model to be suitable to a premade VI algorithm so that the aforementioned steps become obsolete. ADVI starts off by transforming the probability density function of the posterior. ADVI's VI algorithm assumes that all parameters live in the real space and are unconstrained, which is what is achieved by this transformation. After this transformation, ADVI's premade VI algorithm can be used for ELBO maximization.

A family of variational approximations is expressed as Gaussians, either with a diagonal covariance matrix (mean field variational inference) or with a full covariance matrix (full-rank variational inference). The latter is more precise but also computationally more costly. The objective function to be optimized is expressed as the expectation of the distribution, which can easily be derived by sampling from the distribution and taking the empirical mean.

The gradient of the logarithm of the joint probability distribution of the parameters and the data with respect to the parameters ($\nabla_\theta \log q(\boldsymbol{x}, \boldsymbol{\theta})$) is computed through automatic differentiation [24]. Automatic differentiation is a technique to determine the gradient of a function which has been implemented in Stan (and was one of the reasons that Stan was chosen as a suitable environment for ADVI). Using the expectation of the gradient of the log joint probability, it is possible to obtain a noisy estimate of the gradients of the ELBO. The noisy gradient of the function is then used to optimize the variational distribution q by stochastic gradient ascent. Finally, the ELBO has been maximized and a (local) optimum for the posterior distribution of the parameters has been found. This sequence of steps is integrated in PyStan under the name Variational Bayes (**VB**) and will be referred to under this name in this report.

2.3 Detailed description of the model

2.3.1 The model

Our visualization model is a hierarchical one, based on an earlier model by Bishop & Tipping [11] (see Figure 2.1). It visualizes data, and if group structures are present, it attempts to find groups within the data on the top-level, and then possibly sub-groups within the groups at the deeper levels. The model presented in this report is mainly used for visualization of data. Therefore, the model was built to find the latent space of the data on every level, so that the data could be represented in its latent dimensions. The first latent data-space, on top-level, can easily be obtained with a normal PPCA model. The result gives the projection of the data on the first two principal components. Next, a MoPPCAs model is applied to the data, which attempts to find clusters and gives their projection onto their first two principal components. From this point on, the model repeatedly applies MoPPCAs to every cluster of data for the desired number of levels. For every level, the algorithm evaluates the clusters found on the level above. First, every cluster is tested for how many sub-clusters it possibly contains. If the cluster contains two or more sub-clusters, a MoPPCAs is initialized. If the cluster contains only one sub-cluster, i.e. the cluster cannot be divided any further into sub-clusters, its branch is marked as fully analyzed. Once the MoPPCAs returns the results, we have the new latent spaces of the cluster of interest. The MoPPCAs gives us also the probabilities of all the data-points belonging to the sub-cluster γ_i^k (see equation 2.7). When all clusters have been divided into their sub-clusters, the next level is initiated, where every cluster of the last level will be attempted to split into its sub-clusters. The researcher has the possibility to select using NUTS or VB for fitting the PPCA and MoPPCAs models before initializing the model.

Note that, even though every MoPPCAs model returns the latent spaces of the found sub-clusters, the full-dimensional data is used as input to these models. The found latent data is not reused as input for later MoPPCAs models. The latent space is only used for visualization purposes (Section 3.3.1) and for estimation of the number of sub-clusters (Section 2.3.5).

Clusters that are found to be fully analyzed do not undergo further evaluation on deeper levels. Instead, the latent data-set is copied over to the next level directly. When all clusters are fully analyzed or when the maximum depth of levels as specified by the

researcher is reached, the model stops. A summary in pseudocode for this model is given in Algorithm 1.

Algorithm 1: Pseudocode of HmPPCAs model

Data: X

Result: $\boldsymbol{Z}$ and γ_i^j for all data-points $\boldsymbol{x}_i$ and clusters j

initialize $level := 0, max\_depth = 5, max\_tries = 3$;

Initialize top-level PPCA and plot latent projection;

Make every data-point part of one cluster c with $\gamma_i^c = 1.0$;

while $level < max\_depth$ **do**

  Increment $level := level + 1$;

  for *every cluster c found in last level* **do**

    Find initial clustering with GMM in latent space;

    if *there are no sub-clusters* **then**

      skip to next cluster

    while *No MoPPCAs solution has been found* **do**

      Set $tries := 0$;

      while $tries < max\_tries$ **do**

        Increment $tries := tries + 1$;

        Initialize weighted MoPPCAs model on full-dimensional data-set with weights γ_i^c;

        if *MoPPCAs find model with same number of clusters as initial GMM* **then**

          Break loop

      if *No MoPPCAs fit is found with same number of clusters as GMM* **then**

        Break loop and use MoPPCAs fit with highest number of clusters as initialization for new MoPPCAs model

    for *every sub-cluster sc found with MoPPCAs model* **do**

      Extract probabilties γ_i^{sc};

      $\gamma_i^{sc} := \gamma_i^{sc}\gamma_i^c$;

      Plot found latent data-set with ink density proportional to γ_i^{sc};

2.3.2 Drawing a sample

When NUTS is used, the user receives the samples that were generated in the process. This excludes the first warm-up samples and every other sample is thrown away as well to reduce the correlation between samples, but the other samples are returned to the user. With the settings used in this project, this left 300 samples for the user. When

VB has found its optimal posterior distribution, a sample of 1000 points is taken from the distribution. In either case, the mean of all generated data-points is taken as the expected value of the unknown parameters. When the latent data-set or other parameters are computed, then these expected values were used.

2.3.3 Initialization, priors and constraints for the MoPPCAS models

The MoPPCAs model was highly dependent on the initialization. This is mainly due to local optima in (soft) clustering methods. The first trials of the MoPPCAs models were performed with Stans default random initialization of parameters. This would often lead to high computation times and implausible solutions. A common result was that the algorithm would find values of π_k that were near zero for one or more clusters. This meant that no data-points or only a few outliers were determined to be part of that cluster, which made them practically absent. Part of the reason that this happened is that a normal distribution with a very low spread can be fit to a cluster containing only one data-point with a very high likelihood. The spread of the remaining cluster(s) would be set high enough to encompass the remaining data-points. To prevent this behaviour, the initial clustering was performed by a GMM. GMMs were fitted in the latent space while varying the number of components. The optimal number of components was determined using the Bayesian information criterion (**BIC**; see Section 2.3.5) and was used as a reference for the MoPPCAs clusters. After the GMM clustering completed, only the most probable cluster for each data-point was noted, not the probabilities of belonging to each cluster. The values of μ_k and σ_k were inspired by the values of the centers and the spread of the clusters as found by the GMM. Because the GMM clustering was performed in the latent space, the labeling of the data-points to their clusters had to be applied to the same data-points in the full-dimensional space first. This way, the center and covariance matrix of each cluster could be estimated in the full-dimensional space. Basing σ^2 on the covariance matrix of the clusters is not entirely correct, because the covariance matrix of the PPCA model is $\boldsymbol{W}\boldsymbol{W}^T + \sigma^2\boldsymbol{I}$. Directing the value of σ^2 to values based on the found covariance matrix would actually overestimate σ^2, which in turn would lead to underestimation of $\boldsymbol{W}$. However, the resulting inference of the model would be off without doing so. Therefore, after trying various alternatives, the following three steps were taken to direct σ^2 to the spread of the clusters as found by the GMM.

First of all, the centers and the spread of the clusters that were found by the GMM were used to initialize parameters μ_k and σ_k in the MoPPCAs model. Values for π_k were computed from the GMM solution and used as initial values for π_k similarly. The solution of the GMM was also used to initialize γ_i^k for all values as 1 (when i belonged to k according to the GMM) or 0 (otherwise). It was then evaluated whether the problem

where few to no data-points were assigned to a cluster was avoided. The problem was solved for low-dimensional data-sets, but the problem persisted when more complex data-sets were analyzed. Another reason for this initialization was that a random initialization would lead to random labeling of clusters. This would lead to inconsistencies in cluster labels between samples and affect the outcome negatively. A consistent initialization meant also that the label-swapping problem during clustering was solved.

Secondly, a prior was set over μ_k and σ_k, to direct the parameters towards the correct solution. For μ_k, a prior was specified in the form of a normal distribution that was fit to the points that the GMM had labeled to cluster k in the full-dimensional space. As a prior for the standard deviation of the clusters, an inverse gamma distribution, fit with help of the GMM solution, was specified at first, but this seemed to worsen the results. A normal distribution was then specified as a prior as this yielded better results. Since σ_k is a single value related to the spread in all dimensions within that cluster, the mean of the diagonal of the covariance matrix of a cluster was used as the expectation of σ_k, thus $E[\sigma_k] = \frac{1}{d}Tr(\Sigma_k^{GMM})$ with Σ_k^{GMM} being the covariance matrix of GMM cluster k. This approach saved computational cost, but it did not necessarily prevent the problem discussed above.

Since the problem of empty clusters resulted in clusters with a minimal or large spread, constraints were set on the standard deviations of the MoPPCAs clusters in the data space. Therefore, a lower bound on σ_k was set of 0.75 times the standard deviation of the GMM cluster in the dimension with the lowest spread. This did not prevent the problem, as there could still be clusters with near-zero π_k, while other clusters took over their data-points by increasing their σ_k. Therefore, an upper bound was also set on the value of σ_k. This upper bound was set to be 1.25 times the standard deviation of the GMM cluster in the dimension with the largest spread. Stan does not accept different constraints on the entries of one vector and our lower and upper bounds were different for every cluster. As a solution, *raw* vectors of $\boldsymbol{\mu}_k$ and σ_k were evaluated as normalized vectors with entries between 0 and 1. Now, they all had the same upper and lower bounds (1 and 0). These normalized values were then later in the model transformed to their real values by multiplying them with the difference between the upper and lower bound values and adding the lower bound values to them.

Using all these constraints put the initial GMM solution largely in charge of the clustering part of the MoPPCAs solution, leaving the MoPPCAs model to make only minor adjustments to the clustering of the full-dimensional space. Despite this, it still happened occasionally that 'empty' clusters were found and that the MoPPCAs model found a different optimal clustering with fewer clusters. This was not necessarily a problem on the side of the MoPPCAs model, because overestimation of the number of clusters also

happened from time to time (see Section 2.3.5). If this happened three times in a row, the MoPPCAs solution with the highest found number of clusters was accepted. However, because this model was initialized with a prior directed at a different solution, a second MoPPCAs model was initialized with the newly found number of clusters. All initial values were copied from the first MoPPCAs model, and priors and constraints on the model were taken from the first MoPPCAs solution in the same way that they were taken from the initial GMM for the first MoPPCAs model.

2.3.4 Dealing with responsibility terms on the deeper levels

When analyzing a cluster, every data-point i has a responsibility term γ_i^k, describing the probability of belonging to cluster k. Therefore, all data-points are input to the MoPPCAs, even the ones that probably don't belong in this cluster, albeit with a very low weight. These numbers are passed into the MoPPCAs to be used in the expected complete-data log-likelihood of equation 2.10. For the first level, $\gamma_i^k = 1$ for all data-points, meaning that every data-point starts as being part of the same cluster with complete certainty. When we move to deeper levels, the likelihood equation changes from 2.10 to 2.11, where l sums over the sub-clusters at the next level.

$$\sum_{i=1}^{n}\sum_{j=1}^{K_1}\gamma_i^j\sum_{l=1}^{|\mathcal{G}_j|}\gamma_i^{l|j}\sum_{m=1}^{|\mathcal{G}_l|}\gamma_i^{m|j,l}\ln\pi_{m|j,l}p(\boldsymbol{x}_i,\boldsymbol{z}_i) \tag{2.11}$$

Noting that $\gamma_i^j\gamma_i^{l|j} = \gamma_i^{j,l}$ we can rewrite 2.11 back to 2.10, where now j denotes the sub-clusters on the next level. We can, therefore, reuse the same MoPPCAs model, but this time inputting the probabilities of belonging to the new sub-clusters as γ_i^j. Every MoPPCAs returns $\gamma_i^{l|j}$. Seeing as how $\gamma_i^j\gamma_i^{l|j} = \gamma_i^{j,l}$, we just have to multiply this number with the probability of belonging to the cluster on the level above to obtain the probability of belonging to the new sub-cluster, which will be the input responsibility term for the next level. Therefore, the model initializes a vector of ones of length n as the responsibility terms, inputs it to the MoPPCAs model and then multiplies it element-wisely with the output responsibility terms to obtain the new responsibility terms used for the MoPPCAs for the new sub-clusters on the next level.

2.3.5 BIC score to assess the number of clusters

The number of clusters to be found was estimated using the Bayesian information criterion (**BIC**). The data was clustered by a GMM with various values for the number of clusters. For the GMM models, a ready-to-use GMM package 'GaussianMixture' from

the 'Sklearn' package was used, as it worked faster than Stan and we needed only a simple estimate for initialization, not the whole posterior of all parameters. Data-points were assigned to the plot with their highest responsibility term and only the points assigned to a plot were included in the GMM. These GMMs were estimated on the latent data. Although the full-dimensional data may contain more information, it also contains more noise and we found that better estimates were therefore obtained from using the latent data. The maximum number of clusters to be found, K_{max}, was set relatively low to $K_{max} = 3$. This lead to the need for multiple levels in our models, to get a better grasp of the hierarchical working of the model. The minimum number of clusters to be found K_{min} was always set to 1, except for the top-level, where it was set to 2 to stimulate the model to make at least one division in the data-set. For each value of K, three GMM models were fit to the data, although users can choose to fit as many GMM models as they like. Pseudocode for this process is given in algorithm 2. Note that Z in this algorithm contains not all data-points in the latent space, but only the ones that were assigned to the plot which is analysed.

The BIC of a model fit is defined as $BIC = k \ln n - 2 \ln \mathcal{L}$, where n is the number of data-points, k the number of clusters and $\mathcal{L}$ the likelihood. The likelihood of a GMM model is defined in Appendix A.3. For each model, the BIC was computed, and the model with the lowest BIC was chosen as the best suitable model for the data.

Even though the BIC score usually gave a very good estimate for the number of sub-clusters, it happened from time to time that the MoPPCAs found fewer clusters. In these cases, the responsibilities were all approaching 0 for every data-point for that cluster. If a sub-cluster j was not assigned any data-point i (i.e. $\text{argmin}_k \gamma_i^k) \neq j \ \forall i$), the MoPPCAs was retried for up to three times. If the suggested number of sub-clusters was still not found by the third time, a new MoPPCAs model was initialized that looked for fewer sub-clusters. After all, it is possible for the GMM to over-estimate the number of sub-clusters. Under-estimation of the number of sub-clusters is not a problem since the found

sub-clusters will be analyzed further at deeper levels.

Algorithm 2: Determining the number of clusters

Data: Z, K_min, K_max, n_ref

Result: GMM fit on Z with optimal number of clusters

Initialize $k = K\_min$;

while $k \leq K\_max$ **do**

  set $ref := 0$;

  while $ref < n\_ref$ **do**

    increment $ref := ref + 1$;

    Cluster Z into k clusters using GMM;

    Compute likelihood L of the resulting GMM fit;

    Compute the BIC as $BIC := k \ln n - 2 \ln L$;

Return GMM fit with lowest BIC;

3 Methods

3.1 Data description

Two types of data-sets were used. The first type consisted of simulated data-sets generated using Splatter [25]. This is an R-package that allows the user to generate customized scRNA-seq data-sets. Since Splatter data-sets are easily obtainable and customizable, two different types of data-sets were generated: one simple type which contained five groups of cells that were easy to differentiate from each other and one complex type containing six groups of cells which showed more overlap in their patterns. The difference in complexity of these data-sets was regulated by use of the 'de.facLoc', 'de.facScale' and the 'de.prob' parameters. These parameters describe how easily diffent groups can be separated. The last parameter 'de.prob', the probability with which genes in a group were differentially expressed, was set higher for the complex data-sets than for the simple data-sets, as the different groups of cell-types were not distinguishable without doing so. For the simple Splatter data-sets, de.prob was set higher for the data-sets containing fewer genes, to make sure that each data-set contained enough differentially expressed genes. For the complex Splatter data-sets, this was not necessary, as de.prob was high enough already for even the data-set containing 5 genes. For both the simple and the complex types of data-sets, data-sets of 5, 25, 50, 150 and 250 genes were created.

The other type of data-sets, were the scRNA-seq data-sets published by Darmanis *et al.* (2015)[1] [26] and Nestorowa *et al.* (2016)[2] [27]. They have respectively measured the expression of 22088 and 4774 genes across 466 and 1656 cells. The Darmanis samples were retrieved from adult and fetal human brain tissue and the samples from the Nestorowa data-set were obtained from mouse hematopoietic stem and progenitor cells. The Darmanis data-set was labeled by ten different cell types: groups of oligodendrocyte precursor cells, oligodendrocytes, astrocytes, microglia, neurons, endothelial cells, fetal replicating cells, fetal quiescent cells, a hybrid group of cells and unannotated cells. The Nestorowa

[1] retrieved from https://hemberg-lab.github.io/scRNA.seq.datasets/human/brain/
[2] retrieved from http://blood.stemcells.cam.ac.uk/single_cell_atlas.html

Table 3.1: Properties of the gene expression data-sets.

	Genes	Cells	cell types	de.prob	de.facLoc	de.facScale
Simple Splatter	5	500	5	0.5	3	0
	25	500	5	0.1	3	0
	50	500	5	0.05	3	0
	150	500	5	0.017	3	0
	250	500	5	0.01	3	0
Complex Splatter	5	750	6	0.5	0.1	0.4
	25	750	6	0.5	0.1	0.4
	50	750	6	0.5	0.1	0.4
	150	750	6	0.5	0.1	0.4
	250	750	6	0.5	0.1	0.4
Darmanis	500 (out of 22088)	466	9	-	-	-
Nestorowa	500 (out of 4774)	1656	3	-	-	-

data-set was labeled with three different cell types: progenitor cells, hematopoietic stem and progenitor cells (**HSPC**) and long-term hematopoietic stem cells (**LT_SHC**). The Darmanis data-set was transformed by taking $\log(\boldsymbol{X}+1)$. The Nestorowa data-set was used as is as the authors had normalised the gene counts themselves according to the method described by Lun *et al.* (2016) [28]. To save computational cost, these data-sets were filtered to include only the 500 genes that showed the largest variance. All data-sets are described in Table 3.1.

3.2 Programming environment

For this project, programming was done by use of Python (Python 3.8.2-1), with the statistical models specified in Stan using the PyStan package (Pystan 2.19.1.1) [17] on a machine running Arch Linux 5.5.13. Stan models were compiled using the GCC compiler (GCC 9.3.0-1).

3.3 Experiment set-up

All data-sets were evaluated using the HmPPCAs model, using both NUTS and VB. The parameter settings were kept constant for all data-sets. These parameter settings are given in Table 3.2.

The performance of the model was evaluated on all data-sets described in section 3.1. Performance, as well as the computation time, was measured both when using NUTS and VB (section 3.3.2). Performance was also measured when using UMAP [9] and t-SNE [8] as a baseline. For this, the UMAP package (UMAP 0.4) [9] and the 'sklearn' package

Table 3.2: Parameter settings of the HmPPCAs model.

Parameter	Comments	value
General		
M	Number of latent dimensions	2
max_depth	Maximum number of levels before termination	5
min_clus_size	If a cluster contains less than this number of data-points, it is not divided into further sub-clusters anymore	10
n_try	Number of trials to find a MoPPCAs fit with the adequate number of clusters	3
k_max	Maximum number of sub-clusters per cluster	3
NUTS		
iterations	Number of iterations	300
chains	Number of Markov chains running simultaneously	1
warmup	Number of steps used for step-size adaptation but not for inference	150
thin	Number of samples skipped between saved samples, to reduce the correlation between successive samples	1
ADVI		
iterations	Number of iterations	10000
algorithm	Mean field or Full-rank	Mean field
Visualization		
vis_theshold	Points that are part of a cluster with a probability lower than this are not plotted	0.05

These settings were maintained for all data-sets.

'TSNE' were used. All parameters were left at their defaults, except for the perplexity parameter for t-SNE. Multiple values for the perplexity were tried, a value of 30.0 was found to yield the best results for the Splatter data-sets, a perplexity value of 50.0 was used when analyzing the Darmanis and Nestorowa data-sets. The performance at the top-level was also recorded to represent a standard PPCA method.

3.3.1 Visualization

After every level, including the top-level which initializes a PPCA on the complete data-set, the latent data is gathered for every group. For every level, the latent data can then be plotted for every cluster. All data-points are plotted in every plot, but the responsibility terms determine the density of the ink with which the points are plotted. This has the effect that most data-points are plotted only in the plot that portrays the latent space of the cluster they belong to, with the occasional data-point being vaguely represented in multiple plots that the data-point might belong to with lesser certainty.

3.3.2 Performance measures

Because the main purpose of the model was to help with visualization, the plots are being evaluated in terms of how well cell types could be separated. In the optimal case, the model produces plots in which the different cell-types are clearly separated from each other into individual clusters, possibly creating even better plots on deeper levels. To evaluate the visual separability of the cell-types, a multinomial logistic regression model was used. The logistic regression from the 'sklearn' package 'LogisticRegression' was used for this and 'lbfgs' was used as its optimization algorithm. For every plot on every level, a logistic regression was performed within a 5-fold cross-validation scheme. Technically, all points appear in every plot with a non-zero probability, but the points were only used for the logistic regression in the plot that they appeared in with the highest probability. In practice, most points appear with a probability of almost 1 in one plot and with a probability close to 0 in the other plots, so the effect of this categorization of data-points into plots is minimal. The data-points were always divided into five folds of approximately equal summed responsibility terms. The accuracy was then measured from these results. The final accuracy taken as evaluation for the individual plot was the average of the accuracies of the five test-sets. The accuracy for the whole level was given by the weighted accuracy of all the plots within that level, where the ratio of data-points in a plot was used as its weight. When reporting the accuracy of the model on an entire data-set, the level with the highest accuracy was used as a reference. This was usually the deepest level.

Apart from the visualization evaluation, the time it took to perform each analysis was also recorded. Whenever a model in Stan would be initialized or finished, the time was noted, so that the difference between the finishing time and start time could be reported as the computational time of a model. This included the individual times of the top-level PPCA. UMAP and t-SNE were left out of this comparison, as these techniques were used to create a single visualization result, so this would be an unfair comparison with inference techniques that produced the entire posterior distribution of all parameters. Because the computation time to fit a complete HmPPCAs model is highly dependent on the number of levels of the model and the number of mixture components that the data-set consists of, the computation time of the individual MoPPCAs models within the HmPPCAs was measured rather than the time it took to solve the whole HmPPCAs model. The number of mixture components that a MoPPCAs model was trying to find was noted to see if it had an effect on the computation time. Sometimes the number of mixture components that a MoPPCAs model had found was lower than the number of mixture components that it was trying to find. For these cases, both the number of

mixture components the model was looking for as suggested by the initial GMM (Section 2.3.5) and the number of mixture components as found by the MoPPCAs were noted.

4 Results

In this section, we will discuss the results found during the experiments. As mentioned in Section 3.3.2, two performance measures were taken into account. First, we will report the accuracy scores as found by the visualization assay. This will be a measure of how good a dimensionality reduction technique was at portraying the data in two dimensions while preserving most information. Secondly, the time it took to perform these analyses are reported. Here we will evaluate the time-wise computational cost and scalability of NUTS and VB and compare them with each other.

4.1 Visualization performance

We start with the results of the visualization assay. A logistic regression was performed on each latent data-set plot to predict cell-types in the plots, where the accuracy of this result was noted. This was done using a 5-fold cross-validation scheme. The average accuracy of the five folds was taken as the accuracy of a plot. In the case of the HmPPCAs models, the accuracy was computed for each level. Since the levels contained multiple plots, the accuracy of one level was computed as the weighted average of each plot within that level, where the weights were determined by the number of data-points that were contained within each plot. When the overall accuracies of HmPPCAs models are reported, the accuracy of the best performing level is given.

All results found are given in Table 4.1. The simple Splatter data-sets were converted to their top-level latent spaces first by PPCA. Figure 4.1 shows the results on the simple Splatter 250 genes data-set when using NUTS and Figure 4.2 shows the results when using VB. The figures for the data-sets containing fewer genes are given in Appendix D.1. We see that reasonable accuracy scores (> 0.88) were achieved on the visualization assay on the top-level latent data. When more levels of depth were added by use of the HmPPCAs, these accuracy scores almost always rose to 1.0.

Table 4.1: Accuracy of multinomial logistic regressions on the latent data-sets found by each model in a 5-fold cross-validation scheme. The highest accuracy for a particular data-set is indicated in bold.

	Splatter simple					Splatter complex					Darmanis	Nestorowa
genes	5	25	50	150	250	5	25	50	150	250	500	500
PPCA (NUTS)	**1.00**	0.88	0.99	0.97	0.94	0.80	0.68	0.82	0.81	0.77	0.60	0.73
HmPPCAs (NUTS)	**1.00**	**1.00**	**1.00**	0.97	**1.00**	0.90	0.82	0.89	0.82	0.77	0.70	0.79
PPCA (VB)	**1.00**	0.88	0.99	0.96	0.94	0.80	0.69	0.81	0.82	0.76	0.73	0.73
HmPPCAs (VB)	**1.00**	**1.00**	**1.00**	0.98	**1.00**	0.90	0.91	0.90	0.94	0.79	0.73	0.79
UMAP	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.95	**0.98**	0.98	**1.00**	**1.00**	**0.82**	**0.82**
t-SNE	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.95**	0.94	**1.00**	**1.00**	**1.00**	0.76	0.80

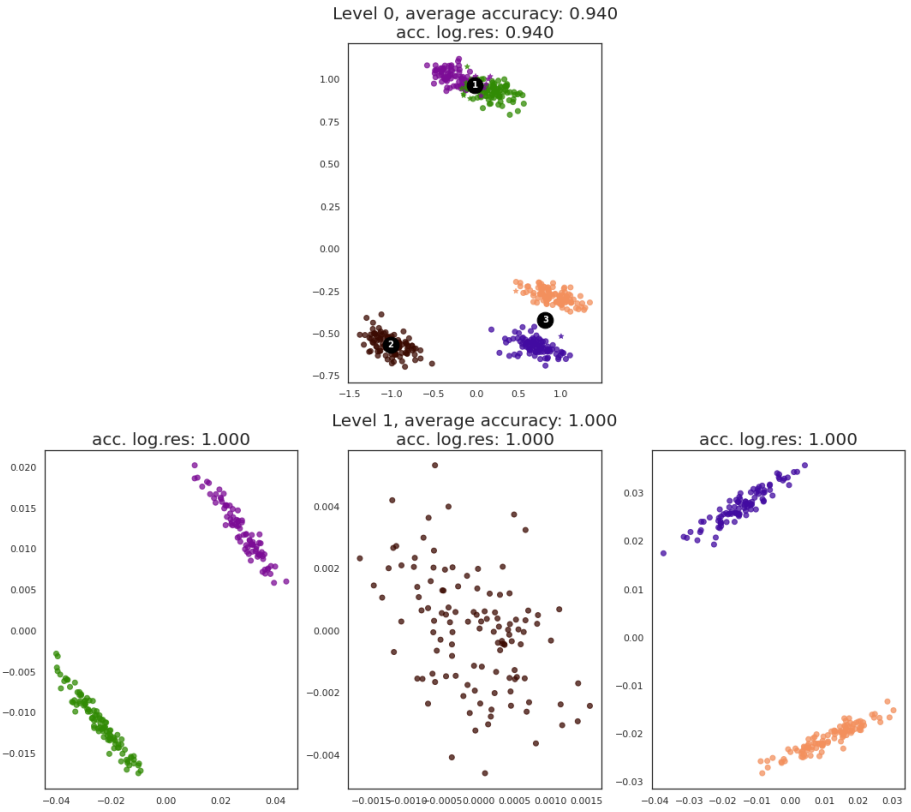


Figure 4.1: HmPPPCAs model performed on the simple Splatter data-set with 250 genes using NUTS. The maximum accuracy of 1.0 was found at level 1. The top-level PPCA achieved an accuracy of 0.940. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

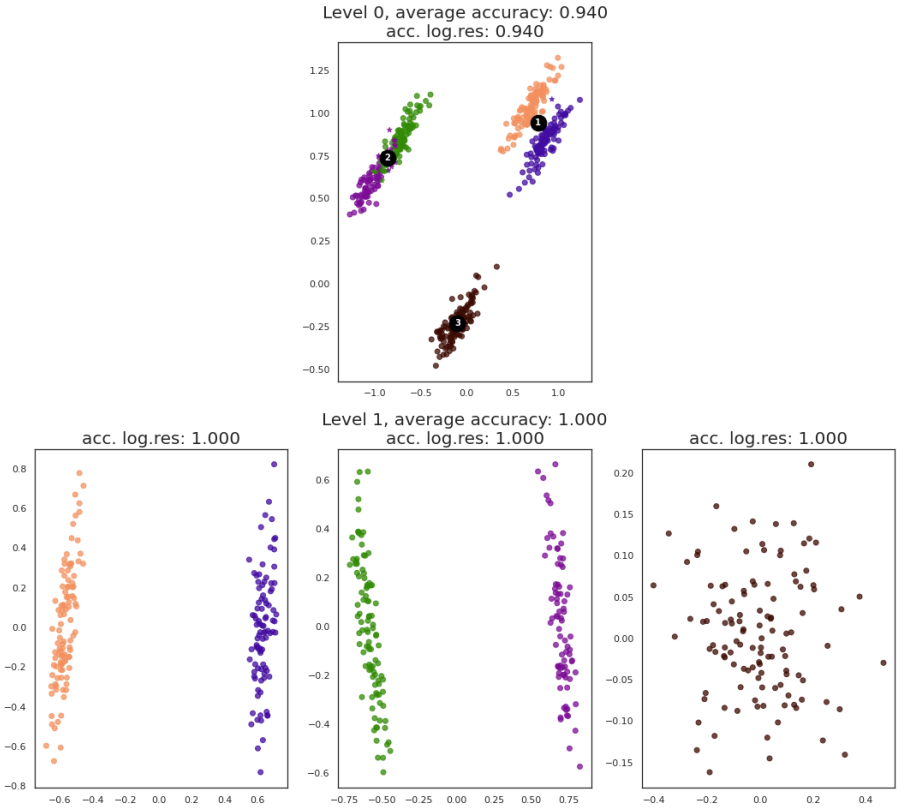


Figure 4.2: HmPPPCAs model performed on the simple Splatter data-set with 250 genes using VB. The maximum accuracy of 1.0 was found at level 1. The top-level PPCA achieved an accuracy of 0.940. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

When evaluating the complex Splatter data-sets, lower and more varying accuracies were observed. The results when analysing the 250 genes complex data-set are shown in Figure 4.3 (NUTS) and Figure 4.4 (VB). The results of the lower dimensional data-sets are given in Appendix D.2. When comparing PPCA and HmPPCAs, we see that the addition of more levels increased the accuracy in almost every case, although neither was able to achieve perfect accuracy scores on the complex data-sets. When we compare NUTS and VB, we see that their results are very similar when performing the PPCA. Occasionally, the scores of NUTS and VB differed, but the difference was always negligible. In the case of the HmPPCAs, however, larger differences between NUTS and VB were observed. VB performed better than or at least as good as NUTS in all cases when comparing the HmPPCAs results.

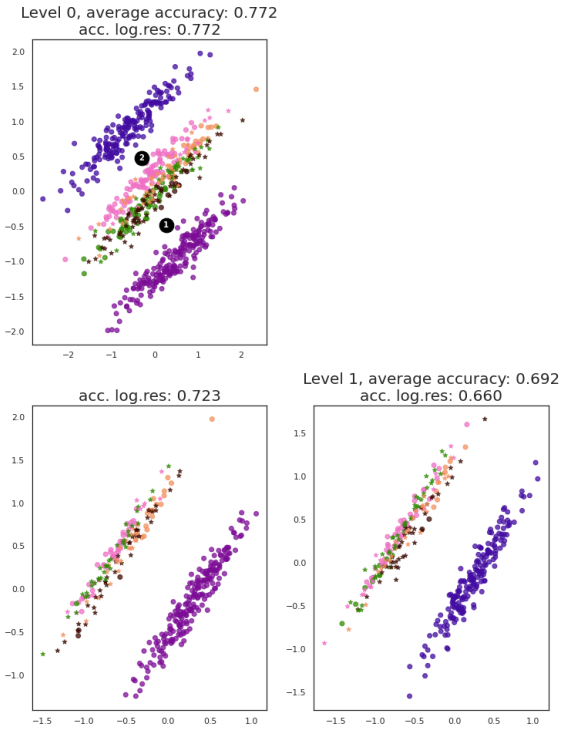


Figure 4.3: HmPPPCAs model performed on the complex Splatter data-set with 250 genes using NUTS. A maximum accuracy of 0.772 was found at level 0. Performance did not improve after the top-level PPCA. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

Both methods achieved lower scores on the Darmanis and Nestorowa data-sets than on the Splatter data-sets. Figures 4.5 and 4.6 show the result of the HmPPCAs on the Darmanis data-set when using NUTS and VB, respectively. Figures 4.7 and 4.8 are the NUTS and VB HmPPCAs results when analyzing the Nestorowa data-set. The HSPC cell types in this data-set showed multiple clusters. When using NUTS, the HmPPCAs separated each of those clusters. The progenitor and LT.HSC cells were difficult to distinguish from each other. When using VB, fewer levels were initialized and therefore the data was separated less. Since a higher accuracy was achieved using NUTS on level 2, it might have been the case that NUTS was able to obtain a slightly better visualization which led to further clustering. Again, HmPPCAs scored higher than PPCA on the Nestorowa data-sets with both NUTS and VB and on the Darmanis data-set when using NUTS. The addition of more levels did not improve the accuracy on the Darmanis data-set when using VB. VB yielded better results than NUTS on the Darmanis data-set, but both methods performed exactly equally on the Nestorowa data-set.

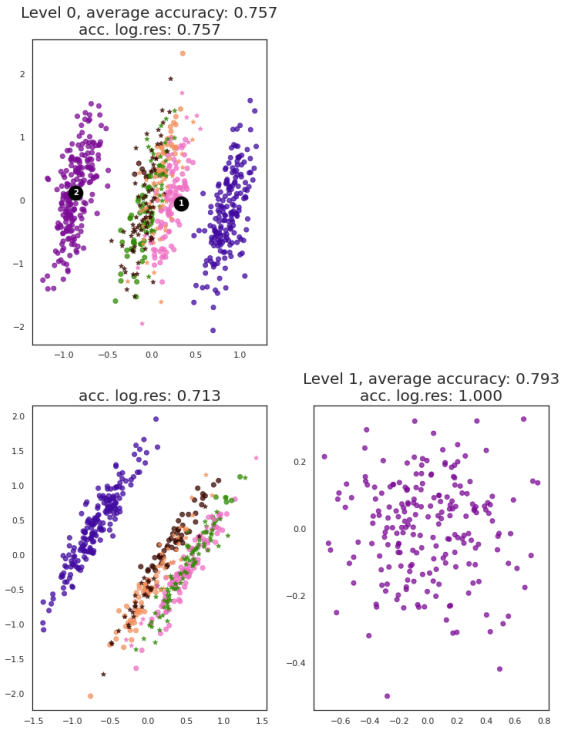


Figure 4.4: HmPPPCAs model performed on the complex Splatter data-set with 250 genes using VB. A maximum accuracy of 0.793 was found after one level. The top-level PPCA achieved an accuracy of 0.757. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

The UMAP and t-SNE baselines also achieved perfect accuracies of 1.0 on the simple splatter data-sets. Slightly lower accuracies were achieved on the complex Splatter data-sets of low dimensionality, but this accuracy rose back to 1.0 for the complex Splatter data-sets that contained at least 150 genes. UMAP and t-SNE performed almost exactly equally good on the 5 genes complex data-sets, with t-SNE outperforming UMAP barely by a negligible difference. UMAP scores slightly higher on the 25 genes complex data-set and t-SNE on the 50 genes complex data-set. Together UMAP and t-SNE achieved the highest accuracy scores on the Splatter data-sets. The resulting visualizations of UMAP and t-SNE are given in Appendix D.3.

The result when using UMAP on the Darmanis data-set is given in Figure D.18, and the t-SNE result is given in Figure D.19. Both of these methods outperformed the HmPPCAs. UMAP achieved the highest accuracy for the Nestorowa data-set, of which the results are visualized in Figure 4.7 (NUTS) and Figure 4.8 (VB). In this case, however, t-SNE

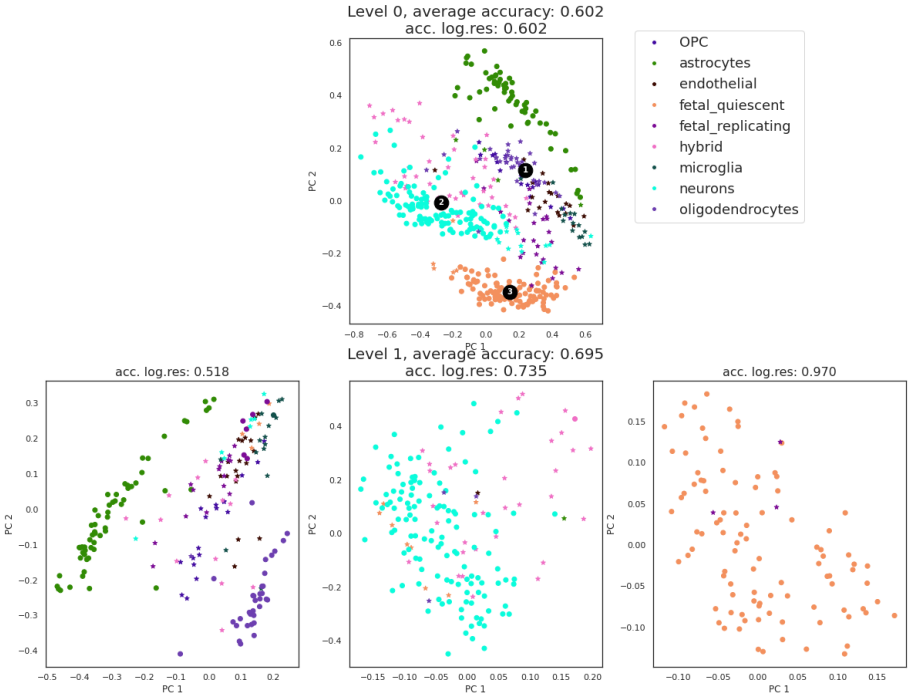


Figure 4.5: The hmPPCAs analysis on the Darmanis data-set. NUTS was used for inference and one level was found. A maximum accuracy of 0.695 was found at level 1. The top-level PPCA achieved an accuracy of 0.602. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

performed only slightly better than the two HmPPCAs models, and the UMAP accuracy was not much higher than the others.

4.2 Computational cost

Table 4.2 contains the computation times it took to infer the PPCA and MoPPCAs models on the Splatter data-sets. Each PPCA model was run only once at the top-level and its computation time is given. The MoPPCAs models were run repeatedly throughout the HmPPCAs models, so the average computation time of all MoPPCAs within the HmPPCAs is given instead. As expected, the PPCA models take less time to compute than the MoPPCAs models. This is the case for both NUTS and VB. We also see that VB takes considerably less time to infer solutions than NUTS for both the PPCA and the HmPPCAs models. The computational time goes up as the number of genes in the data-set rises, but more so when using NUTS than when using VB.

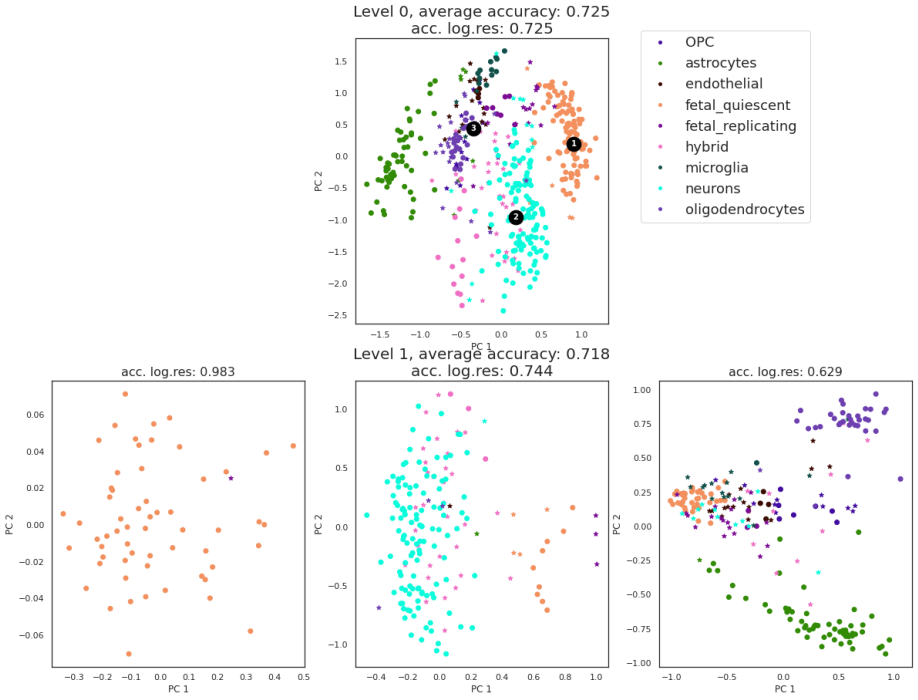


Figure 4.6: The hmPPCAs analysis on the Darmanis data-set. VB was used for inference and one level was found. A maximum accuracy of 0.725 was found at the top-level. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

The time in seconds of the MoPPCAs models was also compared while taking the number of mixture components that the MoPPCAs models included into account. This was done for the Splatter data-sets and the results of this are shown in Figure 4.9. In some cases, a MoPPCAs model was initialized with a specific number of mixture components but ended up finding fewer mixture components. For this reason, both the number of initial mixture components and the number of found mixture components were taken into account separately. In both cases, there was no apparent relationship between the number of mixture components in a MoPPCAs model and the time it took to fit the model. Whereas sometimes the measured time became shorter when fewer mixture components were involved, this was the opposite for other cases, and in many cases, there was no effect to be observed at all.

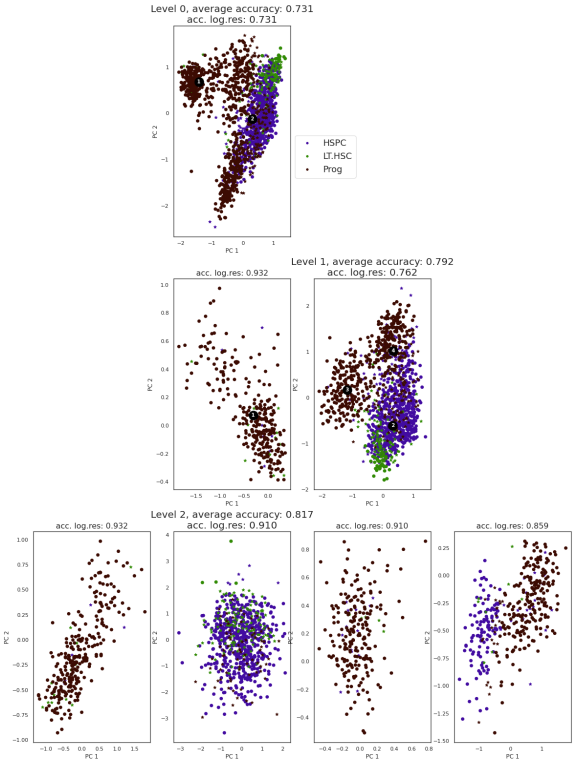


Figure 4.7: The HmPPCAs analysis on the Nestorowa data-set. NUTS was used as a sampling-method and five levels were found. A maximum accuracy of 0.817 was found at level 2. The top-level PPCA achieved an accuracy of 0.731. The clusters found on each level have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

Table 4.2: Computation times of PPCA and MoPPCAs models. In case multiple MoPPCAs were performed on a data-set within a HmPPCAs analyses, the average computation time of all MoPPCAs is reported.

	Splatter simple					Splatter complex				
genes	5	25	50	150	250	5	25	50	150	250
PPCA (NUTS)	21.1	38.7	53.6	163.3	227.4	54.8	157.0	150.1	312.5	441.7
MoPPCAs (NUTS)	589.7	1313.2	1599.9	9266.6	10327.2	997.7	1945.1	2083.9	8409.1	17229.8
PPCA (VB)	2.8	4.4	5.3	11.3	14.0	7.0	16.1	11.3	20.8	14.3
MoPPCAs (VB)	31.5	48.6	101.1	276.6	208.3	11.5	28.3	48.7	210.7	104.9

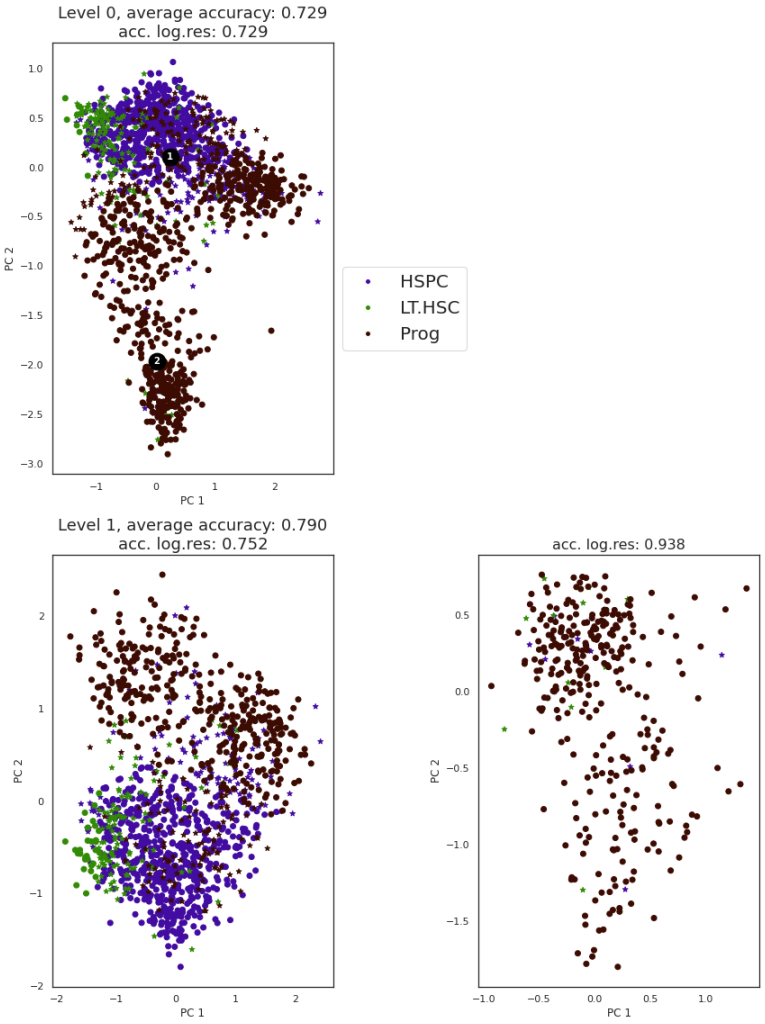


Figure 4.8: The HmPPCAs analysis on the Nestorowa data-set. VB was used as a sampling-method and two levels were found. A maximum accuracy of 0.790 was found at level 1. The top-level PPCA achieved an accuracy of 0.729. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

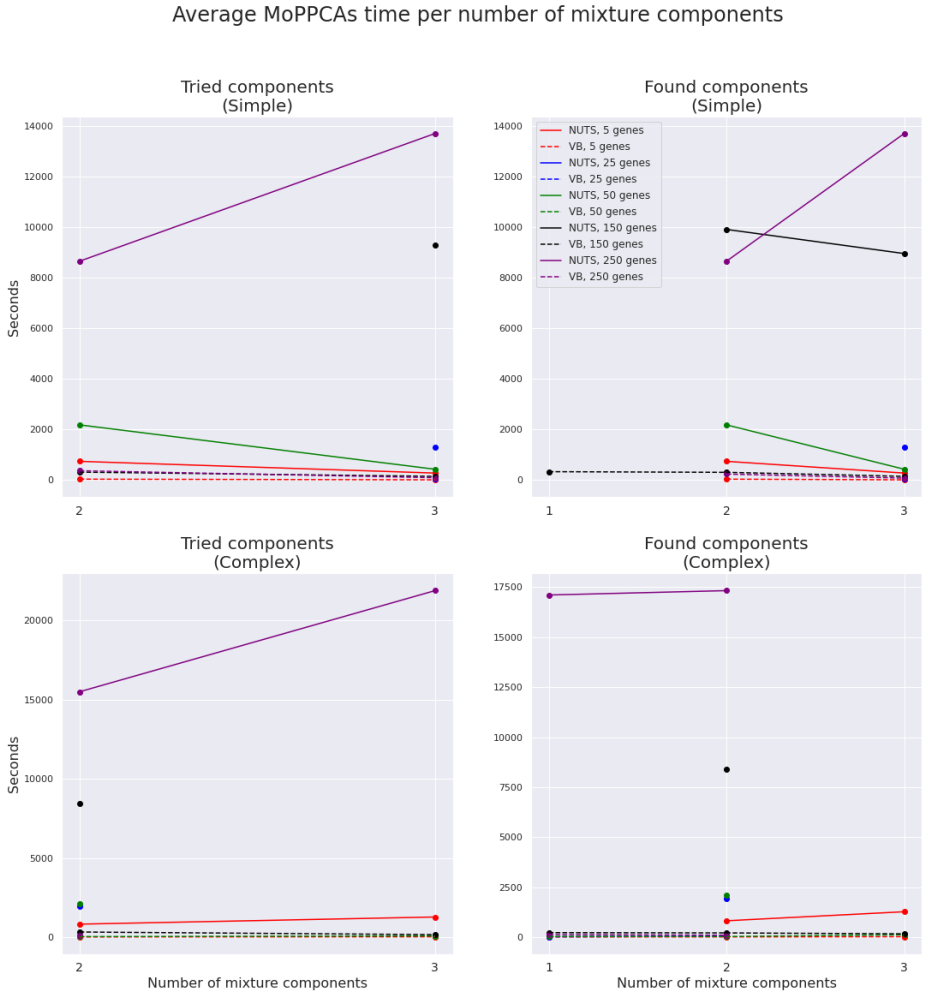


Figure 4.9: Average time to reach solution of MoPPCAs models in relation to their number of mixture components. The two figures on the left show the (average) time is took for a MoPPCAs to be solved in relation to the number of mixture components that the model was suggested to find. The MoPPCAs models were always looking for either 2 or 3 mixture components. In some cases, less mixture components were found. The plots on the right relate the time to the number of mixture components that were found by the model, which may in some cases have been only 1. The top two figures show this relation for the simple Splatter data-sets, the bottom two figures show this for the complex Splatter data-sets. In case of a dot that is not connected to another dot, no MoPPCAs model with a comparable number of mixture components has taken place in the HmPPCAs.

5 | Discussion

We have evaluated the visualization performance of the HmPPCAs model on several simulated and two real scRNA-seq data-sets. We then compared these results with those obtained using PPCA, t-SNE and UMAP. We found that HmPPCAs was most of the time able to improve upon PPCA. Since a PPCA is performed on the top-level data-set as the first step of HmPPCAs, it is always included within the HmPPCAs. Therefore, it was impossible for the HmPPCAs to score lower than the PPCA model. We also found that, although HmPPCAs improved over PPCA, it was not as accurate as t-SNE and UMAP. Both these techniques were in general able to create plots that separated the different cell types better than HmPPCAs could. UMAP scored highest of the two, but t-SNE performed almost equally well.

We saw that the accuracy achieved by the HmPPCAs was comparable when using VB and NUTS as inference method. For the PPCA model negligibles difference were observed, except in the case of the Darmanis data-set, where NUTS performed relatively weak compared to all other methods. The HmPPCAs scored equally well when using NUTS as when using VB on many of the data-sets.

NUTS and VB were also compared in terms of computation time till convergence. VB was always faster, for both the PPCA and HmPPCAs models. Not only did VB return results faster, but the computation time also increased less steeply when adding more genes, indicating that the method is more scalable.

The HmPPCAs model improved visualization performance over PPCA. In most cases, it was able to separate the different cell types better than just a single top-level PPCA. It should be noted though, that here, the level with the highest accuracy was taken to evaluate the performance of a HmPPCAs model. The visualization scores of our HmPPCAs model are therefore an upper bound of the models.

Another point for discussion is that our MoPPCAs model was biased to find a relatively high value of σ^2 and a lower value of $\boldsymbol{W}$, as discussed in Section 2.3.3. An expected value of σ^2 was based on the spread of the clusters in the latent space. Also, constraints

were forced upon the value of σ^2. To evaluate the effect of this approach, the value of σ^2 as found by the MoPPCAs model and the value found by the EM-algorithm for PPCA when performed on the separate clusters were compared with the values at which the constraints were set. Upon evaluation, the lower bound of σ^2 was (barely) high enough to include the value of σ^2 as found by PPCA, and the MoPPCAs solution for σ^2 was very similar to the PPCA solution. Still, future implementations of the MoPPCAs model in Stan should use priors and constraints for σ^2 which are based on their actual expectation. For example, according to Bishop & Tipping [11], the maximum likelihood estimator of σ^2 is given by $\sigma^2_{ML} = \frac{1}{d-2}\sum_{j=3}^{d}\lambda_j$, where λ_j is the jth eigenvalue of the covariance matrix of the cluster. Basing the expectation σ^2 on this approach would be more correct.

It should also be noted that the number of iterations used for NUTS (300) was relatively low, in an attempt to save computational cost. When $\pm$250 iterations were used, Stan would give warnings that the samples might not have converged, so 300 iterations were used to avoid these warnings. Still, 300 iterations are on the low side. We do see reasonable results that were largely comparable with the VB results, but it is possible that better results could have been achieved when using a higher number of iterations.

Lastly, the parameters of our interest (e.g. the latent data-set Z and the responsibility terms γ_i^k) were traceable. However, the factor loadings matrix W is more difficult to find, because W may undergo any arbitrary rotation. Therefore, if a researcher is interested in W, it may be better to use the EM algorithm. Alternatively, they could base the result on a single sample and not the mean of multiple samples.

Despite these shortcomings of our HmPPCAs implementation, HmPPCAs was able to provide more information about the local structure within clusters. A top-level PPCA might show some clusters, but the addition of hierarchy reveals structures within these clusters. For example, in Figure 4.2, the top-level PPCA would suggest that there are only four observable clusters of cells in the data-set. When taking the purple-geen cluster apart and visualizing its data-points in its own latent space, it becomes clear that this cluster actually consists of two sub-clusters which could not be separated by a top-level PPCA. Visualizing the latent space of the data-set gives a good idea of which data-points could form a distinct group. The latent space of clusters within the data-set provides more information on these clusters than the PPCA representation when all data is taken into account. When encountering clusters while using linear dimensionality reduction techniques, it may, therefore, reward to perform separate analyses on the clusters within the data-set.

Additionally, it is noteworthy that HmPPCAs does not necessarily separate cell types, but specifically cluster structures in the data. Figure 4.7 shows how one group, the

progenitor cells, shows different clusters. Each of those clusters is taken separately for further analysis, even though they belong to the same group of cells. This is not necessarily a bad thing and it is possible that there are differences to be found in the cells belonging to this type. Therefore, it could be the case that even subsets of the data-set that belong to the same type of cells need different analyses in terms of dimensionality reduction.

When comparing the HmPPCAs with UMAP and t-SNE however, it did not reach their visualization performance quite yet. It seems therefore that UMAP and t-SNE are non-linear approaches that are better at separating groups within the data than PPCA, even when extended to HmPPCAs. This might be due to the automatic clustering since the interactive EM implementation of HmPPCAs has been found to improve on t-SNE [12]. They may therefore have the advantage in the visualization assay that was performed.

All in all, extending PPCA to a hierarchical model is a good way to obtain more insight into the local structure of the data-set. It might not be as good as UMAP or t-SNE to separate different structures within the data, but it is a valuable extension to linear dimensionality reduction techniques. Also, it seems that VB did not perform significantly worse than NUTS and possibly even better on some models. VB also took less time to converge and showed better scalability when adding more genes. The ADVI algorithm has therefore proven to be of great value when using Stan.

A | Fitting a model to the data

A.1 The Gaussian distribution

Let's say we observe a random variable x. Random variables have a value that is not predetermined, but sometimes we know how the probability of drawing a random value is depending on that value. When the random number can assume any number (i.e. not only discrete integers), then a probability density function gives the probability density for the support of $\mathcal{X}$ (i.e. the values that x can assume). One well-known example of such a probability density function is the Gaussian distribution. It is defined by A.1.

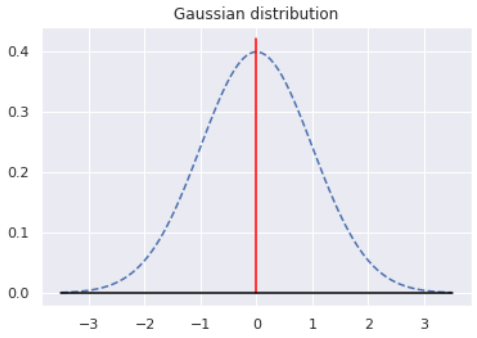


Figure A.1: The probability density function for a standard Gaussian or normal distribution.

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{A.1}$$

The mean μ is the value around which all drawn samples are centered, and the standard deviation σ tells us how far the drawn samples are spread out from this point. Drawing a sample of n independent and identically distributed (i.i.d.) data points from the Gaussian distribution given its mean and variance is done with a probability of the product of the probabilities of drawing the individual data points (see A.2, where $\mathbf{x}^n$ is a set of n data points).

$$p(\mathbf{x}^n|\mu,\sigma^2) = \prod_{i=1}^{n} \mathcal{N}(x_i|\mu,\sigma^2) \tag{A.2}$$

Sometimes, we deal with multivariate random variables. In this case, every data-point $\boldsymbol{x}_i$ and the mean $\boldsymbol{\mu}$ are d-dimensional vectors and the covariance matrix $\boldsymbol{\Sigma}$ is a $d \times d$ matrix noting the co-variances among all dimensions. In this case, $\mathbf{X}$ is a matrix containing n rows $\mathbf{x}_i$ of length d for each data point. In this case, $\boldsymbol{\mu}^d$ is a d-dimensional vector where μ_j notes the mean of the jth variable. The covariances between all variables are given by positive semi-definite matrix $\boldsymbol{\Sigma}$, where $\boldsymbol{\Sigma}_{i,j}$ describes the covariance between the i^{th} and j^{th} variable. Note that $\boldsymbol{\Sigma}$ is symmetric, as a property of covariance, and that $\boldsymbol{\Sigma}_{i,i} = cov(x_i, x_i) = var(x_i)$. For a set of independent variables, $\boldsymbol{\Sigma}$ is a diagonal matrix, since the $cov(x_i, x_j)$ is 0 when $i \neq i$. In a multivariate setting, the Gaussian distribution takes the form of A.3.

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{m}{2}}} \frac{1}{|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{A.3}$$

A.2 Likelihood and the posterior

Let's say we have the probability density function of $\boldsymbol{x}$, $p(\boldsymbol{x}|\theta)$, where θ holds the parameters of the distribution of $p(x)$ (e.g. in this case $\theta = (\mu, \sigma^2)$). Apart from deriving the probability of drawing a value from a given distribution and its parameters, we can also derive the likelihood of the parameters of the distribution being certain values given only the observed data. A noteworthy difference between the probability and the likelihood is that the probability necessarily sums up to 1 when integrated over the support of $\mathcal{X}$, but the likelihood does not when integrating over the parameter space of θ. The likelihood is given by $\mathcal{L}(\theta|\boldsymbol{x}) = p(\boldsymbol{x}|\theta)$. This likelihood is often given by its logarithm, known as the log-likelihood. On the one hand, this makes mathematical manipulations with Gaussian distributions a lot easier. On the other hand, the log function converts very small numbers to larger ones, which helps to avoid numerical underflow on electronic devices. Converting the likelihood function of A.2 to a corresponding log-likelihood function results in A.4 (derivation in B.1).

$$\ln \mathcal{L}(\mu, \sigma^2|\boldsymbol{x}) = \ln p(\mathbf{x}|\mu, \sigma^2) = -\frac{n}{2} \ln 2\pi - \frac{n}{2} \ln \sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{A.4}$$

Using the log-likelihood has two advantages: It makes mathematical manipulations to the distribution fairly easy and it avoids numerical underflow errors by converting very small numbers to larger ones. Since the log function is monotonically increasing, the maximum of the likelihood function s found for the same value as the maximum of the log-likelihood.

The likelihood, however, is not the same as the probability that θ is responsible for generating data $\boldsymbol{x}$. According to Bayes' theorem, $p(A|B) = \frac{p(B|A)p(A)}{p(B)}$, and therefore the likelihood of parameters θ being responsible for observed data $\boldsymbol{x}$ is given by $p(\theta|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\theta)p(\theta)}{p(\boldsymbol{x})}$. In this construction, the first term $P(\boldsymbol{x}|\theta)$ is the likelihood and the second term $p(\theta)$ is known as the prior. The prior takes the distribution of the parameters into account. Since the divisor is not dependent on θ, $p(\boldsymbol{x}|\theta)p(\theta) \propto p(\theta|\boldsymbol{x})$ so that the product of the prior and likelihood determines the probability of θ being a certain value given the data. The last term, $p(\boldsymbol{x})$, is known as the evidence. It normalizes the probability, so that the posterior integrated over the parameter-space θ sums to 1. When estimating θ given $\boldsymbol{x}$, a point estimate is given by $\text{argmax}_\theta p(\theta|\boldsymbol{x})$.

Often, a prior distribution of θ is unknown or assumed to be of a uniform shape. In this case, the prior $p(\theta$ is a constant that is non-dependent on θ and $p(\theta|\boldsymbol{x}) \propto p(\boldsymbol{x}|\theta)$. The point estimate for θ is then $\text{argmax}_\theta p(\boldsymbol{x}|\theta)$, known as the maximum likelihood (**ML**) estimate. This estimate can be found by taking the derivative of $\mathcal{L}(\theta|\boldsymbol{x})$ and setting it to 0. For example, the ML estimates for μ is given by $\mu_{ML} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and for σ^2 by $\sigma^2_{ML} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{ML})^2$ (see B.2 and B.3).

A.3 Gaussian mixture models and EM

Now imagine that our data-set consists of different 'groups' that all have their own mean and covariance matrix. For example, we may have a data-set with several cell-types, and every cell-type demonstrates their own pattern of gene expression. If our data-set consists of K different Gaussians, then we have a Gaussian mixture model (**GMM**), of which the probability density function is denoted by A.5.

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{A.5}$$

In this example, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ give the means and covariance matrix of each distribution k and π_k is the proportion in which mixture component k is responsible for generating the data-set, such that $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$ for all k. Finding the set of parameters $\theta = (\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ in this scenario is not as straight forward as the ML approach we saw earlier. To approximate θ for this model, we can use an iterative expectation-maximization (**EM**) algorithm. An EM-algorithm consist of two steps:

- In the Expectation-step, the expected log-likelihood function of the complete data-set (observed data and missing parameters) is estimated from their joint probabilities.
- In the maximization step, the parameters θ are maximized based on the estimated log-likelihood.

After we have chosen some arbitrary initial values for $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and π_k, we start with the E-step. In the E-step of our algorithm, we try to find the complete log-likelihood function. Let $\boldsymbol{z}_i$ denote which mixture component has generated data-point i. Then the complete likelihood function is given by A.6, where $\mathbb{I}[z_i = j]$ evaluates to 1 if true and 0 otherwise.

$$\mathcal{L}(\boldsymbol{X}|\theta, \boldsymbol{z}) = \prod_{i=1}^{n} \prod_{j=1}^{K} \mathcal{N}(\boldsymbol{x}_i|\theta_k)^{\mathbb{I}[z_i=j]} \tag{A.6}$$

Now, $\boldsymbol{z}$ (i.e. which data-points were generated by which distribution) is unknown, but instead of $\mathbb{I}[z_i = j]$, we could use the probability $p(z_i = j)$. We know that $p(\boldsymbol{x}_i|z_i = j) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ and therefore, we can compute the posterior $p(z_i = j|\boldsymbol{X})$ according to Bayes' theorem as done in A.7.

$$p(z_k = j|\mathbf{x}_i) = \frac{p(z_i = 1)p(\mathbf{x}_i|z_i = j)}{\sum_{k=1}^{K} p(z_i = k)p(\mathbf{x}_i|z_i = k)} = \frac{\pi_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \tag{A.7}$$

In the literature, the posterior probability is often denoted as $\gamma(z_{k,i}) = p(z_i = k|\boldsymbol{X})$. With this, we can compute the expected complete log-likelihood $\ln \mathcal{L}(\theta|\boldsymbol{X}) = \sum_{i=1}^{n} \sum_{j=1}^{K} p(z_i = j) \ln \mathcal{N}(\boldsymbol{x}_i|\theta_k)$. In the M-step, we utilise this expected log likelihood to calculate ML-estimates for the mean $\boldsymbol{\mu}_k$, covariance-matrix $\boldsymbol{\Sigma}_k$ and mixing coefficient π_k of each distribution k.

We start off with the mean of each distribution k. Setting the derivative with respect to $\boldsymbol{\mu}_k$ to 0 shows that the ML estimate is simply given as the weighted average of all points belonging to distribution k (A.8.

$$\boldsymbol{\mu}_k = \frac{1}{\sum_{i=1}^{n} \gamma(z_{k,i})} \sum_{i=1}^{n} \gamma(z_{k,i}) \mathbf{x}_i^m \tag{A.8}$$

Similarly, the variance is computed as the weighted variance.

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_{i=1}^{n} \gamma(z_{k,i})} \sum_{i=1}^{n} \gamma(z_{k,i})(\mathbf{x}_i^m - \boldsymbol{\mu}_k)(\mathbf{x}_i^m - \boldsymbol{\mu}_k)^T \tag{A.9}$$

Finally, π_k is obtained as the average of the individual probabilities with which points belong to distribution k.

$$\pi_k = \frac{\sum_{i=1}^{n} \gamma(z_{k,i})}{n} \tag{A.10}$$

Having obtained new values for all relevant parameters, we return to the E-step and obtain new posterior probabilities. The process is repeated until a converged state is reached.

B | Gaussian distribution

B.1 Converting probability to its log-probability

$$
\begin{aligned}
\ln p(\mathbf{x}_1^n|\mu,\sigma^2) &= \prod_{i=1}^{n}\mathcal{N}(x_i|\mu,\sigma^2) \\
\ln p(\mathbf{x}_1^n|\mu,\sigma^2) &= \ln\prod_{i=1}^{n}\frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}}e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2} \\
\ln p(\mathbf{x}_1^n|\mu,\sigma^2) &= \sum_{i=1}^{n}\ln\frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}}e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2} \\
\ln p(\mathbf{x}_1^n|\mu,\sigma^2) &= \sum_{i=1}^{n}\ln\frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} + \ln e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2} \\
\ln p(\mathbf{x}_1^n|\mu,\sigma^2) &= \sum_{i=1}^{n}-\frac{1}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}(x_i-\mu)^2 \\
\ln p(\mathbf{x}_1^n|\mu,\sigma^2) &= \sum_{i=1}^{n}-\frac{1}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}(x_i-\mu)^2 \\
\ln p(\mathbf{x}_1^n|\mu,\sigma^2) &= -\frac{n}{2}\ln 2\pi - \frac{n}{2}\ln\sigma^2 - \sum_{i=1}^{n}\frac{1}{2\sigma^2}(x_i-\mu)^2
\end{aligned}
\tag{B.1}
$$

B.2 Maximum likelihood for μ

$$\begin{aligned}
\frac{\partial}{\partial\mu}\big(-\frac{n}{2}\ln 2\pi-\frac{n}{2}\ln\sigma^2-\sum_{i=1}^{n}\frac{1}{2\sigma^2}(x_i-\mu_{ML})^2\big)&=0\\
\frac{\partial}{\partial\mu}\big(-\sum_{i=1}^{n}\frac{1}{2\sigma^2}(x_i-\mu_{ML})^2\big)&=0\\
-\sum_{i=1}^{n}\frac{1}{2\sigma^2}(-2x_i+2\mu_{ML})&=0\\
\sum_{i=1}^{n}\frac{2x_i}{2\sigma^2}&=\sum_{i=1}^{n}\frac{2\mu_{ML}}{2\sigma^2}\\
\sum_{i=1}^{n}x_i&=n\mu_{ML}\\
\frac{1}{n}\sum_{i=1}^{n}x_i&=\mu_{ML}
\end{aligned}\tag{B.2}$$

B.3 Maximum likelihood for σ^2

$$\begin{aligned}
\frac{\partial}{\partial\sigma^2}\big(-\frac{n}{2}\ln 2\pi-\frac{n}{2}\ln\sigma^2_{ML}-\sum_{i=1}^{n}\frac{1}{2\sigma^2_{ML}}(x_i-\mu_{ML})^2\big)&=0\\
\frac{\partial}{\partial\sigma^2}\big(-\frac{n}{2}\ln\sigma^2_{ML}-\sum_{i=1}^{n}\frac{1}{2\sigma^2_{ML}}(x_i-\mu_{ML})^2\big)&=0\\
\frac{n}{2\sigma^2_{ML}}-\sum_{i=1}^{n}\frac{1}{2{\sigma^2_{ML}}^2}(x_i-\mu_{ML})^2&=0\\
n\sigma^2_{ML}-\sum_{i=1}^{n}(x_i-\mu_{ML})^2&=0\\
\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_{ML})^2&=\sigma^2_{ML}
\end{aligned}\tag{B.3}$$

C | PPCA model defined in Stan

This below is an example of a PPCA model defined in Stan. Note that priors have been specified over the parameters $\boldsymbol{\mu}$, σ^2 and $\boldsymbol{W}$ (line 32). The distributions of x and z have been specified for the model to infer the log-likelihood by itself (line 39). The user could also have passed the log-likelihood to the model directly instead. The latent distribution is specified to have a covariance matrix $\boldsymbol{I}_m$, to make sure that the latent dimensions show little to no correlation.

```
1  data{
2      int<lower=0> N;// number of data-points
3      int<lower=0> D;// number of dimensions in observed data-set
4      int<lower=0> M;// number of dimensions in latent data-set
5      vector[D] x[N];// observed data
6  }
7
8  transformed data{
9      vector[M] mean_z;
10     matrix[M,M] cov_z;
11
12     for (m in 1:M){
13         mean_z[m] = 0.0;
14         for (n in 1:M){
15             if (m==n){
16                 cov_z[m,n]=1.0;
17             } else{
18                 cov_z[m,n]=0.0;
19             }
20         }
21     }
22 }
23
24 parameters{
25     matrix[M,N] z;          // latent data
26     matrix[D,M] W;          // factor loadings
27     real<lower=0> sigma;    // standard deviations
```

```
28      vector[D] mu;             // added means
29  }
30
31  model{
32      // priors
33      for (d in 1:D){
34          W[d] ~ normal(0.0,sigma);
35          mu[d]~normal(0.0, 5.0) ;
36          }
37      sigma~lognormal(0.0, 1.0) ;
38
39      // likelihood
40      for (n in 1:N){
41          z[:,n] ~ multi_normal(mean_z, cov_z);
42          x[n] ~ normal(W*col(z,n)+mu, sigma);
43          }
44  }
```

Listing C.1: PCPA model defined in Stan

D | HmPPCAs results on Splatter data-sets

D.1 Simple Splatter data-sets

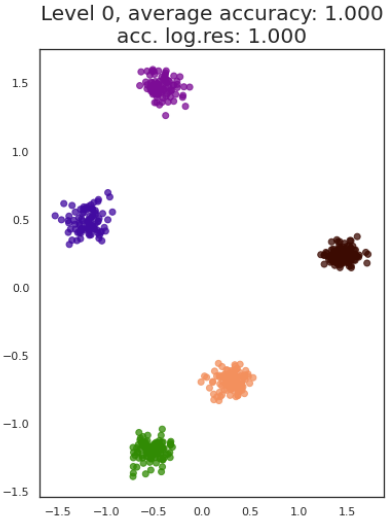


Figure D.1: HmPPPCAs model performed on the simple Splatter data-set with 5 genes using NUTS. The maximum accuracy of 1.0 was found on the top-level. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types.

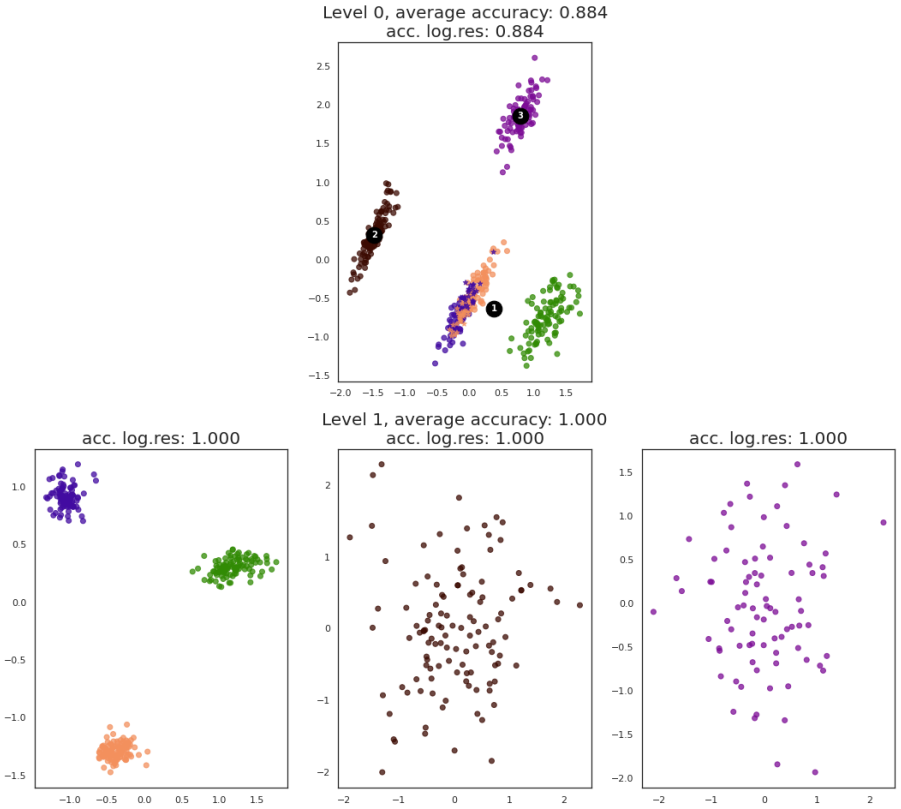


Figure D.2: HmPPPCAs model performed on the simple Splatter data-set with 25 genes using NUTS. The maximum accuracy of 1.0 was found at level 1. The top-level PPCA achieved an accuracy of 0.884. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

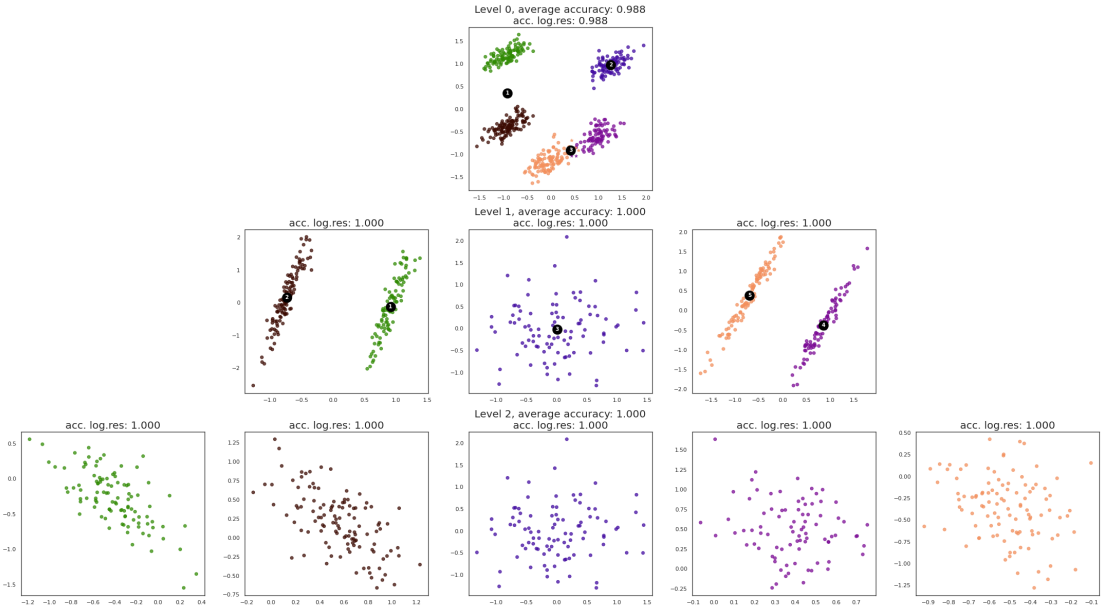


Figure D.3: HmPPPCAs model performed on the simple Splatter data-set with 50 genes using NUTS. The maximum accuracy of 1.0 was found at level 1. The top-level PPCA achieved an accuracy of 0.988. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

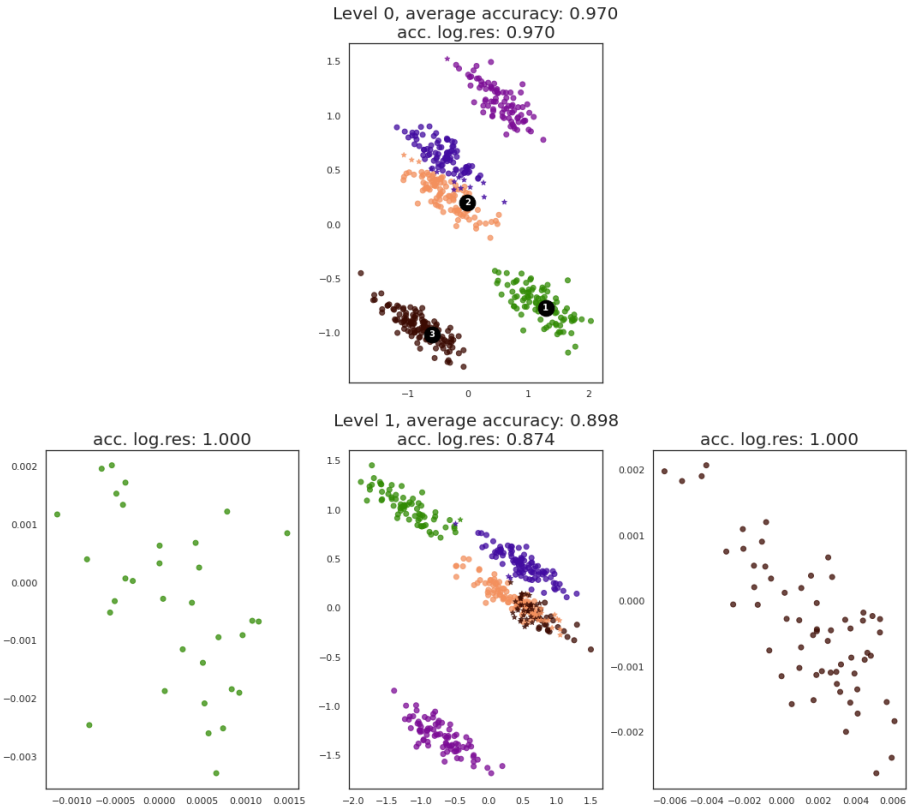


Figure D.4: HmPPPCAs model performed on the simple Splatter data-set with 150 genes using NUTS. The maximum accuracy of 0.970 was found on the top-level. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

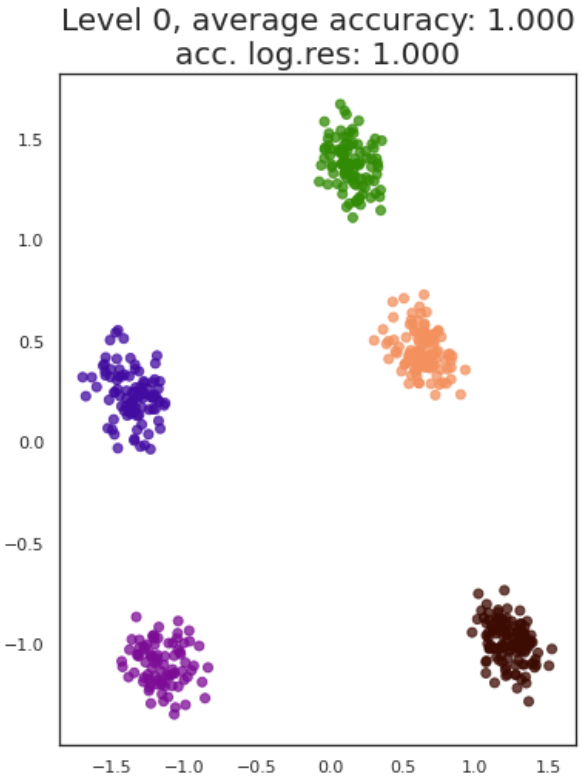


Figure D.5: HmPPPCAs model performed on the simple Splatter dataset with 5 genes using VB. The maximum accuracy of 1.0 was found on the top-level. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types.

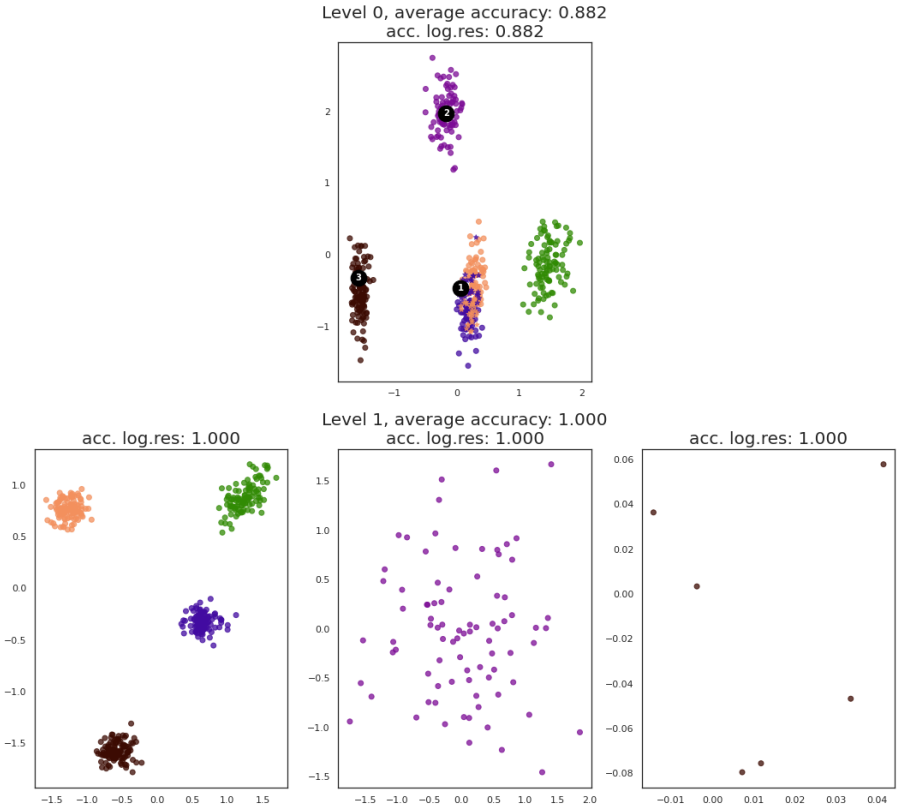


Figure D.6: HmPPPCAs model performed on the simple Splatter data-set with 25 genes using VB. The maximum accuracy of 1.0 was found at level 1. The top-level PPCA achieved an accuracy of 0.882. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

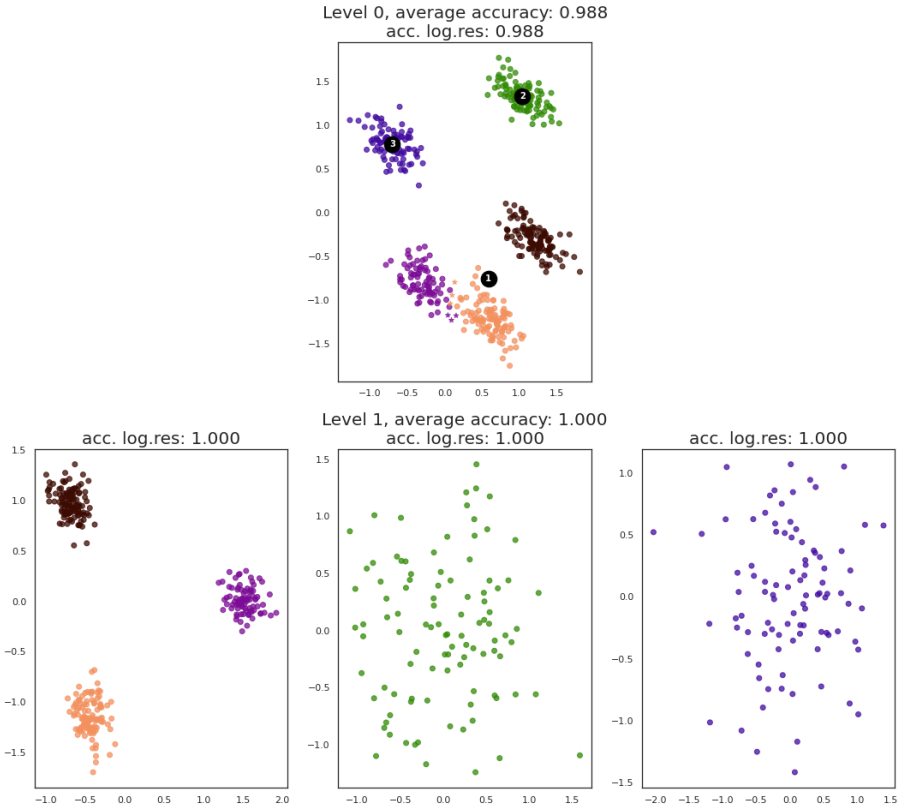


Figure D.7: HmPPPCAs model performed on the simple Splatter data-set with 50 genes using VB. The maximum accuracy of 1.0 was found at level 1. The top-level PPCA achieved an accuracy of 0.988. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

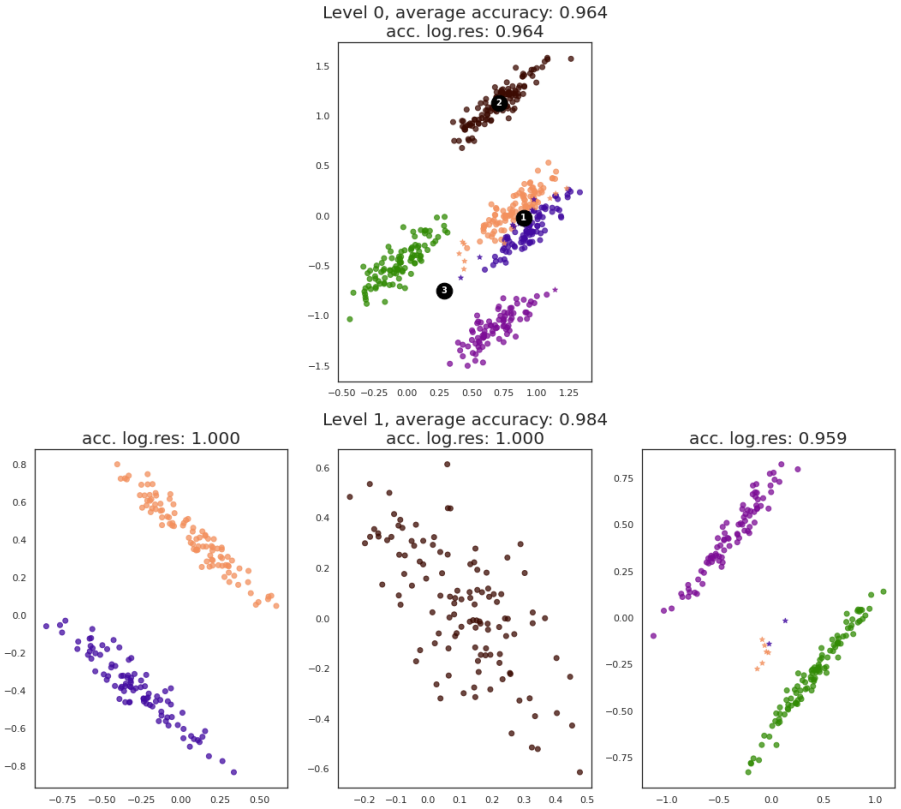


Figure D.8: HmPPPCAs model performed on the simple Splatter data-set with 150 genes using VB. The maximum accuracy of 0.984 was found at level 1. The top-level PPCA achieved an accuracy of 0.964. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

D.2 Complex Splatter data-sets

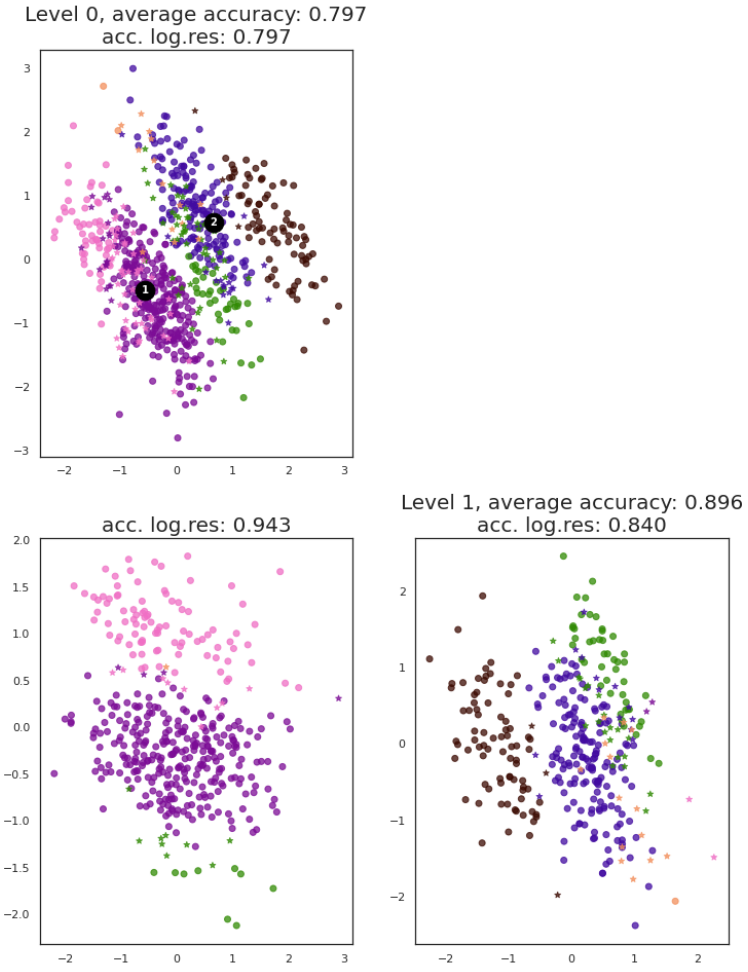


Figure D.9: HmPPPCAs model performed on the complex Splatter data-set with 5 genes using NUTS. The maximum accuracy of 0.896 was found at level 1. The top-level PPCA achieved an accuracy of 0.797. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

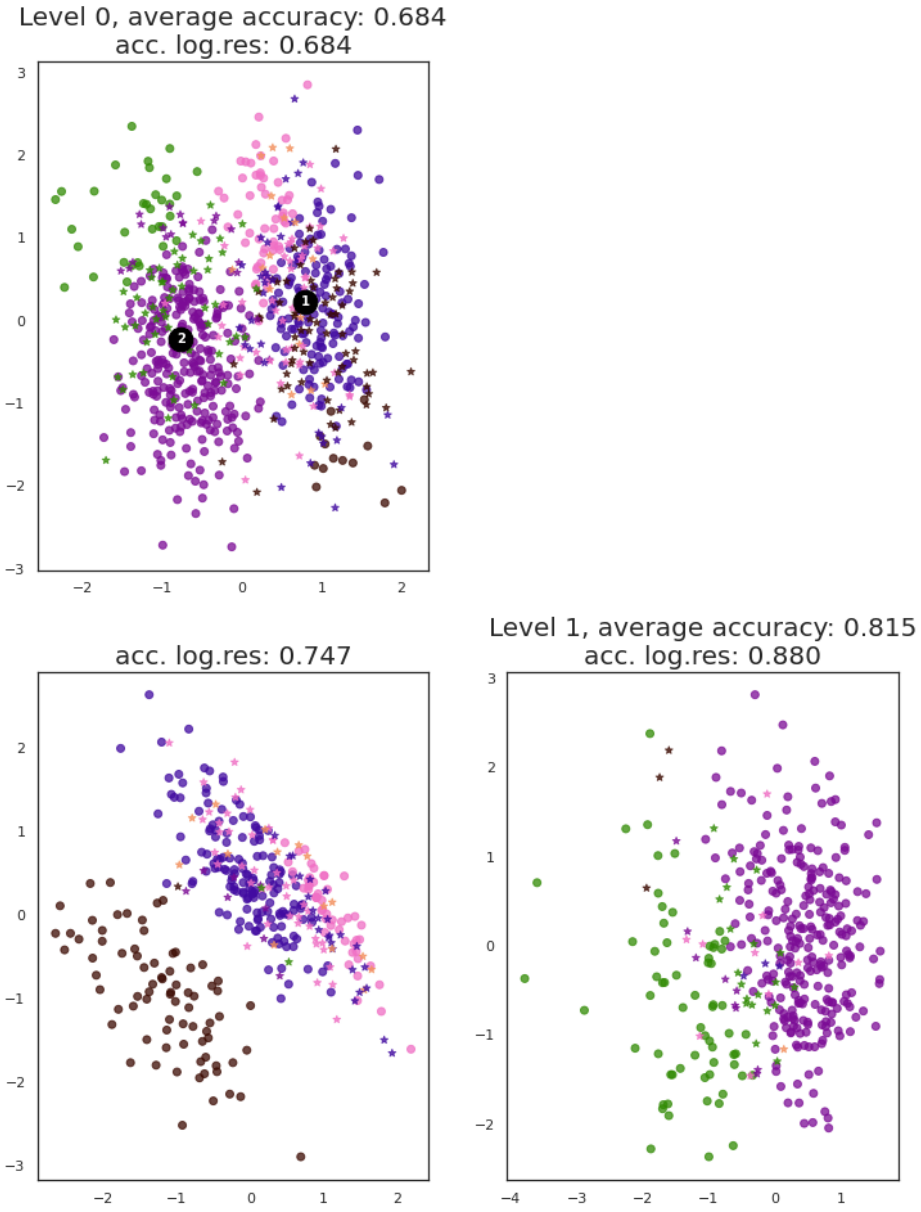


Figure D.10: HmPPPCAs model performed on the complex Splatter dataset with 25 genes using NUTS. The maximum accuracy of 0.815 was found at level 1. The top-level PPCA achieved an accuracy of 0.684. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Datapoints that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

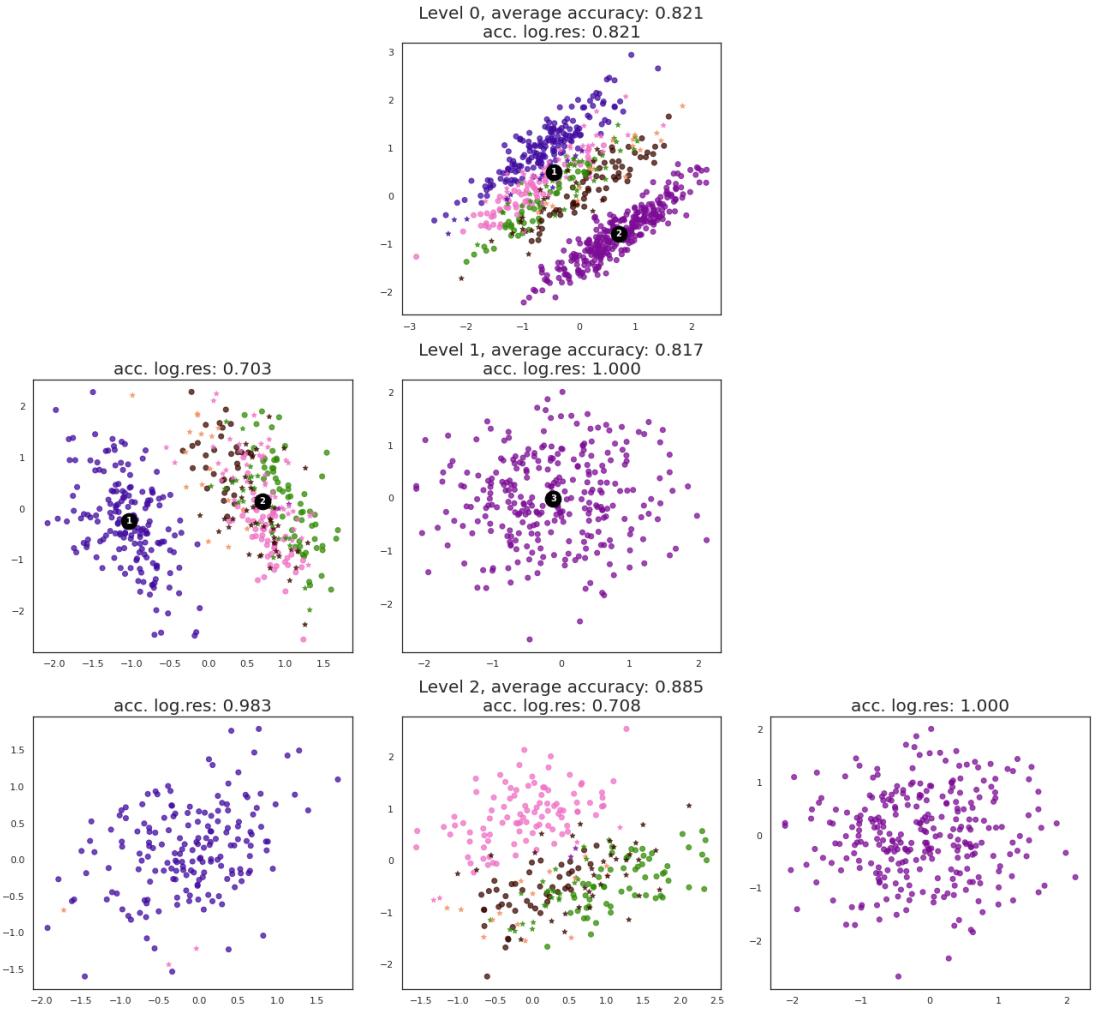


Figure D.11: HmPPPCAs model performed on the complex Splatter dataset with 50 genes using NUTS. The maximum accuracy of 0.885 was found at level 2. The top-level PPCA achieved an accuracy of 0.821. The clusters found within each level have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

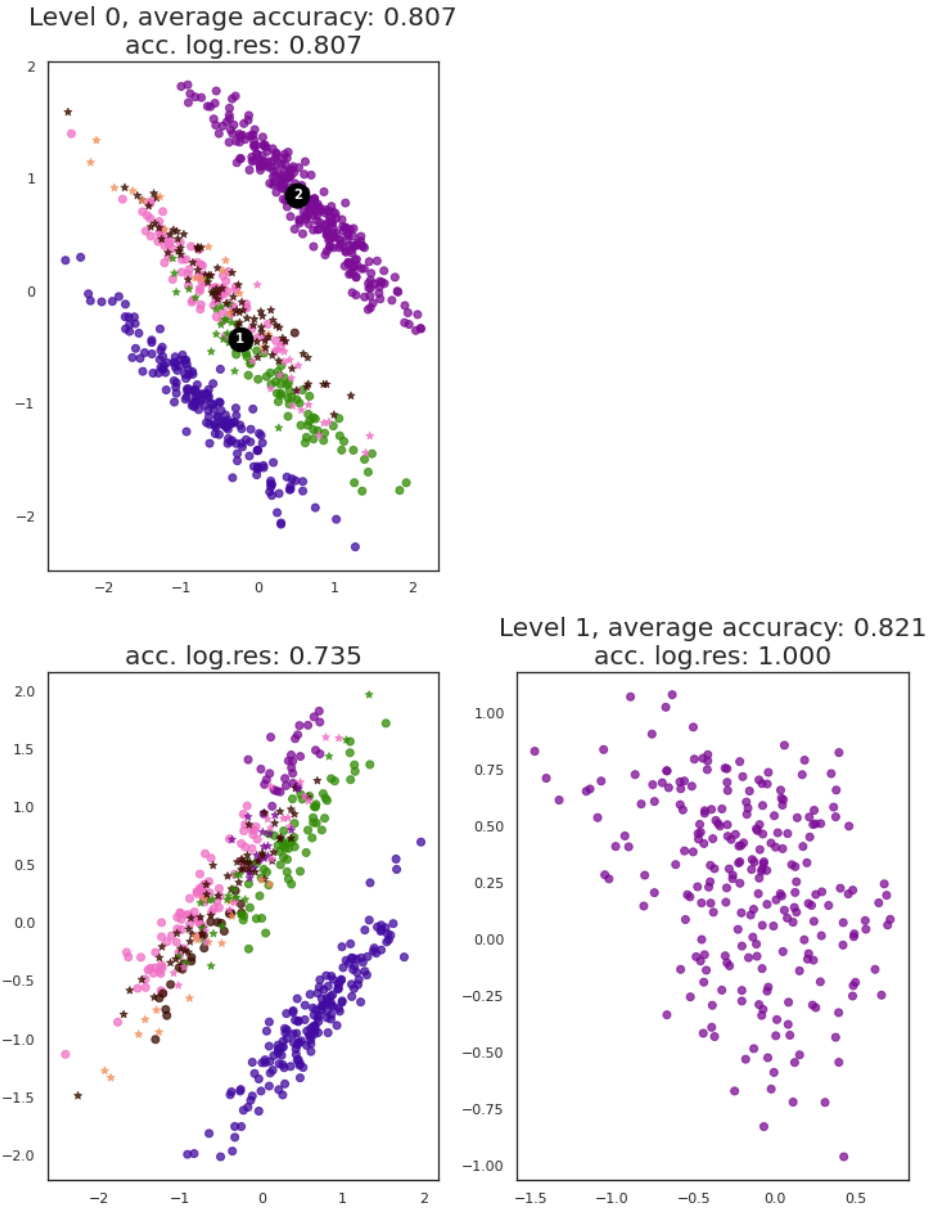


Figure D.12: HmPPPCAs model performed on the complex Splatter dataset with 150 genes using NUTS. The maximum accuracy of 0.821 was found at level 1. The top-level PPCA achieved an accuracy of 0.807. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

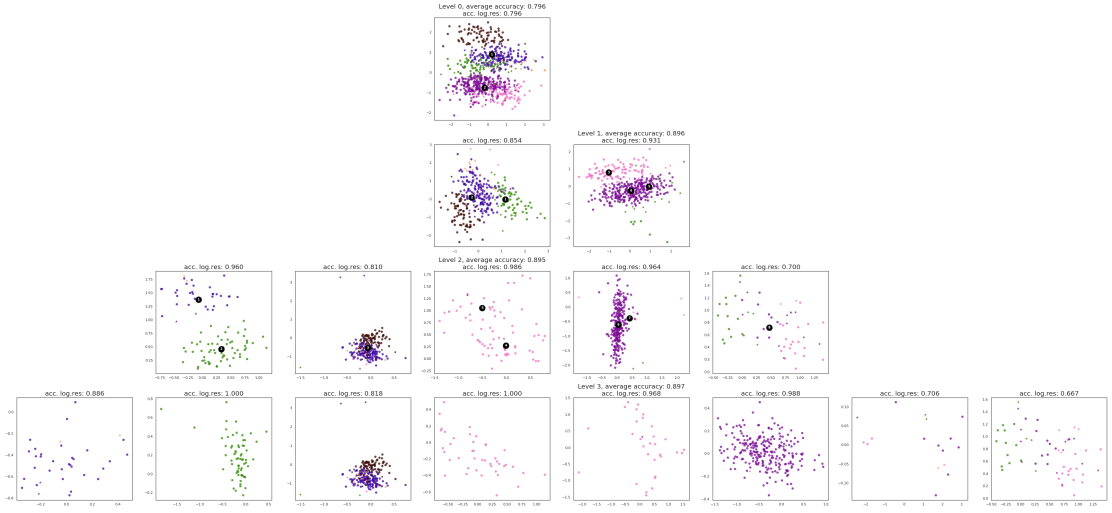


Figure D.13: HmPPPCAs model performed on the complex Splatter dataset with 5 genes using VB. The maximum accuracy of 0.897 was found at level 3. The top-level PPCA achieved an accuracy of 0.796. The clusters found within each level have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

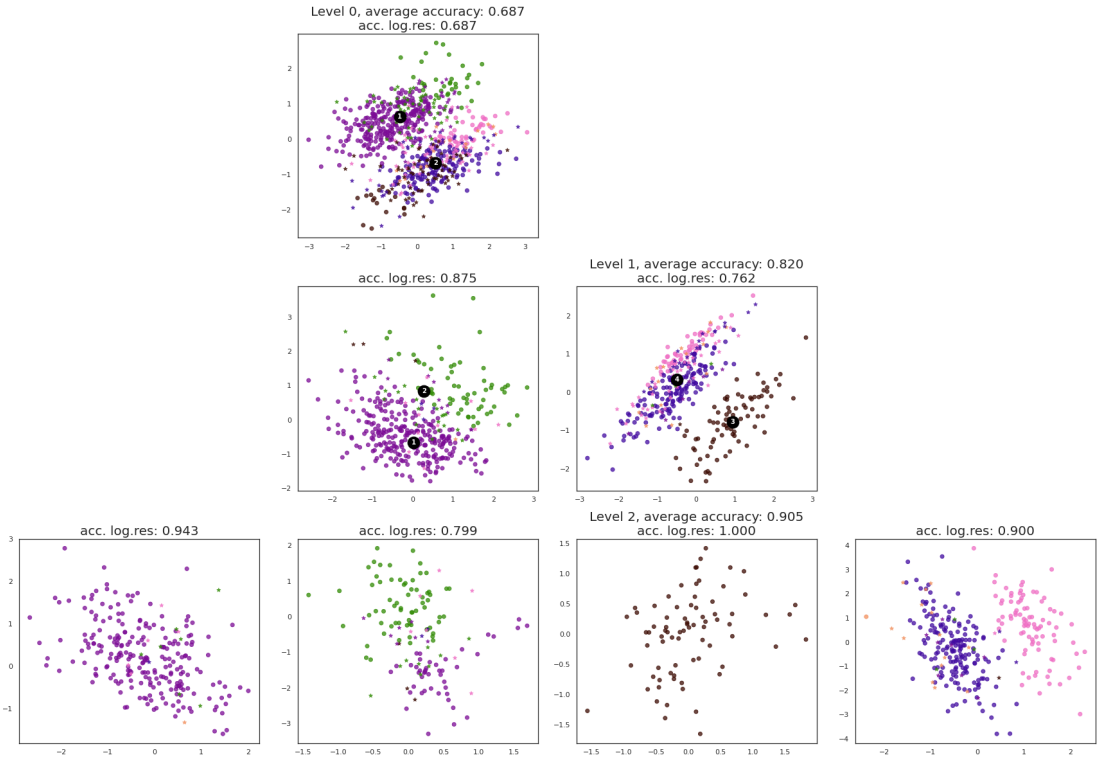


Figure D.14: HmPPPCAs model performed on the complex Splatter dataset with 25 genes using VB. The maximum accuracy of 0.905 was found at level 2. The top-level PPCA achieved an accuracy of 0.687. The clusters found within each level have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

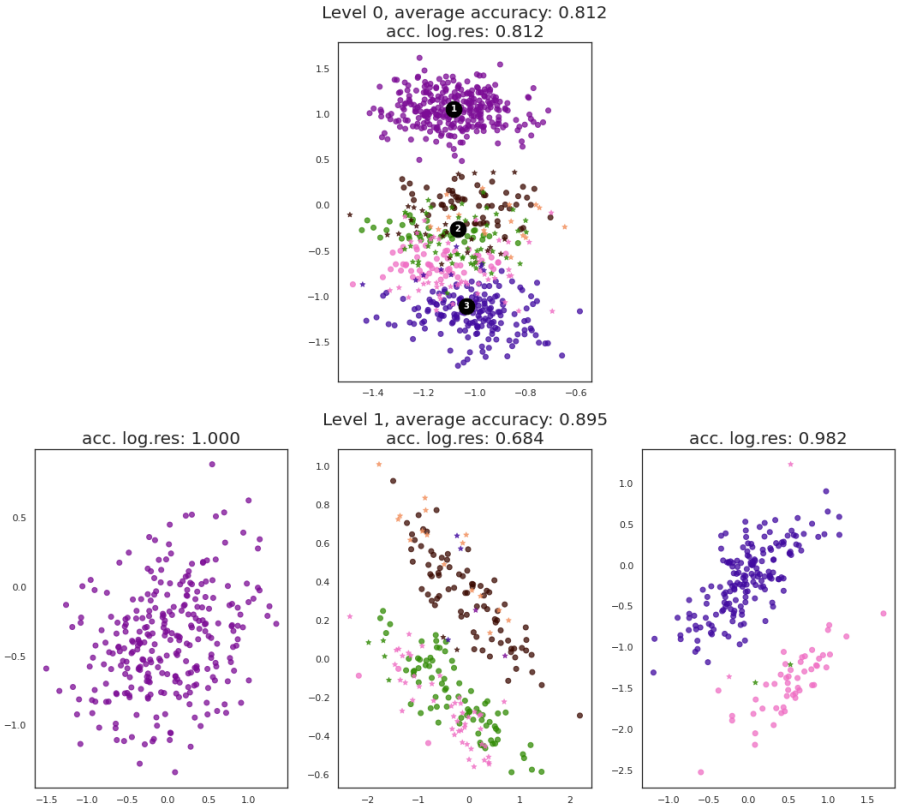


Figure D.15: HmPPPCAs model performed on the complex Splatter dataset with 50 genes using VB. The maximum accuracy of 0.895 was found at level 1. The top-level PPCA achieved an accuracy of 0.812. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

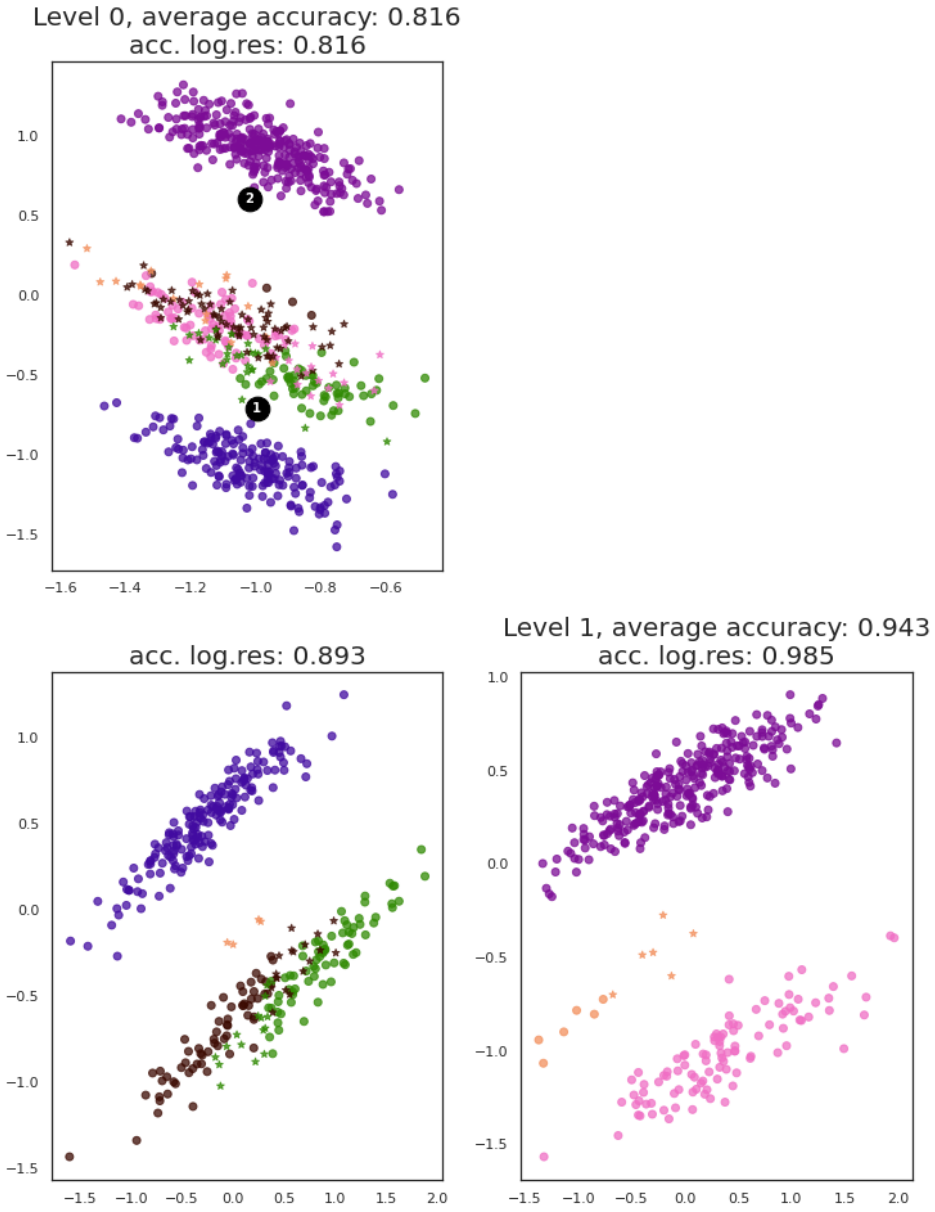
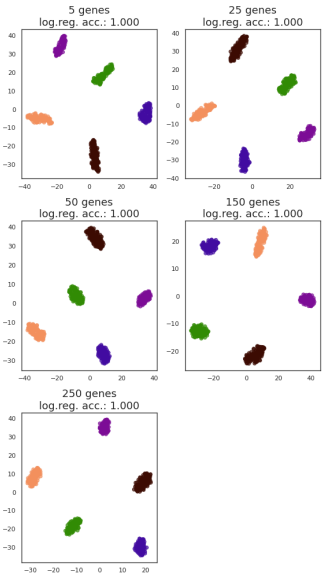
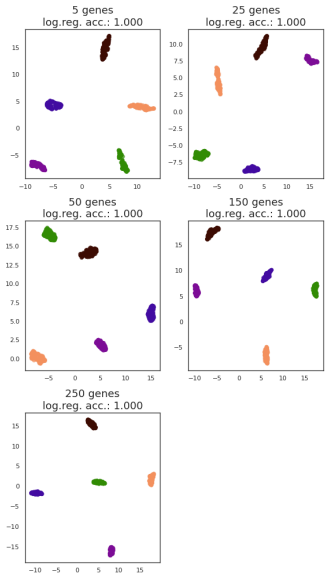


Figure D.16: HmPPPCAs model performed on the complex Splatter dataset with 150 genes using VB. The maximum accuracy of 0.943 was found at level 1. The top-level PPCA achieved an accuracy of 0.816. The clusters found within the top-level PPCA have been numbered. The colours indicate different cell types. Datapoints that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

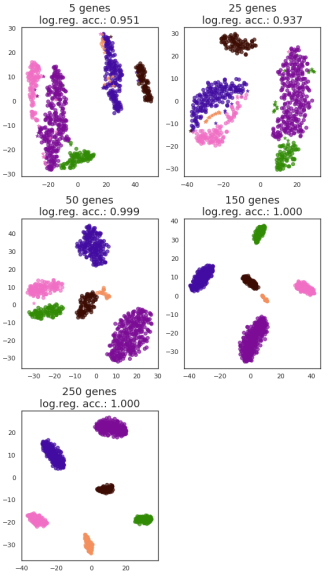
D.3 UMAP and t-SNE results on the Splatter data-sets



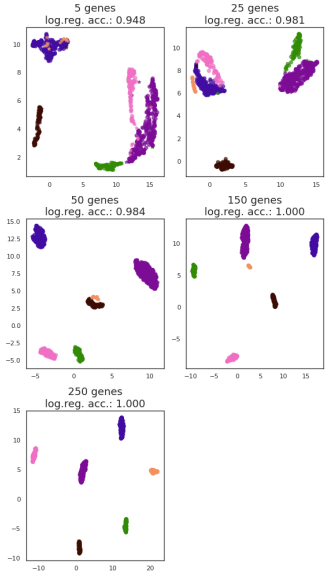
(a) t-SNE performance on the simple Splatter data-sets



(b) UMAP performance on the simple Splatter data-sets



(c) t-SNE performance on the complex Splatter data-sets



(d) UMAP performance on the complex Splatter data-sets

Figure D.17: t-SNE and UMAP performance on the Splatter data-sets. The resulting plots when analyzing the simple Splatter data-sets with t-SNE (a) and UMAP (b) and when analyzing the complex Splatter data-sets with t-SNE (c) and UMAP (d). Colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

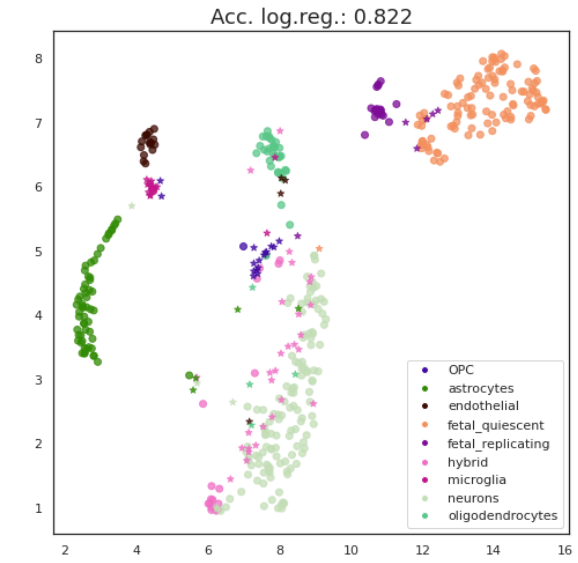


Figure D.18: UMAP results of the Darmanis data-set. An accuracy of 0.822 was measured after performing a logistic regression with a 5-fold cross validation. Colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

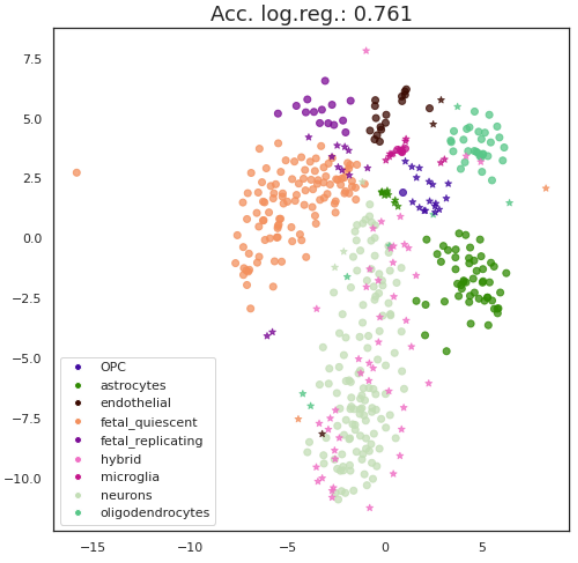


Figure D.19: t-SNE results of the Darmanis data-set. An accuracy of 0.761 was measured after performing a logistic regression with a 5-fold cross validation. Colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

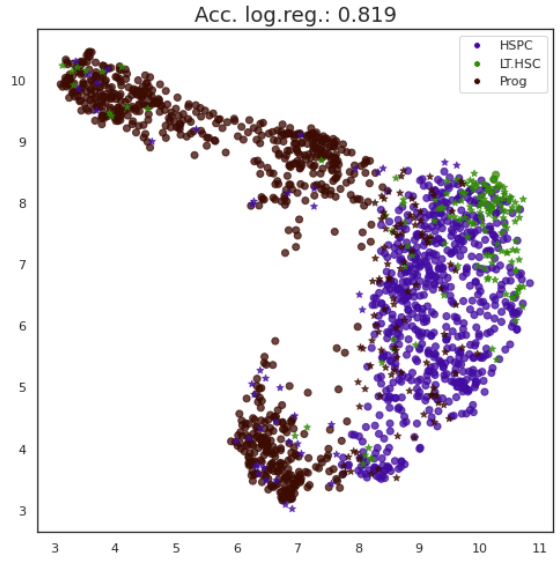


Figure D.20: UMAP results of the Nestorowa data-set. An accuracy of 0.819 was measured after performing a logistic regression with a 5-fold cross validation. Colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

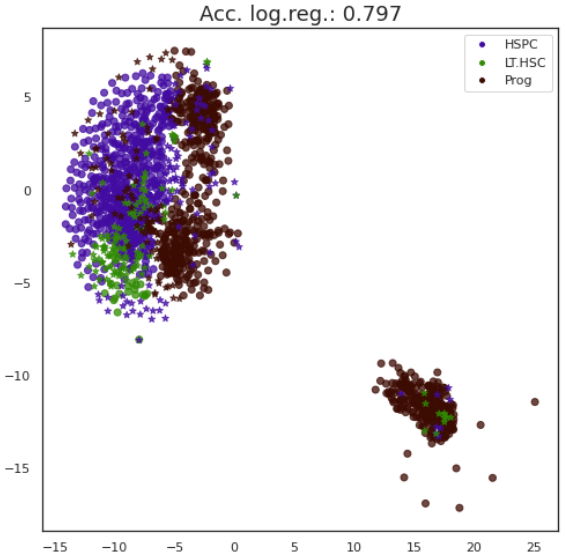


Figure D.21: t-SNE results of the Nestorowa data-set. An accuracy of 0.797 was measured after performing a logistic regression with a 5-fold cross validation. Colours indicate different cell types. Data-points that were predicted correctly by the logistic regression are plotted with dots, incorrect predictions are plotted with stars.

Bibliography

[1] Ashraful Haque, Jessica Engel, Sarah A Teichmann, and Tapio Lönnberg. A practical guide to single-cell RNA-sequencing for biomedical research and clinical applications. *Genome medicine*, 9(1):1–12, 2017.

[2] Ehud Shapiro, Tamir Biezuner, and Sten Linnarsson. Single-cell sequencing-based technologies will revolutionize whole-organism science. *Nature Reviews Genetics*, 14 (9):618–630, 2013.

[3] Fuchou Tang, Catalin Barbacioru, Yangzhou Wang, Ellen Nordman, Clarence Lee, Nanlan Xu, Xiaohui Wang, John Bodeau, Brian B Tuch, Asim Siddiqui, et al. mRNA-seq whole-transcriptome analysis of a single cell. *Nature methods*, 6(5):377–382, 2009.

[4] Qiaolin Deng, Daniel Ramsköld, Björn Reinius, and Rickard Sandberg. Single-cell RNA-seq reveals dynamic, random monoallelic gene expression in mammalian cells. *Science*, 343(6167):193–196, 2014.

[5] 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.

[6] Karl Pearson. LIII. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[7] Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

[8] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[9] Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[10] Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel WH Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W Newell. Dimensionality reduction for visualizing single-cell data using UMAP. *Nature biotechnology*, 37(1):38, 2019.

[11] Christopher M Bishop and Michael E Tipping. A hierarchical latent variable model for data visualization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):281–293, 1998.

[12] Philip van Kuiken. Hierarchical visualization of single cell RNA-seq data. Master's thesis, Vrije Universiteit Amsterdam, the Netherlands, 2017.

[13] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

[14] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

[15] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[16] Michael E Tipping and Christopher M Bishop. Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482, 1999.

[17] Stijn Van Hoey, Johannes van der Kwast, Ingmar Nopens, and Piet Seuntjens. Python package for model structure analysis (pySTAN). *EGUGA*, pages EGU2013–10059, 2013.

[18] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[19] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1), 1970.

[20] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

[21] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[22] Matthew D Hoffman and Andrew Gelman. The no-U-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[23] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.

[24] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[25] Luke Zappia, Belinda Phipson, and Alicia Oshlack. Splatter: simulation of single-cell RNA sequencing data. *Genome biology*, 18(1):174, 2017.

[26] Spyros Darmanis, Steven A Sloan, Ye Zhang, Martin Enge, Christine Caneda, Lawrence M Shuer, Melanie G Hayden Gephart, Ben A Barres, and Stephen R Quake. A survey of human brain transcriptome diversity at the single cell level. *Proceedings of the National Academy of Sciences*, 112(23):7285–7290, 2015.

[27] Sonia Nestorowa, Fiona K Hamey, Blanca Pijuan Sala, Evangelia Diamanti, Mairi Shepherd, Elisa Laurenti, Nicola K Wilson, David G Kent, and Berthold Göttgens. A single-cell resolution map of mouse hematopoietic stem and progenitor cell differentiation. *Blood, The Journal of the American Society of Hematology*, 128(8): e20–e31, 2016.

[28] Aaron TL Lun, Karsten Bach, and John C Marioni. Pooling across cells to normalize single-cell rna sequencing data with many zero counts. *Genome biology*, 17(1):75, 2016.