Dimensionality Reduction of single-cell RNAseq data

Simon E. Streib
Department of Biosystems Science and Engineering
ETH Zürich
Basel, 4058
`sstreib@student.ethz.ch`

Abstract

In this study we investigate the ability of autoencoders to reduce the dimensionality of single-cell transcriptome data while retaining cell-type-specific characteristics. For this, the latent layers of multiple autoencoders were extracted after training and compared with more traditional dimensionality reduction techniques such as PCA, LSA, ICA, t-SNE, and UMAP. The dimensionality-reduced embeddings were then evaluated on two tasks: i.) an unsupervised clustering task using k-means and agglomerative clustering and ii.) a supervised classification task using a random forest and support vector machine with the original cell type as label. Our results show, that the autoencoders fail to exceed most of the established baselines. Instead, it was UMAP that performed best in the clustering task, and LSA which performed best in the classification task. The most sophisticated autoencoder (DCA) performed especially bad despite having been built for this type of data, meaning it might not have been set up optimally. Other, less complex autoencoders (BCA, SCA) produced better embeddings that scored more or less on par with most of the baselines.

1 Motivation

The main goal of this project is to perform an objective evaluation of dimensionality reduction techniques on single-cell data with a special focus on autoencoders. Neural networks such as autoencoders have the potential to model any function [1] and can therefore emulate or even surpass conventional dimensionality reduction techniques. In an autoencoder, the latent layer must learn the characteristics of the data in order to reconstruct it, and using this layer as a form of dimensionality-reduced data could rival standard dimensionality reduction techniques, provided that the celltype-specific characteristics are captured well enough. The goal of this work is, to see if an autoencoder built by Eraslan et al. [2] and a simple model autoencoder can reduce the dimensionality of a single-cell dataset better than standard techniques.

2 Introduction

2.1 Single-cell transcriptome analysis

Data collection: The principle of RNA sequencing has been used since 2008 in order to measure the transcriptome (set of mRNA) of cells in hopes of gaining information about cell identity and function. [3]. However, doing mRNA analysis on a decent number of cells at a single-cell level has remained a challenge and was linked to very high costs until the introduction of droplet-based microfluidic approaches in 2015, such as inDrops (indexing droplets) [4] and DropSeq [5]. In both these methods, single cells are encapsulated in droplets, lysed and then their mRNAs are captured, reverse-transcribed, added with a barcode and finally sequenced using for example the Illumina

Preprint. Work in progress.

platform [6]. The individual mRNA reads are then mapped to genes on the genome in order to receive information about the strength of expression for each gene in each cell.

This makes it possible to measure the transcriptomes of thousands of genes in up to millions of cells for a relatively low cost. However, this vast data collection leads to extremely large datasets, which cannot be interpreted without extensive bioinformatics analysis [7]. Furthermore, these datasets are often very sparse due to the large number of genes with low or no expression, as well as the inherently low capture chance for each individual mRNA molecule.

Standard analyses of single-cell data There are many standard analysis types for single-cell data and the following list is by no means conclusive. Bioinformatic pipelines often include a clustering step to find similar cells that might correspond to a certain cell type or cell state in the sample. This could be used in order to classify cells. Another very common goal is the detection of differentially expressed genes (DEGs). These are genes that are expressed on a different level in an experimental group compared to their counterparts in a control group, giving hints to the intracellular mechanisms that might be responsible for a specific phenotype. The full set of DEGs can then be compared to previously annotated and defined processes and pathways ("GO terms"), and the significance of their enrichment assessed [7]. Two more very exciting techniques are temporal and spatial alignment of cells. If the analysed tissue contains developing cells, then it is possible to look at all the cells and order them according to their developmental timepoint - if enough cells were captured, then there should be a few cells in each developmental stadium, and a temporal trajectory showing the change in gene expression as a cell develops can be built [8][9]. Similarly, if the dissociated and analysed tissue can be assumed to have a gradient of certain factors, e.g. an embryo during development, then we can also infer the spatial position of dissociated cells based on the expression data, if at least some spatial information of a few genes is available, e.g. through RNA-FISH [10][11][12].

2.2 Dimensionality Reduction

Dimensionality reduction is the transformation of high-dimensional data into a meaningful representation of reduced dimensionality [13], such that similar input objects are mapped to nearby points on the low dimensional manifold. This can offer several advantages. First of all, we might suspect or even know that such a simple manifold exists, in which case we would want to reduce the number of parameters to the minimum number that is needed to account for the observed properties of the data, which is also called the intrinsic dimensionality of data [13]. This may drastically reduce the data size and computational effort, especially for tasks that scale exponentially with the number of dimensions [14].

2.3 Related work

There is a plethora of methods that are being used for the reduction of dimensionality of single-cell data. Many of these rely on preprocessing to isolate the genes that were reliably detected and carry information. The most important technique is Principal Component Analysis (PCA) [15]. It produces a representation of the data through “principal components” (PCs) which are ordered, meaning that the first few PCs are usually sufficient to explain most of the variation within the dataset. There are other linear methods available and in this project Latent Semantic Analysis (LSA) [16] and Independent Component Analysis (ICA) [17] were also used. LSA reduces the workload for very large datasets [18], and ICA imposes independence up to the second order instead of the orthogonality enforced by PCA [17]. However, for complex structures, as might be contained in single-cell data, these linear methods may fail to recognize the proper manifolds. Therefore, many non-linear dimensionality reduction techniques have risen to prominence, and two of the most famous ones were used in this work as well, namely t-Distributed Stochastic Neighbour Embedding (t-SNE) [19] and Uniform Manifold Approximation and Projection (UMAP) [20]. t-SNE was introduced first and revolutionized the presentation of single-cell data, however, it has problems with preserving the global structure since it focuses more on keeping the low-dimensional representation of similar datapoints close to each other than on keeping the highly dissimilar datapoints away from each other. UMAP is generally better than t-SNE since it preserves global structure better and also has the superior runtime and reproducibility [21]. But both are still widely used to this day. A common problem of most non-linear methods is that they do not scale well with the data dimension, and therefore they are usually applied on reduced dimensions obtained by a linear method [22].

Using autoencoders for the analysis of single-cell transcriptome data has not become a standard method yet. The "Denoising Count Autoencoder" (DCA) by Eraslan et. al [2] is an example of applying an autoencoder to single-cell data and serves as the starting point of this project. DCA uses a special model of a loss function which considers zero-inflation, a property of highly sparse data in which a zero in the data may have been produced by two possible causes: either a gene was not being expressed in a cell ("a true zero") or the gene was expressed but not captured by the library preparation protocol ("a dropout"). DCA was intended as a denoising tool and not as a means for dimensionality reduction, however like any autoencoder, it produces a reduced bottleneck layer which can be regarded as a low-dimensional representation of the data, and which is compared to the other reduced data embeddings in this work.

3 Methodology

3.1 Description of the pipeline

An overview of the pipeline can be seen in Figure 1. The input data is collected and preprocessed, which includes i.) the removal of outliers, ii.) feature selection and iii.) the transformation into a normalized logarithmic format. Step iii is only conducted if the data is input into the baselines, it is not conducted if the data is fed into the autoencoders, as they were built similar to DCA and expect integer count data.

Afterwards, the baselines as well as autoencoders run on the data and produce different dimensionality reduced representations. Due to the nature of certain dimensionality reduction techniques, not all reduced forms of data have the same number of dimensions.

After dimensionality reduction, each group of data is assessed to find out how well it managed to preserve the intrinsic differences between the sample groups. For this, two tasks are conducted: a classification task and a clustering. The classification task is done using three different train-test splits, hence the whole pipeline must run four times in total, three times for the three different test-train splits, and once without any split for the clustering task. When using a split, the dimensionality reduction model gets trained using the training data, and then training and testing data are both transformed using the this model. For t-SNE and DCA, this is not possible, as no intermediate dimensionality reduction model is built and instead the dimensionality reduction happens directly on the input data. Because of this, no proper testing data is available, and these techniques are only evaluated using the clustering.

3.2 Input Data

The input data was downloaded from the official 10X Genomics public single-cell gene expression datasets. [1]. In the section *single-cell 3' Paper: Zheng et al. 2017 / Cell Ranger 1.1.0* there are many available datasets, of these, the 10 datasets directly corresponding to an individual cell type (see Table 1) were downloaded. The downloaded data was already filtered, i.e. background and non-cellular barcodes were already removed.

The data is extremely sparse and could easily be saved as small .mtx files, despite its massive dimensions of 95'000 cells $\times$ 33'000 genes. In Figure 2 a histogram illustrating the numbers of genes and transcripts that were detected in each cell are displayed, and the same information on a per-gene basis is shown in Supplementary Figure B.1. As can be seen, in most cells, less than a 1000 genes are captured, and the total number of captured transcripts is often less than 2000, which is a minuscule amount. If we expect a "standard" mammalian cell to have about 360'000 mRNA molecules [23], this corresponds to less than 0.5% of the transcriptome, which is very bad by nowadays standards, where 20-30% of the transcriptome can be captured [24] [25].

By looking at the average number of non-zero genes and the number of detected transcripts, we can calculate the strength of expression. As seen in Supplementary Figure B.2, the average number of transcripts per gene was only about 2.5 which is very little for statistical analysis. Looking at the bottom part of the plot, it gets even more extreme. By looking at the expression per gene instead of averaged per cell, we can see that almost all genes are expressed less than that, with a few cells being expressed much stronger ($\sim$35 transcripts per cell on average for the strongest).

[1] https://support.10xgenomics.com/single-cell-gene-expression/datasets

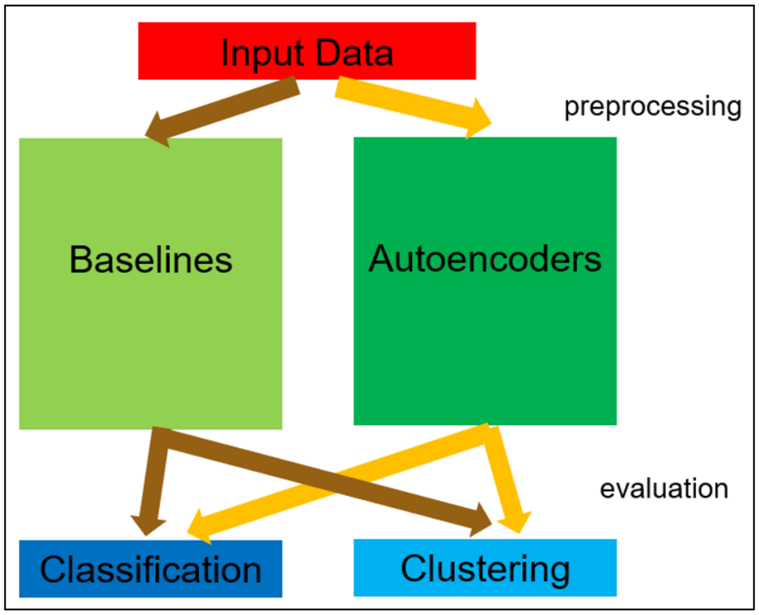
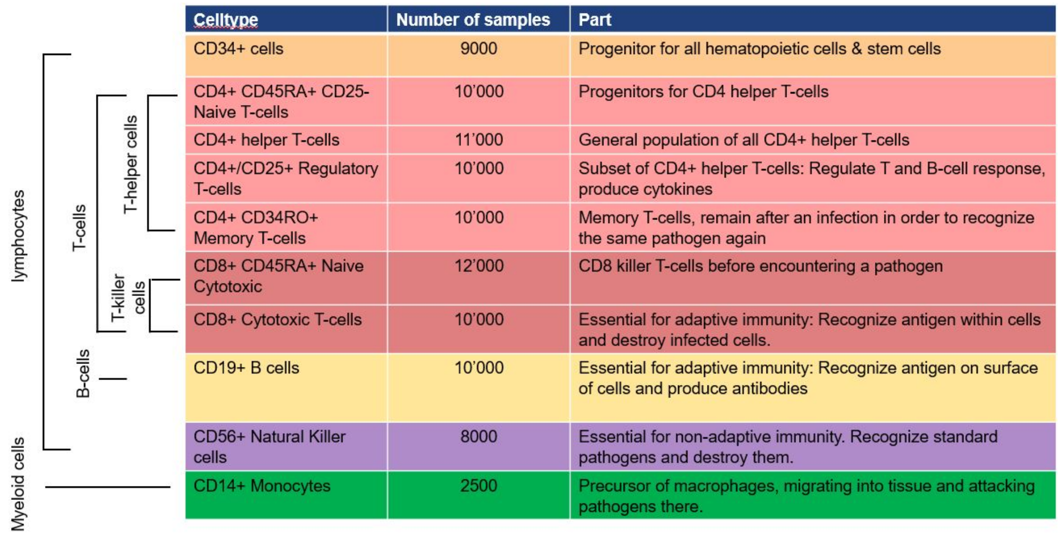


Figure 1: A simple overview of the project. The same data is preprocessed, once as count data for the autoencoders, and once for the baselines. After the dimensionality reduction, all the reduced forms of data are evaluated by classification and clustering.

Table 1: Information about the different types of cells and their biological meaning. The majority of the data is comprised of different types of T-cells, meaning the cell types discussed in this work are often similar to each other and not trivial to distinguish.


lymphocytes
T-cells
T-helper cells
T-killer cells
B-cells
Myeloid cells

Celltype	Number of samples	Part
CD34+ cells	9000	Progenitor for all hematopoietic cells & stem cells
CD4+ CD45RA+ CD25- Naive T-cells	10'000	Progenitors for CD4 helper T-cells
CD4+ helper T-cells	11'000	General population of all CD4+ helper T-cells
CD4+/CD25+ Regulatory T-cells	10'000	Subset of CD4+ helper T-cells: Regulate T and B-cell response, produce cytokines
CD4+ CD34RO+ Memory T-cells	10'000	Memory T-cells, remain after an infection in order to recognize the same pathogen again
CD8+ CD45RA+ Naive Cytotoxic	12'000	CD8 killer T-cells before encountering a pathogen
CD8+ Cytotoxic T-cells	10'000	Essential for adaptive immunity: Recognize antigen within cells and destroy infected cells.
CD19+ B cells	10'000	Essential for adaptive immunity: Recognize antigen on surface of cells and produce antibodies
CD56+ Natural Killer cells	8000	Essential for non-adaptive immunity. Recognize standard pathogens and destroy them.
CD14+ Monocytes	2500	Precursor of macrophages, migrating into tissue and attacking pathogens there.

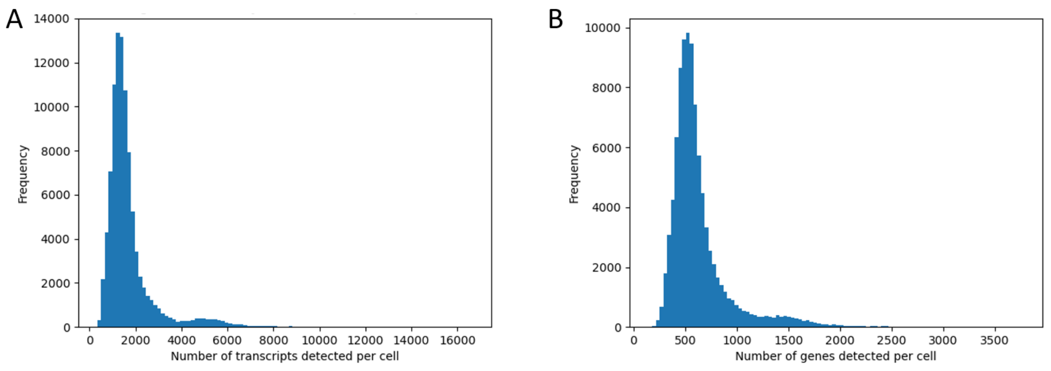


Figure 2: Histograms illustrating the number of transcripts (A) and the number of genes (B) that were detected in each cell. Both numbers are considerably low.

3.3 Preprocessing

Preprocessing was done in Python using ScanPy, and following a tutorial which is presented by ScanPy [26], as well as a tutorial received as part of Prof. Dr. Treutlein's lecture "single-cell Technologies" [27]. Both of these tutorials are in fact based on a different tutorial, originally presented by Satija Lab and implemented in R using Seurat [28] [29].

While the standard procedure is to log-transform and normalize the data, the Deep Count Autoencoder only works with count data because it has an internal normalization step. As such, the preprocessing was divided into two different pipelines: One time, the full preprocessing was done, which supplied the data for all the baselines, and a parallel preprocessing would prepare count data for the autoencoders. All cut-off values and parameters were chosen identically and are listed in the figure description of Supplementary Figure B.3. In order to guarantee the same selection of highly variable features, the preprocessing script for the autoencoders has an internal duplication of the data, as the highly variable genes are usually selected on normalized logarithmic data. The chosen genes were then selected in the other, non-normalized and non-logarithmized copy. Furthermore, the preprocessing was also the step that defined a 25% fraction of the data as test data by creating a separate text file which indicates for each line in the data whether it belongs to the test or train set without actually separating them.

3.4 Methods

Version numbers of the used software are listed in Supplementary Table B.1.

3.4.1 PCA

For PCA, a total of 100 principal components (PCs) were calculated. Optimization efforts were undertaken to see how the number of PCs influences the results, however, neither k-means clustering nor classification with a random forest improved when using more than 20 PCs. Interestingly, the information captured by 100 PCs corresponds to only about 16% of the total variance in the data, as illustrated by Supplementary Figure B.4. Figures showing the optimization result are included in Appendix C (Figure C.1).

3.4.2 LSA

For LSA, a total of 100 components were calculated. Optimization efforts were undertaken to see how the number of components influences the results, however, similarly to PCA, about 20 components would have been enough. Additional plots are found in the appendix (Figure C.2).

3.4.3 ICA

ICA was not run on the original data but instead run on the data reduced by PCA, taking the first 20 PCs as input dimensions and producing 20 output dimensions. The number of input dimensions was determined by an optimization run, which showed that the evaluation stopped improving after ∼15 input dimensions. This figure can be found in the appendix (Figure C.3).

3.4.4 t-SNE

t-SNE scales badly with the number of features (Complexity $O(n^2)$) [30], and was therefore also run using PCA-reduced data as input. 17 PCs were chosen as this seemed to produce the best clustering scores, however, there is a lot of randomness involved as the conducted clustering was run with an early version of the script that did not yet feature multiple repetitions of the clustering. For details, please consult Appendix Figure C.4. The number of output dimensions was 2.

3.4.5 UMAP

UMAP received 20 input dimensions of the PCA-reduced data and produces 2 output dimensions. Optimization effort showed, that no improvement was achieved by using more than 15 input dimensions, however, for a high number of input dimensions (80+), the quality of the data even seemed to diminish. For details, please consult Appendix Figure C.5.

3.4.6 DCA: Denoising Count Autoencoder

DCA was run directly in bash, following the readme on their GitHub page [31]. No additional parameters were passed. To make DCA work with the rest of the pipeline, barcode and gene labels had to be added to the count data as well. Please note, that DCA was using different version numbers of the software dependencies, which are listed in Supplementary Table B.1. While the full optimization of DCA as intended by its authors is out of the scope of this work, the rough autoencoder types as implemented by Eraslan et. al. were compared quantitatively, and the results are shown in Supplementary Figure C.10.

3.4.7 SCA: Simplified Count Autoencoder

SCA is a modification of the publicly available DCA code. For building the pipeline, the inner workings of DCA had to be analysed in detail, and SCA was built up piece by piece from fragments of the DCA code. This allowed for better control of the process, allowing the adaptation and modification of parameters beyond what Eraslan et al. originally implemented. The end result allows the running of SCA in all the major configurations that were also contained in DCA. These can generally be divided into the ones based on the negative binomial (nb) or the zero-inflated negative binomial (zinb) model. The results presented for SCA are however not based on either of these two configurations, but instead uses a much simpler model that Eraslan et. al. referred to as "poisson autoencoder". It has no input or hidden dropout, uses "glorot_uniform" regularizer, "relu" activation function, "RMSprop" optimizer, and the regularizers were run with a patience of 15 for the learning rate reduction and 35 for an early stop. If no early stop is reached, the autoencoder runs for 300 epochs.

3.4.8 BCA: Basic Count Autoencoder

BCA was another autoencoder set up with the goal of creating a very simple autoencoder model and see if it could rival the much more advanced DCA. Even the most basic implementation of an autoencoder in Keras allows for the manipulation of important parameters, and these were optimized to get the best possible result. Each parameter was optimized independently from the others and interactions were not considered. The varied parameters are: Activation function, optimizer, loss function, and the overall structure of the autoencoder, henceforth referred to as layer structure. The results of these optimizations are listed in Appendix Figures C.6, C.7, C.8 and C.9 respectively. From these results, the following parameters were chosen: "poisson" loss function, "relu" activation function, "Adam" optimizer, and the layer structure was kept analogously to DCA ("default"). BCA runs for 500 epochs at most, with a batch size of 256. The validation fraction of the data was 0.1, and the training data was shuffled before each epoch. The same callbacks were used as for SCA, namely reducing the learning rate on plateau (15 epochs) and early stopping on plateau (35 epochs). The initial learning rate was 0.001.

3.4.9 Classification Task

The quality of dimensionality reduction was evaluated using a classification task. If the cell-specific features are still contained within the reduced form of the data, then a classifier should still be able to be trained on recognizing a certain cell group.

Strategy As previously stated, the data was routinely separated into a training and test set during the preprocessing. For the classification task, the preprocessing ran three times on the downloaded data, producing three different sets of data that were each sent through the pipeline individually. The cell type for each cell is known, and the following classifiers were then trained on the training data using the known label and evaluated on the test data. This means, that it uses the same test split that was already used in the dimensionality reduction step. The classifier then reports the accuracy for each split, and the average of the three accuracies is reported with error bars. At first, binary classifiers were used (see Appendix A.1) but eventually replaced with the following multiclass classifiers. Furthermore, within each split, the accuracy was also calculated on a per-celltype basis to see which cell types were harder to differentiate.

It is important to mention that the reporting of the accuracy as an absolute value is not ideal, as it does not take into context "how wrong" the classification was. Classifying a CD4+/CD25+ regulatory T-cell as a CD4+ helper T-cell is not that wrong, since they can in fact be regarded as a subset thereof and are at least prone to showing a lot of similarities. On the other side, misclassifying one of them as a B-cell or even Monocyte would be a much graver mistake. However, in this pipeline, both errors have the same effect on the score.

Random Forest Random Forest had already been implemented in the binary one-vs-all pipeline (Appendix A.1), but since the scikit-learn implementation [32] supports multiclass it was simply reimplemented with multiple classes. Many of the parameters and possible restrictions were tested, however, no advantage was found in moving away from the defaults. The ideal number of trees in the forest was evaluated in more detail, and while about 30 trees seemed to be sufficient for each dataset (see Appendix Figure C.11), the default value of a 100 trees was kept.

SVM As an additional classifier, a Support Vector Machine (SVM) was implemented, as the sk-learn version of an SVM can solve multiclass problems. All default parameters were kept.

3.4.10 Clustering Task

The clustering task was built with a similar intention than the classification. If the cell types have intrinsic differences, then a dimensionality reduction should be able to capture that, and cells of a similar type should be closer to each other than to cells of a different type.

Strategy For the clustering, no train-test split was used. Instead, the data was preprocessed and piped through the pipeline a fourth time independently of the runs which were evaluated through classification. This fourth preprocessed dataset was then dimensionality reduced with all techniques, and all data was treated like the train data from the classification splits during their dimensionality reductions. Afterwards, two clustering algorithms were run on each reduced dataset. K-means clustering and hierarchical clustering are the most common clustering algorithms for single-cell studies [7], hence these were used here as well. Since we knew the real labels of each cell, we were able to assess the performance of the clustering algorithms, resulting in a semi-supervised strategy in which the clustering was still conducted in an unsupervised manner. Each cluster then received a label that corresponds to the label of the cell type that was the majority within each cluster. This allowed for quantification by calculating the purity and recall of each cluster, and also allows for the evaluation of the number of clusters with unique labels ("how many cell types have we found") and bad/ outlier clusters with less than 50 cells.

For this reason, the reported numbers have to be interpreted with caution. Additionally, unlike for the classification where three folds were used, there was only one fold in the clustering, meaning a non-deterministic dimensionality reduction can be prone to producing an extraordinary good or especially bad dimensionality reduction by pure chance. For future projects, the establishment of at least three folds is advisable. Please note that the clustering itself was still done with repetitions: Additionally to the multiple initializations that most clustering algorithms do anyway (n_init parameter in sklearn, default "10"), each clustering was repeated a number of times (usually 50). Certain optimizations were run before introducing the repetitions to the clustering. Therefore, to indicate if repetitions were done, the number of repetitions will be shown in parentheses under each label in the graphs.

Since the reporting of precision and recall is often unclear, especially in the presence of mini-clusters that easily reach scores of 1 or 0, the harmonic mean or F1-score was reported instead. Additionally, the F1-score of each cluster was normalized to the cluster size, meaning the F1 score of each cluster was multiplied with its cluster size and divided through the total number of cells, and then all the scores were added up, resulting in something like a weighted average. This aimed to reduce the influence of tiny outlier clusters which would otherwise heavily skew the results. As an additional second metric, the Normalized Mutual Information (NMI) [33] was used.

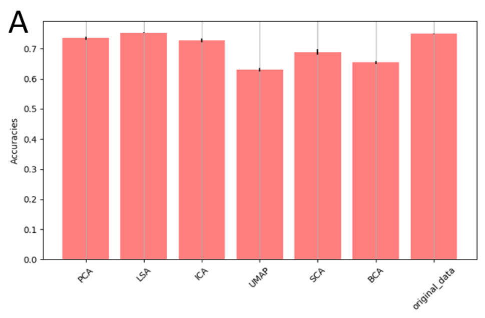
K-means clustering K-means clustering was implemented with sk-learn. For k-means clustering one needs to define the parameter k, which indicates how many clusters should be created. 10 was the true number of cell types in the data, however not all 10 clusters were possible to be found. Some cell types are so similar that their clusters overlap, making it unlikely that a clustering algorithm can differentiate them. Given this motivation, a wide range of k-values was tried out for all the reduced datasets, and 8 seemed to be the average number of clusters retained in the reduced form for all dimensionality reduction techniques. For the original data however, the true value of 10 was finding very good results, indicating that the differences of the cell types might still be present before the dimensionality reduction. For more information please refer to Appendix Figure C.12. In the end, for comparability and simplicity, it was decided to use the same k parameter of 10 for all techniques.

Hierarchical clustering There are many hierarchical clusterings algorithms available. In the end, scikit-learn's Agglomerative Clustering using "euclidean" affinity and "ward" linkage was chosen, as staying consistent with using sklearn implementations seemed favourable, and allowed to pass the number of clusters (10) as a parameter similar to k-means clustering, instead of the standard approach of defining a distance threshold parameter.

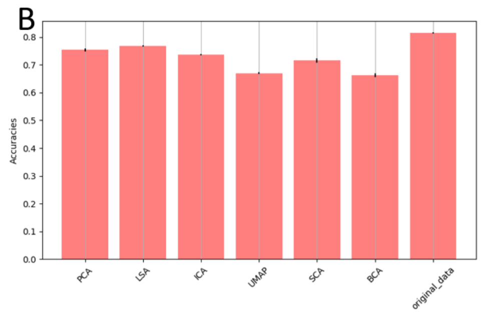
DBScan Both, k-means and this implementation of hierarchical clustering have the weakness of requiring the number of clusters *a priori*. An alternative approach with a density-based clustering (here: DBScan) was tried. Unfortunately, this did not work out as explained in Appendix A.2, and was removed.

4 Results

The results of the main pipeline, the comparison between the different dimensionality reduction techniques is shown in Figures 3, 4 and 5.



	PCA	LSA	ICA	UMAP	SCA	BCA	original_data
Average	0.734835	0.752743	0.726977	0.629942	0.688277	0.654171	0.749539
Stdev(P)	0.004574	0.000571	0.005608	0.00576	0.009367	0.004652	0.001559



	PCA	LSA	ICA	UMAP	SCA	BCA	original_data
Average	0.75368	0.767653	0.736342	0.669973	0.715243	0.662467	0.814547
Stdev(P)	0.005226	0.001763	0.001859	0.002422	0.008228	0.006769	0.001331

Figure 3: The result of the main pipeline using the classification task averaged over three splits for random forest (A) and SVM (B). The errorbars are indicated. Results for t-SNE and DCA and denoised reconstructed data are not available, as those techniques cannot handle the split.

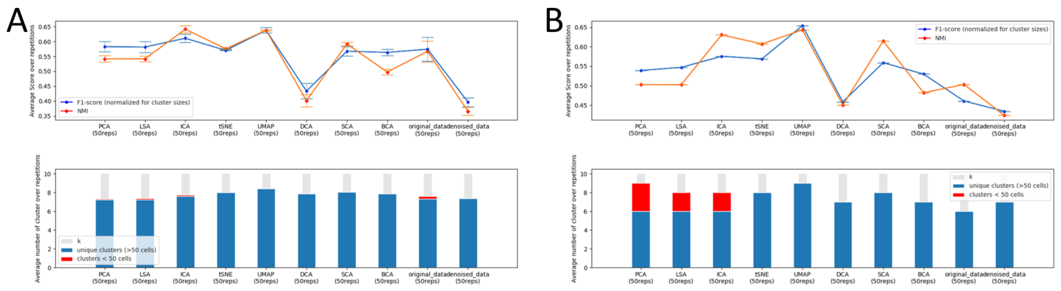


Figure 4: Result of the main pipeline evaluated using k-means (A) as well as hierarchical clustering (B). The normalised F1-score is the (size)-weighted average of the F1-scores of each cluster. NMI refers to the Normalized Mutual Information [33]. Regarding the barplot, a cluster is considered unique, if there are no other clusters with the same majority celltype. If more than one cluster with the same majority celltype exists, then only one of these cluster will be counted as unique. Clusters with less than 50 cells were not considered for this.

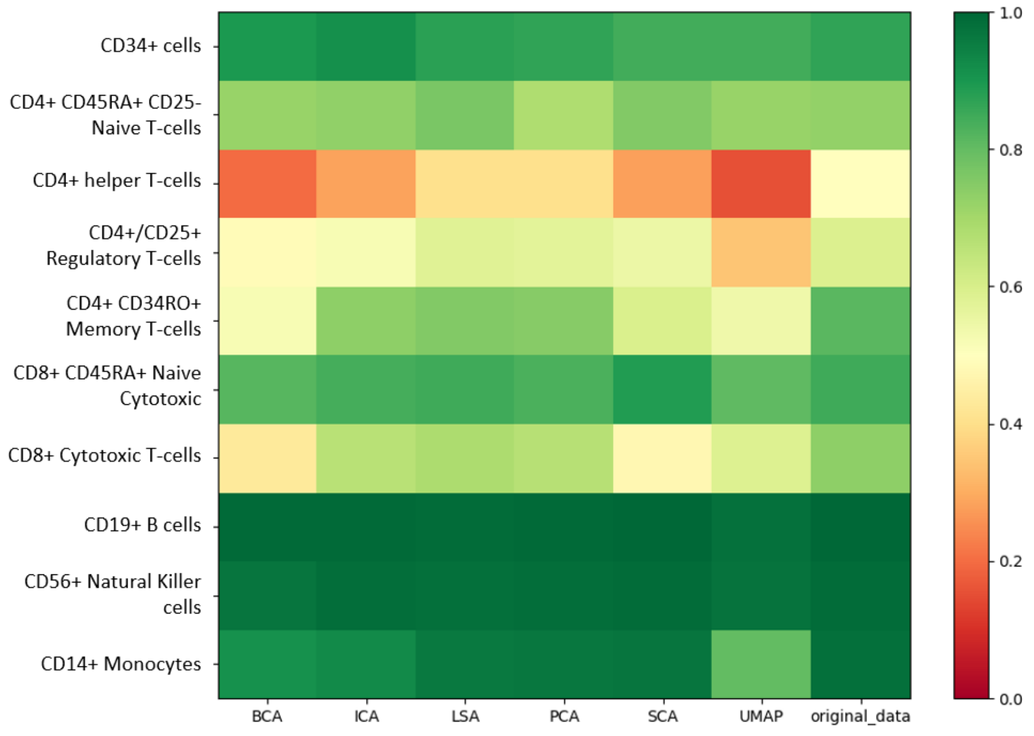


Figure 5: The average classification accuracy per cell type and technique. For this, the average accuracy between the random forest and SVM was taken.

4.1 Clustering Result

In this work, the autoencoders were not shown to surpass the baselines. DCA (run with defaults) reached the worst score in the clustering task, which is surprising since it had been specifically built for this type of data. In contrast, both other autoencoders (BCA and SCA) seem to perform better. Nevertheless, all autoencoders fail to reach scores as good as the baselines. Of these, ICA and UMAP work best, with SCA and t-SNE behind them. PCA, LSA and BCA perform in the lower range and DCA is the worst, both, in dimensionality reduced as well as reconstructed form. The non-reduced original data clusters is interesting to analyse, for hierarchical clustering it performs very bad, showing the usefulness of dimensionality reduction to obtain a less noisy lower dimensional representation of the data. However in k-means clustering it manages to get a healthy score itself and only a few dimensionality reduction techniques are able to further improve the score.

Comparing the clustering strategies shows, that hierarchical clustering scores are more consistent with themselves than k-means clustering, as indicated by the almost invisible error bars. However, the barplots indicate, that k-means clustering might still be the better approach for this kind of data since it is able to find more uniquely labelled clusters and hence cell types. Hierarchical clustering not only finds less uniquely labelled clusters, but also finds more "outlier-clusters" with a size smaller than 50 cells (red part of bars). But despite this, the scores of both hierarchical and k-means clustering are surprisingly close to each other.

4.2 Classification Result

Comparing clustering to the classification task shows an interesting switch. The linear methods (PCA, LSA and ICA) lead the classification scores, but were only in the medium range for the clustering. In contrast, the non-linear method of UMAP, which outperforms all the other techniques in the clustering task, has the worst score. Unfortunately, the only other non-linear baseline (t-SNE) could not run the classification task for its inability of properly integrating the train-test split. The autoencoders (BCA and SCA) perform on the lower end as well, with SCA generally outperforming BCA but both being worse than the linear baselines. Surprisingly, none of the dimensionality reduction techniques perform better than just doing the classification on the high-dimensional data. LSA is the only technique which might be at least equally good, but also not really better.

Overall, SVM and random forest agree on all scores, with the SVM generally reaching higher scores. BCA is the only one with a noteable difference, which is interesting as BCA was also the one which shows the most noteable difference between NMI and F1-score in the clustering.

In Figure 5, an additional overview of the individual cell types is given. CD14+ Monocytes, CD56+ Natural Killer cells and CD19+ B cells reach very high scores, meaning they can easily be differentiated from the other cell types. CD34+ cells, the general progenitor cells are also very distinct, and so are the CD4+ CD45RA+ Naive Cytotoxic cells. Of the rest of the cell types, some show slightly lowered scores (CD4+ CD45RA+ CD25- Naive T-cells, CD4+ CD34RO+ Memory T-cells, CD8+ Cytotoxic T-cells) while others are considerably more difficult to classify (CD4+ helper T-cells, CD4+/CD25+ Regulatory Cells).

5 Discussion

Clustering Clustering worked well and illustrates how a clustering can be improved after reducing the dimensionality of single-cell data. Not only is the runtime much better when clustering over reduced data, but the scores reached do in many cases surpass the ones from non-reduced data. However, it largely depends on which dimensionality reduction technique is applied. Generally, UMAP, t-SNE and ICA seem to be the best ones. It is unclear why the autoencoders fail to improve the clustering. For DCA we can see, that the reconstructed data (denoised_data) performs even worse, which makes it likely that this autoencoder was not trained properly. Most probably, DCA was not set up correctly, and if more time were available, it would be advisable to further dig into why DCA performed so badly and what parameters might mitigate this. It is interesting to mention that in the original paper [2] DCA was also run on data by Zheng [34], albeit not the exact same dataset that was used here. They used a very special 2-node bottleneck configuration but produce a nice clustering, showing that DCA can work well. Unfortunately, they do not describe their set up in detail, and while an effort was made to reconstruct their configuration, the results did not look as good as Eraslan et. al.. If more resources were available, these problems might have been fleshed out.

Comparing the k-means clustering to hierarchical clustering, one notices that the scores mostly agree, however there is a huge difference for the original_data, where the hierarchical clustering did not seem to work very well. This could be an outlier or hierarchical clustering is unable to cope with the extreme high dimensionality of the data. Due to the lack of other high dimensional clusterings this cannot be evaluated, but it seems reasonable to assume that for some applications the high dimensionality might be more of a problem than for others.

Classification One problem of the classification task is, that unlike the clustering, scores of the reduced data are generally lower than the score of the original_data. This is unfortunate, as it was hoped that reducing the dimensionality would help crystallize out the relevant information, however, for some reason the classification task is not improved. The seeming trade-off between a good clustering and classification score is also difficult to evaluate, as the truth might be concealed by other effects. One possibility is, that having more dimensions in the reduced form may be an advantage for the classification task, while being a hindrance for the clustering. The number of components for PCA and LSA were both 100 and they perform well in the classification, while UMAP with only 2 components doesn't, even though it performs great in the clustering. The autoencoders with 32 reduced dimensions perform in the mid-range in both cases. An outlier to this theory would is ICA, which produces "only" 20 reduced dimensions, and still performs about as well as PCA and LSA. In the end, it is difficult to say whether the number of reduced dimensions or the nature of the dimensionality reductions plays a bigger role here. For future projects, I would definitively recommend to keep the number of reduced dimensions constant wherever this is possible.

Regarding the celltype-specific analysis, CD4+ helper T-cells are the most difficult to differentiate from the rest of the cells, which is not surprising as they were assumed to be very close to the naive T-cells, regulatory T-cells and memory T-cells, which all have a slightly lowered accuracy (Table 1). B-cells, CD-14+ monocytes, CD34+ cells and CD56+ natural killer cells are all very well distinguished from each other as expected. Interestingly however, the naive cytotoxic cells are much easier to classify than their more differentiated counterparts, the CD8+ cytotoxic T-cells. This is counter-intuitive, as the naive cytotoxic cells would be considered closer to the rest of the naive T-cells and CD4+ cells than the mature ones. The reason, why the mature ones classify worse is not known. One possibility could be, that the cytotoxic CD8+ T-cells mature in a path similar to the maturation of the CD4+ helper cells, with the end result of the two being similar in some aspects (e.g. both being "primed") and therefore making the differentiation hard again. This is however pure speculation, as it was not evaluated which cells a cell type gets most often misclassified as. One needs to keep in mind that cell typing is a complex subject, and treating each cell type as an identical type of class with the same broadness of definition, as it was done here, is dangerous and may obscure results.

Conclusion Unfortunately, autoencoders did not prove to be a better alternative to standard dimensionality reduction in single-cell transcriptomics. The results did however vary greatly depending on the task at hand, therefore it is not impossible that autoencoders might be very well suited for a task not tested here. Even for clustering tasks it cannot be excluded, since both clustering algorithms used here were based on the same constraint of finding exactly 10 clusters, another approach might produce a completely different result. Furthermore, it is very hard to objectively evaluate the results. As mentioned in sections 3.4.9 and 3.4.10, both evaluation approaches do have problems when quantitatively determining the goodness of their results, and finding a completely true and unbiased metric is almost impossible. Therefore, we believe that autoencoders might have a future in single-cell transcriptomics, despite the results presented here.

References

[1] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

[2] Gökcen Eraslan, Lukas M Simon, Maria Mircea, Nikola S Mueller, and Fabian J Theis. Single-cell rna-seq denoising using a deep count autoencoder. *Nature communications*, 10(1):1–14, 2019.

[3] Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature methods*, 5(7):621–628, 2008.

[4] Allon M Klein, Linas Mazutis, Ilke Akartuna, Naren Tallapragada, Adrian Veres, Victor Li, Leonid Peshkin, David A Weitz, and Marc W Kirschner. Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell*, 161(5):1187–1201, 2015.

[5] Evan Z. Macosko, Anindita Basu, Rahul Satija, James Nemesh, Karthik Shekhar, Melissa Goldman, Itay Tirosh, Allison R. Bialas, Nolan Kamitaki, Emily M. Martersteck, John J. Trombetta, David A. Weitz, Joshua R. Sanes, Alex K. Shalek, Aviv Regev, and Steven A. McCarroll. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, 161(5):1202 – 1214, 2015.

[6] Illumina. *Illumina sequencing platforms*, accessed January 24, 2021. `https://www.illumina.com/systems/sequencing-platforms.html`.

[7] Clarissa M Koch, Stephen F Chiu, Mahzad Akbarpour, Ankit Bharat, Karen M Ridge, Elizabeth T Bartom, and Deborah R Winter. A beginner's guide to analysis of rna sequencing data. *American journal of respiratory cell and molecular biology*, 59(2):145–157, 2018.

[8] Barbara Treutlein, Doug G Brownfield, Angela R Wu, Norma F Neff, Gary L Mantalas, F Hernan Espinoza, Tushar J Desai, Mark A Krasnow, and Stephen R Quake. Reconstructing lineage hierarchies of the distal lung epithelium using single-cell rna-seq. *Nature*, 509(7500):371–375, 2014.

[9] J Gray Camp, Farhath Badsha, Marta Florio, Sabina Kanton, Tobias Gerber, Michaela Wilsch-Bräuninger, Eric Lewitus, Alex Sykes, Wulf Hevers, Madeline Lancaster, et al. Human cerebral organoids recapitulate gene expression programs of fetal neocortex development. *Proceedings of the National Academy of Sciences*, 112(51):15672–15677, 2015.

[10] Keren Bahar Halpern, Rom Shenhav, Orit Matcovitch-Natan, Beáta Tóth, Doron Lemze, Matan Golan, Efi E Massasa, Shaked Baydatch, Shanie Landen, Andreas E Moor, et al. Single-cell spatial reconstruction reveals global division of labour in the mammalian liver. *Nature*, 542(7641):352–356, 2017.

[11] Nikos Karaiskos, Philipp Wahle, Jonathan Alles, Anastasiya Boltengagen, Salah Ayoub, Claudia Kipar, Christine Kocks, Nikolaus Rajewsky, and Robert P Zinzen. The drosophila embryo at single-cell transcriptome resolution. *Science*, 358(6360):194–199, 2017.

[12] Mor Nitzan, Nikos Karaiskos, Nir Friedman, and Nikolaus Rajewsky. Gene expression cartography. *Nature*, 576(7785):132–137, 2019.

[13] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[14] Luis O Jimenez and David A Landgrebe. Supervised classification in high-dimensional space: geometrical, statistical, and asymptotical properties of multivariate data. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 28(1):39–54, 1998.

[15] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[16] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[17] Pierre Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

[18] scikit learn. *"sklearn.decomposition.TruncatedSVD", scikit-learn 0.23.2*, accessed November 19, 2020. `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html`.

[19] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:857–864, 2002.

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

[21] 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):38–44, 2019.

[22] Emma Pierson and Christopher Yau. Zifa: Dimensionality reduction for zero-inflated single-cell gene expression analysis. *Genome biology*, 16(1):1–10, 2015.

[23] Quiagen. *FAQ: How much RNA does a typical mammalian cell contain?*, accessed November 27, 2020. `https://www.qiagen.com/ch/resources/faq?id=06a192c2-e72d-42e8-9b40-3171e1eb4cb8&lang=en#:~:text=Approximately%20360%2C000%20mRNA%20molecules%20are,account%20for%20less%20than%200.1%25`.

[24] Saiful Islam, Amit Zeisel, Simon Joost, Gioele La Manno, Pawel Zajac, Maria Kasper, Peter Lönnerberg, and Sten Linnarsson. Quantitative single-cell rna-seq with unique molecular identifiers. *Nature methods*, 11(2):163, 2014.

[25] 10x Genomics FAQ. *What fraction of mRNA transcripts are captured per cell?*, accessed January 06, 2021. `"https://kb.10xgenomics.com/hc/en-us/articles/360001539051-What-fraction-of-mRNA-transcripts-are-captured-per-cell-#:~:text=Answer%3ADepending%20on%20the%20Single,transcripts%20are%20captured%20per%20cell."`.

[26] Scanpy. *Preprocessing and clustering 3k PBMCs*, accessed August 23, 2020. `https://scanpy-tutorials.readthedocs.io/en/latest/pbmc3k.html`.

[27] Barbara Treutlein Zhisong He. Tutorial of single-cell rna-seq data analysis in r. Lecture Handout of the course "Single-cell technologies" (636-0121-00L) of ETH Zürich, 04.05.2020.

[28] Rahul Satija, Jeffrey A Farrell, David Gennert, Alexander F Schier, and Aviv Regev. Spatial reconstruction of single-cell gene expression data. *Nature biotechnology*, 33(5):495–502, 2015.

[29] Satija Lab. *"Seurat - Guided Clustering Tutorial"*, compiled October 02, 2020. `https://satijalab.org/seurat/v3.2/pbmc3k_tutorial.html`.

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

[31] gokceneraslan theislab. *Deep count autoencoder for denoising scRNA-seq data*, accessed August 12, 2020. `https://github.com/theislab/dca`.

[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[33] Tarald O Kvålseth. On normalized mutual information: measure derivations and properties. *Entropy*, 19(11):631, 2017.

[34] Grace XY Zheng, Jessica M Terry, Phillip Belgrader, Paul Ryvkin, Zachary W Bent, Ryan Wilson, Solongo B Ziraldo, Tobias D Wheeler, Geoff P McDermott, Junjie Zhu, et al. Massively parallel digital transcriptional profiling of single cells. *Nature communications*, 8(1):1–12, 2017.

[35] Nadia Rahmah and Imas Sukaesih Sitanggang. Determination of optimal epsilon (eps) value on dbscan algorithm to clustering data on peatland hotspots in sumatra. In *IOP Conference Series: Earth and Environmental Science*, volume 31, page 012012. IOP Publishing, 2016.

A Removed Features

A.1 One-vs-all

This section explains the original strategy for evaluation of a dimensionality reduction with a classification task. Since there are many classifiers available for binary classification, a one-vs-all pipeline was first established. The script would loop through all the available cell types, and cells of the current cell type were set to class "1", and all other cell to class "0". Then a classifier would be trained on the training portion of the data, using the same split that was already used in the dimensionality reduction, and finally, the fit would be evaluated using the testing portion of the data. The classifiers implemented were logistic regression, linear discriminant analysis (LDA) and a random forest. The script would then report the average of accuracy and recall over each cell type for each classifier. However, this pipeline was eventually abandoned, as the results were often not clearly interpretable. A big problem was, that the classifiers would often resort to classifying everything as the negative "0" class, due to the imbalance of the sizes of the train/test sets.

A.2 DBScan

DBScan was originally implemented as an alternative clustering algorithm. However, unlike k-means clustering, it was very hard to get good results using DBScan, as it heavily depends on its density parameters. The neighbourhood parameter "eps" defines how close a neighbourhood is for all points, and the density parameter "minpts" decides, how many points a data point needs to have within its neighbourhood, in order to be counted as a core point. These parameters vary greatly for the different data dimensionality reduction techniques. Furthermore, these parameters might even vary within one dataset, as not all the celltype-clusters necessarily need to have the same density. An additional problem is the presence of outliers and outlier clusters, which often lead to the finding of a large number of clusters of only a few cells, obscuring the interpretability of the results.

In order to optimize the density parameters, different approaches were tried. Rahmah & Sukaesih presented a method which was based on the k-nearest-neighbour distances within the dataset. For this, the distances to e.g. the 2nd nearest neighbour are calculated for all points, and afterwards sorted according to length. The resulting graph shows a "knee" at a good cutoff value for what a neighbourhood is supposed to be [35]. Unfortunately, they do not explain why they used the 2nd nearest neighbour, but the difference between the first few is usually not very large. Therefore, their method was used on all the reduced datasets and the plots and eps values are shown in Optimization Figure C.13. However, the received values did not produce good clusterings at all, except for t-SNE and UMAP (not shown).

As an alternative approach, a grid-search was carried out for the PCA-reduced data, trying out a wide range of eps and minpts values (See Appendix Figure C.14). It shows, that there does in fact seem to be something like a sweet spot, however, it does also heavily depend on the minpts parameter, and the value for eps (5-6) did not at all correspond to the value theoretically expected from the k-nearest-neighbour method of Rahmah and Sukaesih (30), and still produced comparably low F1 and NMI scores. Also, the number of intrinsic major clusters found never exceeded 5, proposing that some clusters are very close or even woven into each other, that density-based clustering will always combine them. For this reason and the huge runtime for one such grid-search (10+ days), it was decided to abandon DBScan entirely.

B Supplementary Figures

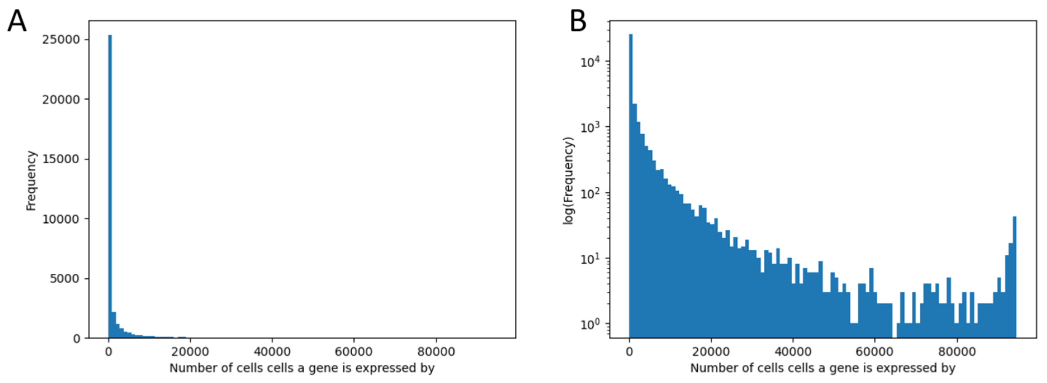


Figure B.1: Histograms illustrating the number of cells each gene was detected in, once as a normal histogram (A), and once, with a log-transformed y-axis (B). Most genes were detected in only a few thousand cells each. The log-transformed histogram helps to see the population of genes that are expressed in more cells, and there is even a slight increase of genes that were detected in all cells and most probably correspond to household genes.

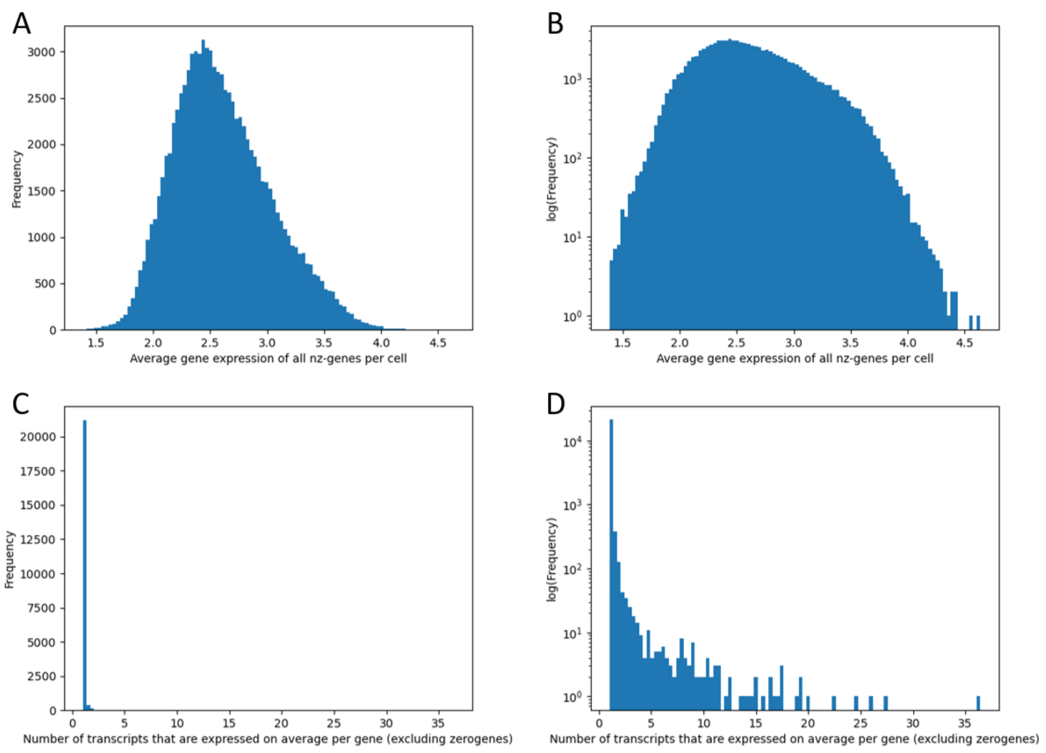


Figure B.2: Histograms illustrating the average gene expression of each gene. A: Average expression of all non-zero genes per cell, i.e. the total number of transcripts divided by the number of non-zero genes for each cell. B: shows the same graph on a log-transformed Y-axis. C: Shows the average expression per gene (total number of transcripts per gene divided by the number of cells in which the gene was detected). Note that zero-genes were already excluded. As can be seen, the vast majority of genes is not or extremely rarely expressed globally. D: The same graph in a log-transformed Y-axis, highlighting that most of the expressed genes have less than 12 transcripts measured per cell in which they are expressed.

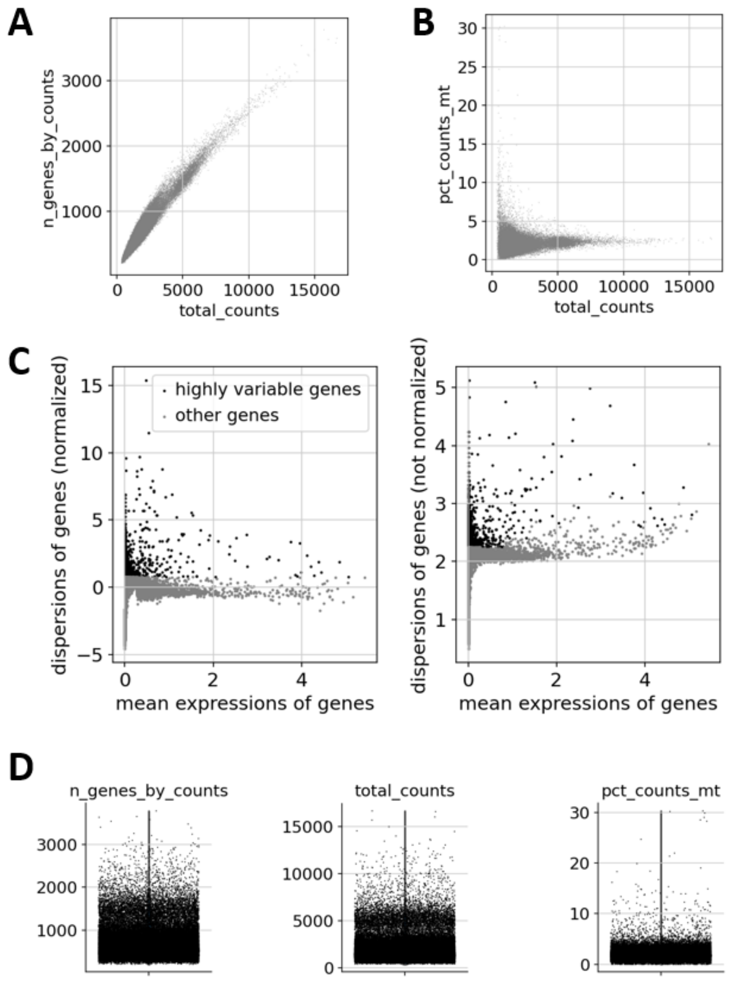
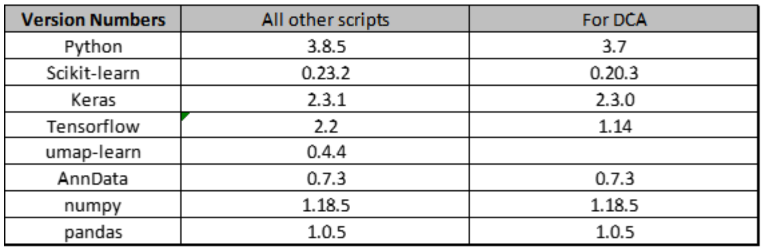


Figure B.3: Information about the preprocessing. A: The number of expressed genes (y-axis) and the total number of transcripts (x-axis) for each cell. B: The percentage of mitochondrial transcripts (y-axis) as a function of the total number of transcripts (x-axis) for each cell. C: The dispersion of genes as a function of their mean expression for each gene. The black genes are the ones with a high variability, which were selected for the analysis. The grey genes were removed from the data. D: Plots showing various properties of the data, from left to right: The number of genes detected per cell, the number of transcripts detected per cell, the percentage of mitochondrial genes. The information of the first two plots is the same shown in Figures 2. Based on these figures, the following parameters were chosen: minimal number of genes expressed in a cell: 200, minimal number of cells a gene must be expressed in: 1, maximal number of genes expressed in a cell: 1500, maximal percentage of mitochondrial transcripts: 5%, number of highly variable features: 2000, test fraction (for classification): 25%.

Table B.1: Version numbers of the used software. DCA was created in an older version of Keras, therefore it was run in a different python environment.



Version Numbers	All other scripts	For DCA
Python	3.8.5	3.7
Scikit-learn	0.23.2	0.20.3
Keras	2.3.1	2.3.0
Tensorflow	2.2	1.14
umap-learn	0.4.4	
AnnData	0.7.3	0.7.3
numpy	1.18.5	1.18.5
pandas	1.0.5	1.0.5

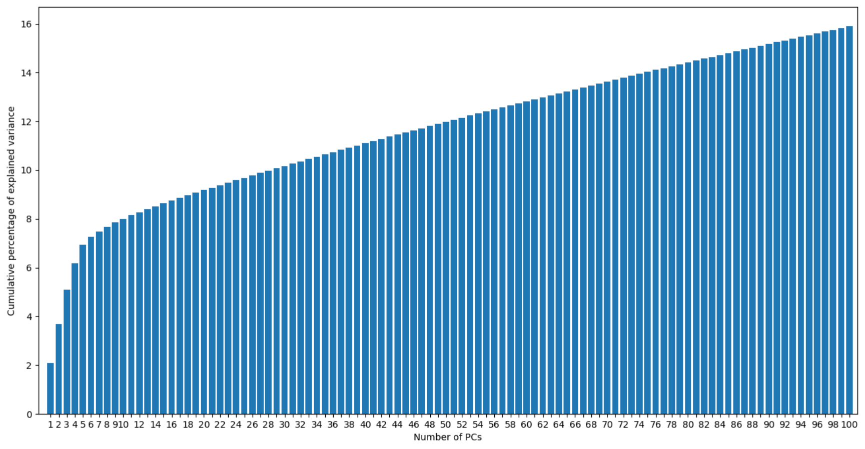


Figure B.4: Knee plot of split 3 of PCA, showing the percentage of variance of the input data that is explained by the first 100 principal components. Please note, that while a 100 PCs were used in the end, the results did not improve with using more than 20 PCs (see Figure C.1).

C Optimization Results

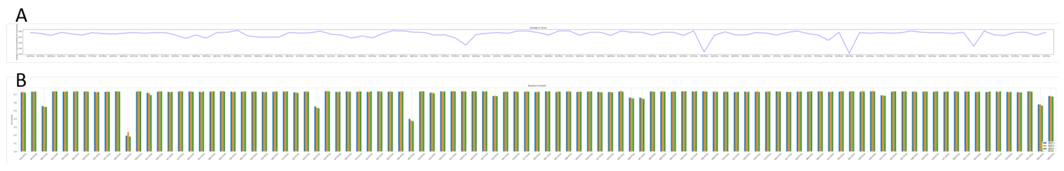


Figure C.1: Results of the optimization of the number of PCA components to use. Unfortunately, the optimization was run with deprecated versions of k-means clustering (A) and random forest classification (B) scripts, which results in a non-ordered x-axis and makes the figure hard to read. However, the plateau in the classification score is first reached with around 15 PCs. The average F1 score from the clustering reaches the plateau earlier, at around 10 PCs.

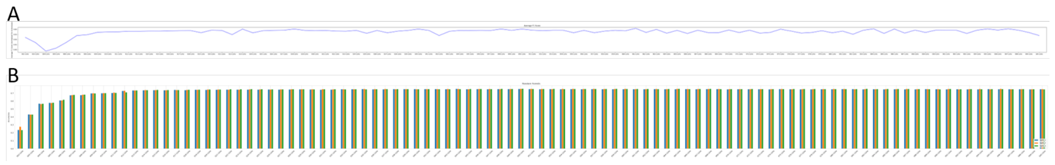


Figure C.2: Results of the optimization of the number of LSA components to use. This includes results of k-means clustering (A) and random forest (B). The score plateau is reached at about 13 components for classification and 9 for clustering. Unlike in PCA, the F1 clustering score does not diminish for LSA even with a large number of components.

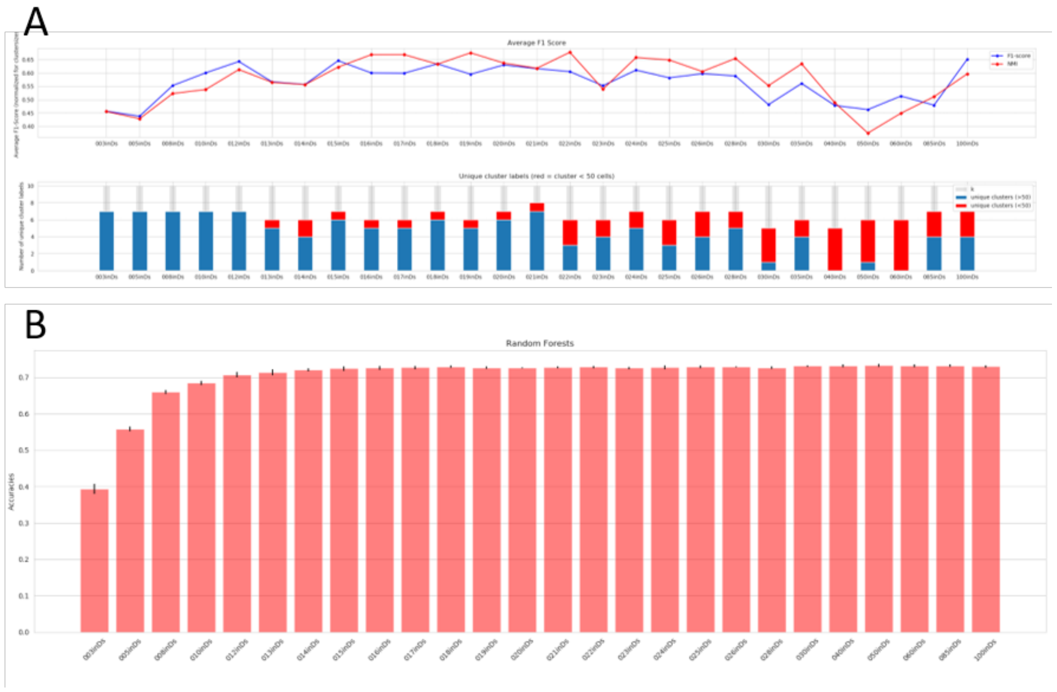


Figure C.3: Results of the optimization of the number of PCA components to use for ICA. This includes results of k-means clustering (A) and random forest (B). The random forest score reaches its plateau with around 15 input dimensions (inDs). The F1 and NMI scores of the clustering are however relatively constant, and no clear trend can be observed. Unfortunately, these clustering runs do not contain any repetitions yet, as this was only implemented at a later timepoint.

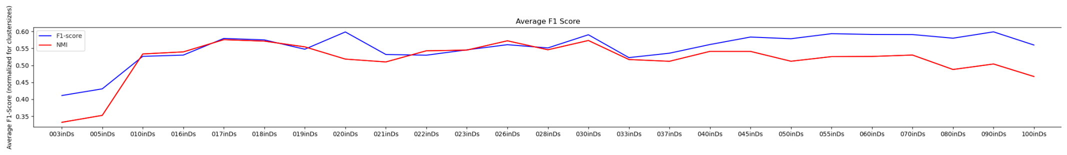


Figure C.4: Results of the optimization of the number of PCA components to use for t-SNE. Only results of k-means clustering are included, as no classification task was possible with t-SNE. The lack of repetitions makes clear statements difficult. Additionaly, the fluctuation makes the onset of the "score plateau" difficult to see, which is why the relatively odd number of 17 was chosen.

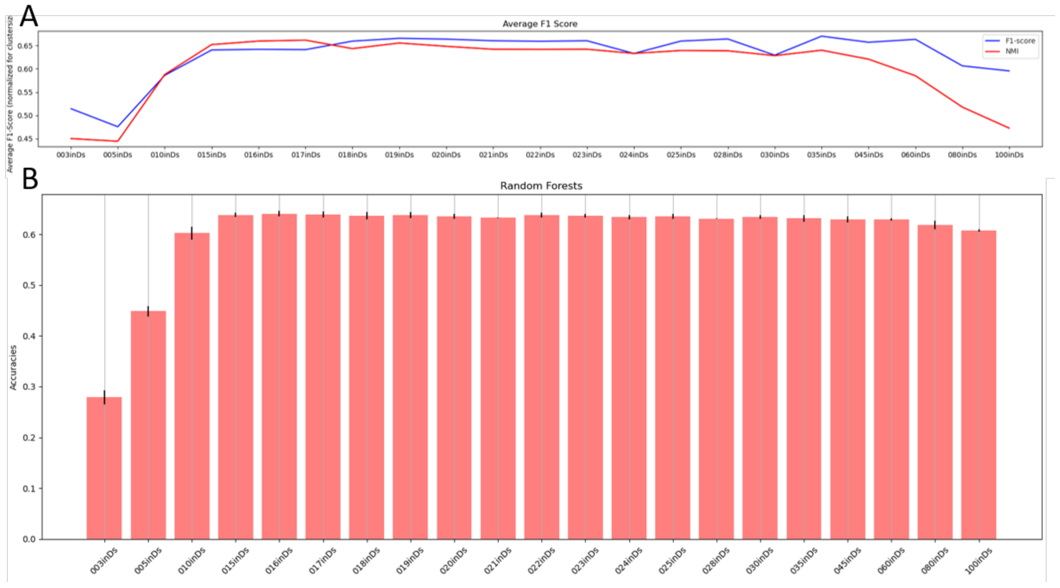


Figure C.5: Results of the optimization of the number of PCA components to use for UMAP. This includes results of k-means clustering (A) and random forest (B). High scores are reached after about 15 input dimensions (inDs) for both, clustering and classification scores.

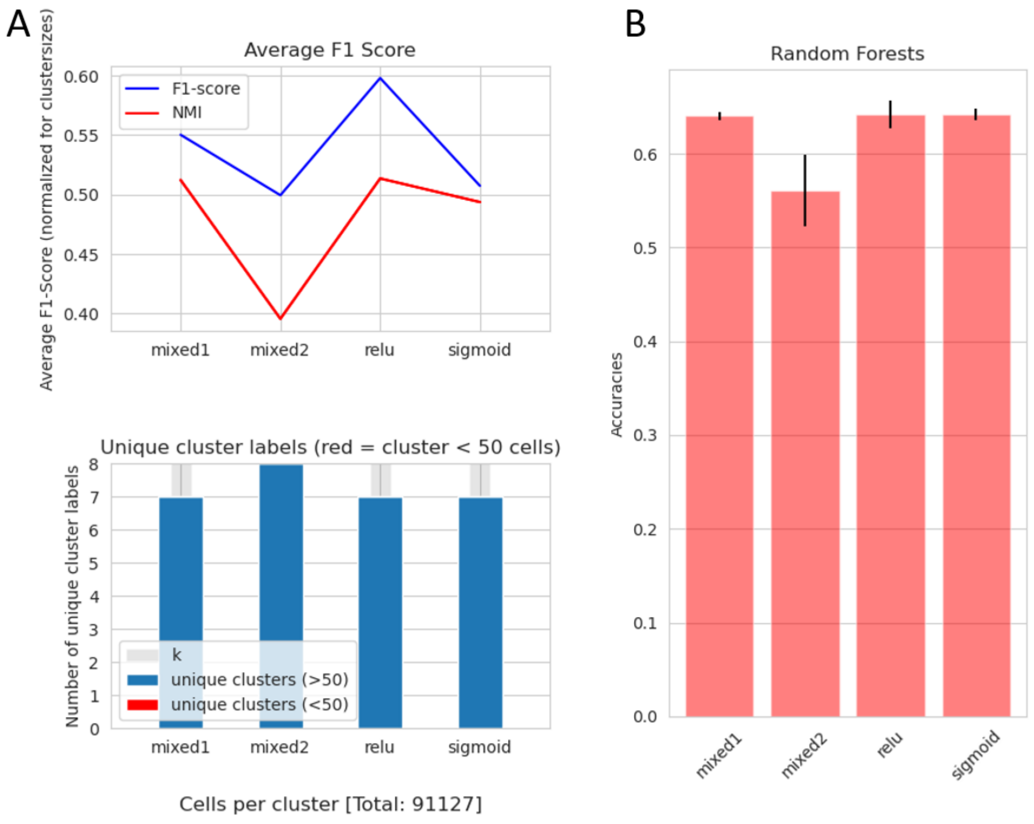


Figure C.6: Result of the BCA experimentation with the activation function as evaluated with k-means clustering (A) and random forest (B). Mixed1 means, the two "outer" layers of the autoencoder (from input data to 64 nodes and from the other 64 nodes to reconstructed data) were activated with "sigmoid", and the inner layers (from 64-32 and from 32-64) were activated with "relu". Mixed2 means the outer layers of the autoencoders were activated with "relu", and the inner layers of the autoencoder were activated with "sigmoid".

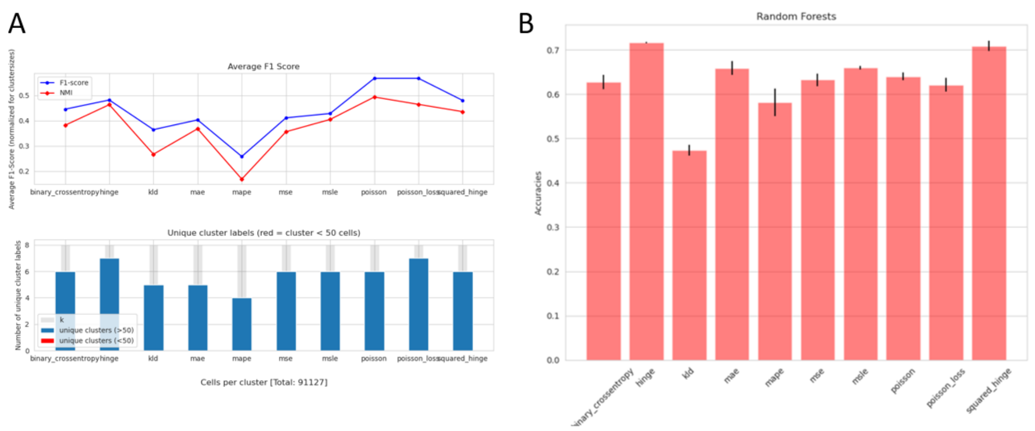


Figure C.7: Result of the BCA experimentation with the loss functions, as evaluated by K-means clustering (A) and random forest (B). Please note, that poisson_loss and poisson are inherently the same, it is just two different implementations, from Eraslan et al. ([2]) and the default keras implementation respectively.

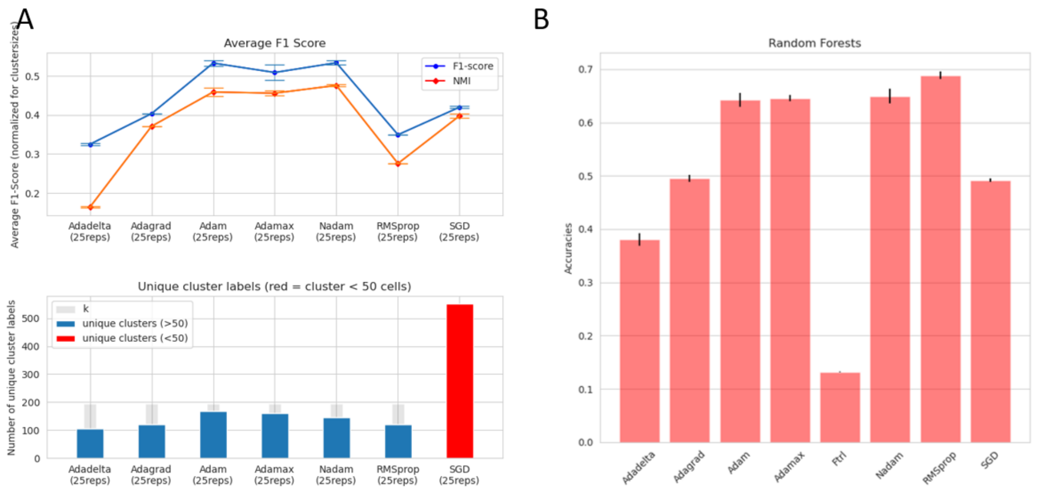
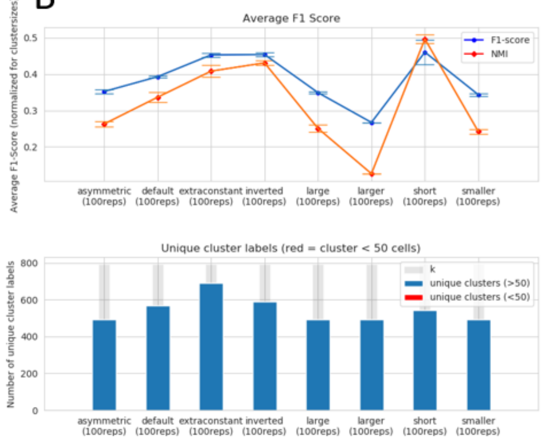


Figure C.8: Result of the BCA experimentation with the optimizer, as evaluated by K-means clustering (A) and random forest (B).

A

Layer structure	Encoder	Latent Layer	Decoder
asymmetric	Input - 88 - 51	32	45 - Output
default	Input - 64	32	64 - Output
extraconstant	Input - 64 - 64 - 32	32	32 - 64 - 64 - Output
inverted	Input - 32	64	32 - Output
large	Input - 128 - 64	32	64 - 128 - Output
larger	Input - 256	32	256 - Output
short	Input	32	Output
smaller	Input - 40	32	40 - Output

B



C

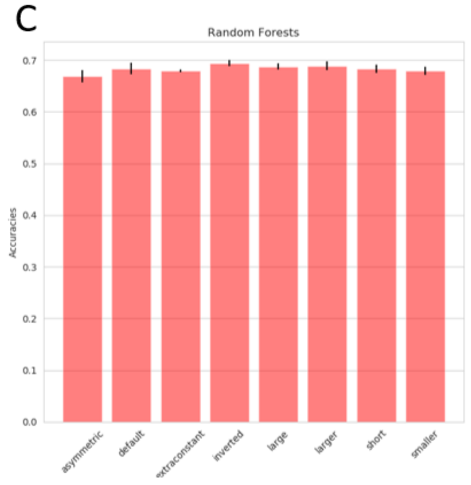
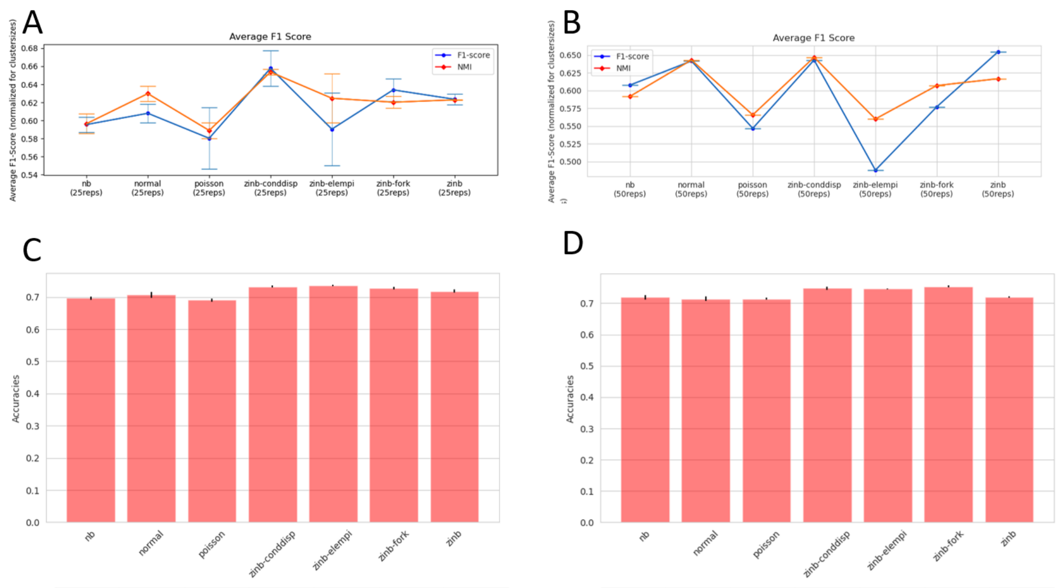


Figure C.9: Result of the BCA experimentation with the layer structure, as evaluated by K-means clustering (B) and random forest (C). The layer structure names are explained in the table (A). Unfortunately, even the best clustering scores are relatively bad, compared to the usual scores reached. Classification scores are in the usual range however. Interestingly, more complex structures usually lead to lower scores. In fact, using only the latent layer without any intermediate layer reached one of the best clustering scores, and classification scores were mostly constant. The layer where all the layers were duplicated also stands out, because it was able to find the most unique clusters and has very good clustering scores. This is interesting, as the duplication of a layer intuitively doesn't give the autoencoder any advantages, despite the additional freedom. The inverted ones with small layers around a larger "latent" layer reached the best score, however, this should not be overestimated, as it was the only case where the classification ran on 64 dimensions instead of 32, however, considering that this layer essentially cannot provide information that wasn't contained in the previous 32-nodes bottleneck already means that it would be interesting to run this previous bottleneck through the evaluation as well. In the end, the default layout of BCA was kept, less because of the scores shown here, but more as an effort to keep it more comparable to SCA and DCA.



	nb	normal	poisson	zinb-conddisp	zinb-elempi	zinb-fork	zinb
Average	0.696939	0.706859	0.690311	0.73264	0.735039	0.727826	0.718477
Stdev(P)	0.005082	0.009122	0.005814	0.003047	0.001875	0.00458	0.004957

	nb	normal	poisson	zinb-conddisp	zinb-elempi	zinb-fork	zinb
Average	0.718623	0.71359	0.713926	0.747637	0.745867	0.753665	0.719808
Stdev(P)	0.007018	0.007186	0.002963	0.004622	0.001476	0.002342	0.001975

Figure C.10: Comparison of different subtypes of DCA, as run in SCA with a different autoencoder type. Some of the implementations did not work for an unknown reason possibly connected to the way the loss is defined, which leads to invalid numbers. This seems to be purely chance-based, as did sometimes produce valid output when repeated often enough, however the missing ones (nb-conddisp, nb-shared, nb-fork, zinb-shared) never produced valid outputs.

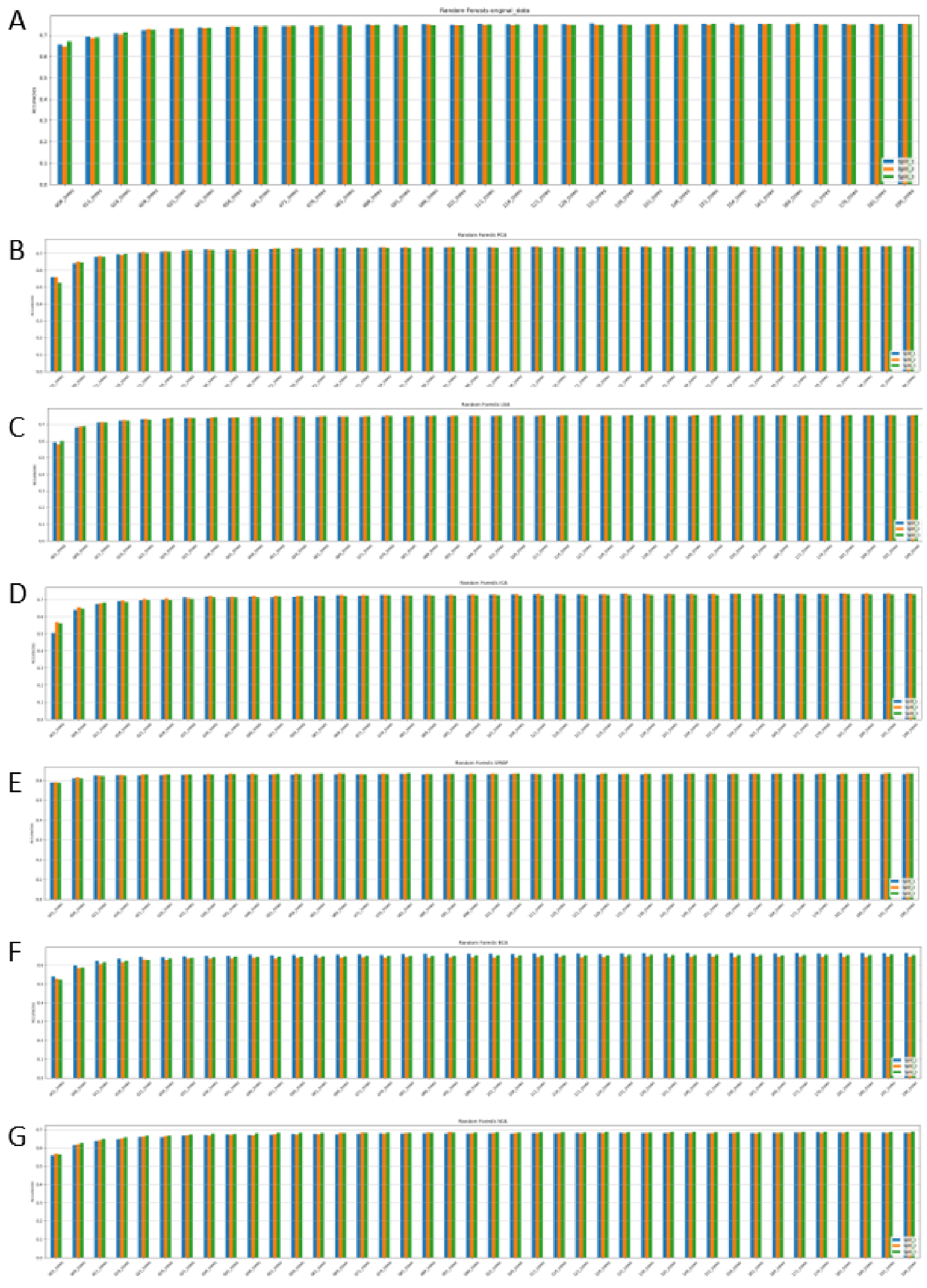


Figure C.11: Experiments regarding the optimal number of decision trees in the random forest. From top to bottom: Original Data (A), PCA (B), LSA (C), ICA (D), UMAP (E), BCA (F), SCA (G). The triplicates are shown as individual bars. Generally speaking, most techniques reach their plateau after around 15-20 trees. However, even using a single or very few trees already produces pretty good results. In the end, the default value of a 100 trees was kept for all evaluations.

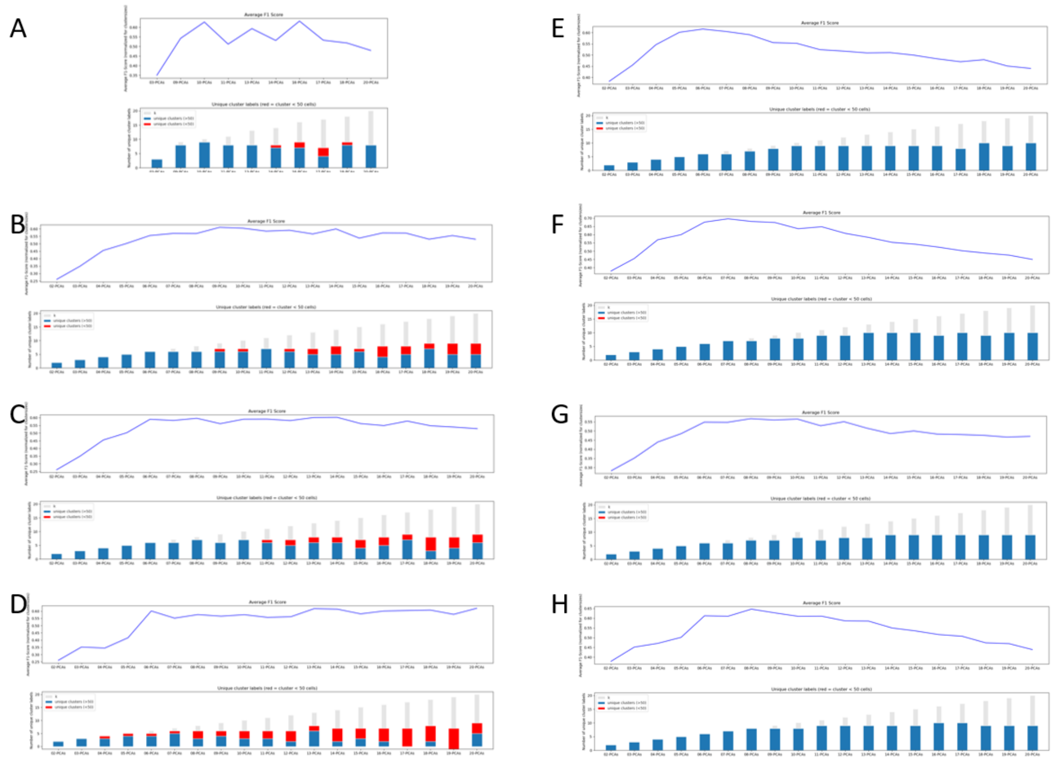


Figure C.12: Experiments regarding the optimal number of k for the k-means clustering algorithm. From top to bottom: Original Data (A), PCA (B), LSA (C), ICA (D), t-SNE (E), UMAP (F), BCA (G), SCA (H).

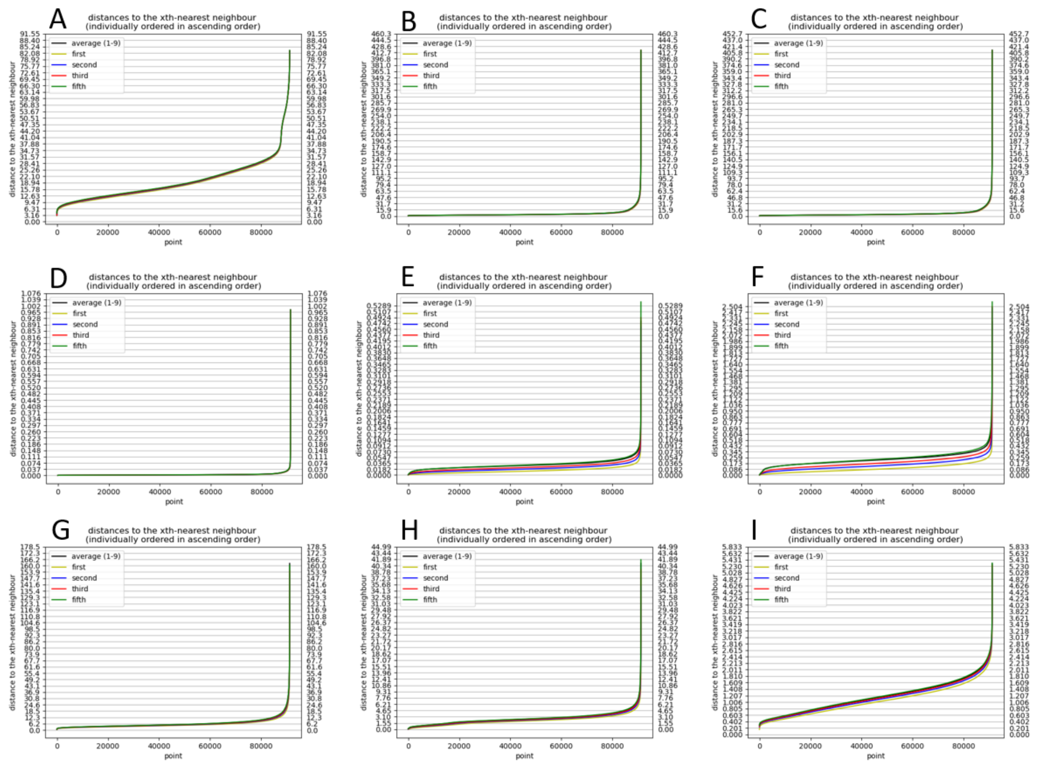


Figure C.13: Experiments regarding the optimal number of k for the k-means clustering algorithm. From left to right: Original Data (A), PCA (B), LSA (C), ICA (D), t-SNE (E), UMAP (F), BCA (G), DCA (H), SCA (I). The eps values chosen are Original Data: 35, PCA: 30, ICA: 0.03, LSA: 28, tSNE: 0.45, UMAP: 0.07, DCA: 4.6, BCA: 17, SCA: 2.1.

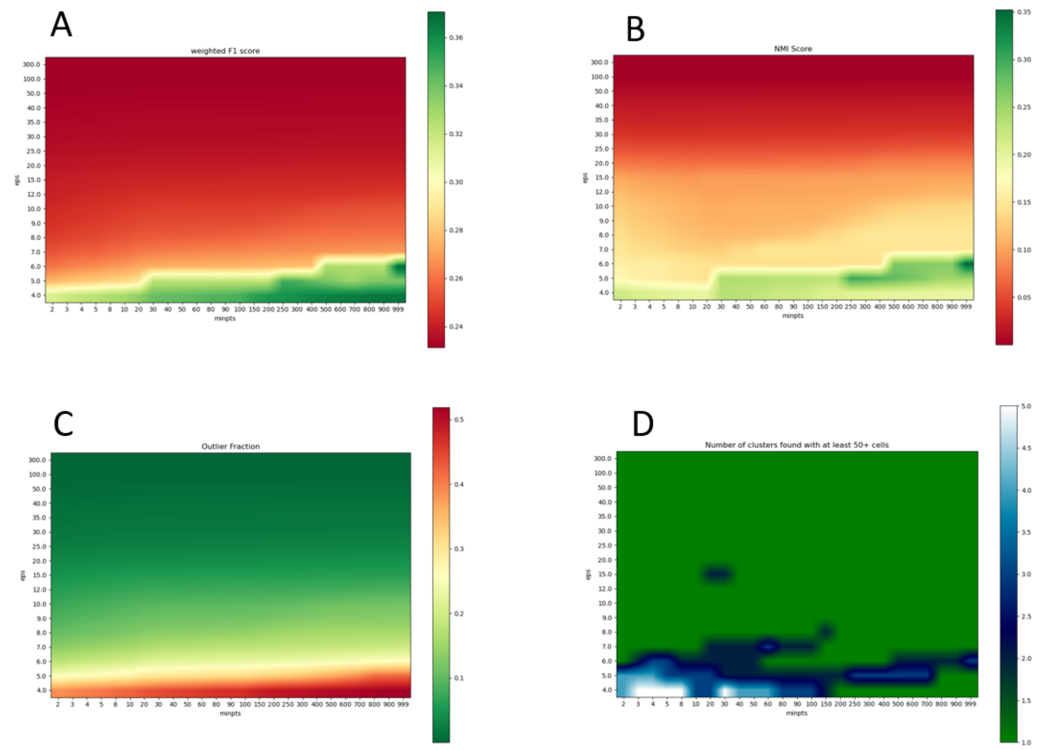


Figure C.14: Grid-search for a wide range of possible eps and minpts values. The axes are not scaled. Every point with a label on both axes has been calculated, the other points are coloured with the goal of producing a smooth colour gradient. A: Showing the weighted F1-score. B: showing the Normalized Mutual Information (NMI) score. C: Showing the outlier fraction. D: Showing the number of clusters with at least 50 cells.