Automatic **cell identification** in **single-cell RNA-sequencing** data

Lieke Michielsen

Automatic cell identification in single-cell RNA-sequencing data

by

L.C.M. Michielsen

to obtain the degree of Master of Science at Delft University of Technology,
to be defended in public on Thursday January 30th, 2020 at 10:30.

Student number	*4282361*
Master programme	*Computer science, Data Science and Technology track, Bioinformatics specialisation*
Faculty	*Electrical Engineering, Mathematics and Computer Science*
Thesis committee	*Prof. Dr. Ir. Marcel Reinders*
	Dr. Ahmed Mahfouz
	Dr. Willem-Paul Brinkman
	Dr. David Tax
	Soufiane Mourragui

TU Delft

Preface

In front of you lies the master thesis 'Automatic cell identification in single-cell RNA-sequencing data'. I am really happy that I could combine the most interesting subjects of my studies in this master thesis. During my Nanobiology bachelor, I was introduced to many aspects of cell biology. I soon noticed, however, that working in a wet lab was not my cup of tea, so made a switch to the Computer Science master. During this master, pattern recognition and machine learning were to me one of the most interesting subjects. This project gave me the opportunity to combine these two interests.

Before you start reading I would like to thank several people for their help during this project. First of all, I would like to thank Ahmed and Marcel for providing me with this interesting subject and their helpful, constructive feedback. I really appreciate their support and the amount of time they invested in this project. Next, I would like to thank Tamim for his idea to do the benchmarking paper and especially to have faith that it would be possible to submit the first version in less than six weeks. I really enjoyed working on the benchmark together. Finally, I would like to thank my family and friends for their general support throughout my studies and unconditional belief in me.

Lieke Michielsen
Delft, January 2020

Note to committee

This master thesis includes two different projects, which both took 4.5 months to finish. During the first project I worked together with Tamim Abdelaal on a benchmarking paper. Here, we compared several classifiers for scRNA-sequencing data. We both did half of the work. Tamim focused on the preparation of the datasets and ran all the R tools, while I ran the python tools and focused on evaluating the results. During the second project, I worked on my own project where I developed a new method that combines multiple datasets to improve the classification performance.

A

A comparison of automatic cell identification methods for single-cell RNA sequencing data

Abdelaal et al. Genome Biology (2019) 20:194
https://doi.org/10.1186/s13059-019-1795-z

Genome Biology

RESEARCH Open Access

A comparison of automatic cell identification methods for single-cell RNA sequencing data

Tamim Abdelaal[1,2†], Lieke Michielsen[1,2†], Davy Cats[3], Dylan Hoogduin[3], Hailiang Mei[3], Marcel J. T. Reinders[1,2] and Ahmed Mahfouz[1,2*]

Abstract

Background: Single-cell transcriptomics is rapidly advancing our understanding of the cellular composition of complex tissues and organisms. A major limitation in most analysis pipelines is the reliance on manual annotations to determine cell identities, which are time-consuming and irreproducible. The exponential growth in the number of cells and samples has prompted the adaptation and development of supervised classification methods for automatic cell identification.

Results: Here, we benchmarked 22 classification methods that automatically assign cell identities including single-cell-specific and general-purpose classifiers. The performance of the methods is evaluated using 27 publicly available single-cell RNA sequencing datasets of different sizes, technologies, species, and levels of complexity. We use 2 experimental setups to evaluate the performance of each method for within dataset predictions (intra-dataset) and across datasets (inter-dataset) based on accuracy, percentage of unclassified cells, and computation time. We further evaluate the methods' sensitivity to the input features, number of cells per population, and their performance across different annotation levels and datasets. We find that most classifiers perform well on a variety of datasets with decreased accuracy for complex datasets with overlapping classes or deep annotations. The general-purpose support vector machine classifier has overall the best performance across the different experiments.

Conclusions: We present a comprehensive evaluation of automatic cell identification methods for single-cell RNA sequencing data. All the code used for the evaluation is available on GitHub (https://github.com/tabdelaal/scRNAseq_Benchmark). Additionally, we provide a Snakemake workflow to facilitate the benchmarking and to support the extension of new methods and new datasets.

Keywords: scRNA-seq, Benchmark, Classification, Cell identity

Background

Single-cell RNA sequencing (scRNA-seq) provides unprecedented opportunities to identify and characterize the cellular composition of complex tissues. Rapid and continuous technological advances over the past decade have allowed scRNA-seq technologies to scale to thousands of cells per experiment [1]. A common analysis step in analyzing single-cell data involves the identification of cell populations presented in a given dataset. This task is typically solved by unsupervised clustering of cells into groups based on the similarity of their gene expression profiles, followed by cell population annotation by assigning labels to each cluster. This approach proved very valuable in identifying novel cell populations and resulted in cellular maps of entire cell lineages, organs, and even whole organisms [2–7]. However, the annotation step is cumbersome and time-consuming as it involves manual inspection of cluster-specific marker genes. Additionally, manual annotations, which are often not based on standardized ontologies of cell labels, are not reproducible across different experiments within and across research groups. These caveats become even more pronounced as the number of cells and samples increases, preventing fast and reproducible annotations.

* Correspondence: a.mahfouz@lumc.nl
†Tamim Abdelaal and Lieke Michielsen contributed equally to this work.
[1]Leiden Computational Biology Center, Leiden University Medical Center, Einthovenweg 20, 2333 ZC Leiden, The Netherlands
[2]Delft Bioinformatics Laboratory, Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands
Full list of author information is available at the end of the article

To overcome these challenges, a growing number of classification approaches are being adapted to automatically label cells in scRNA-seq experiments. scRNA-seq classification methods predict the identity of each cell by learning these identities from annotated training data (e.g., a reference atlas). scRNA-seq classification methods are relatively new compared to the plethora of methods addressing different computational aspects of single-cell analysis (such as normalization, clustering, and trajectory inference). However, the number of classification methods is rapidly growing to address the aforementioned challenges [8, 9]. While all scRNA-seq classification methods share a common goal, i.e., accurate annotation of cells, they differ in terms of their underlying algorithms and the incorporation of prior knowledge (e.g., cell type marker gene tables).

In contrast to the extensive evaluations of clustering, differential expression, and trajectory inference methods [10–12], there is currently one single attempt comparing methods to assign cell type labels to cell clusters [13]. The lack of a comprehensive comparison of scRNA-seq classification methods leaves users without indications as to which classification method best fits their problem. More importantly, a proper assessment of the existing approaches in comparison with the baseline methods can greatly benefit new developments in the field and prevent unnecessary complexity.

Here, we benchmarked 22 classification methods to automatically assign cell identities including single-cell-specific and general-purpose classifiers. The methods were evaluated using 27 publicly available single-cell RNA sequencing datasets of different sizes, technologies, species, and complexity. The performance of the methods was evaluated based on their accuracy, percentage of unclassified cells, and computation time. We performed several experiments to cover different levels of challenge in the classification task and to test specific features or tasks such as the feature selection, scalability, and rejection experiments. We evaluated the classification performance through two experimental setups: (1) intra-dataset in which we applied 5-fold cross-validation within each dataset and (2) inter-dataset involving across datasets comparisons. The inter-dataset comparison is more realistic and more practical, where a reference dataset (e.g., atlas) is used to train a classifier which can then be applied to identify cells in new unannotated datasets. However, in order to perform well across datasets, the classifier should also perform well using the intra-dataset setup on the reference dataset. The intra-dataset experiments, albeit artificial, provide an ideal scenario to evaluate different aspects of the classification process (e.g., feature selection, scalability, and different annotation levels), regardless of the technical and biological variations across datasets. In general, most classifiers perform well across all datasets in both experimental setups (inter- and intra-dataset), including the general-purpose classifiers. In our experiments, incorporating prior knowledge in the form of marker genes does not improve the performance. We observed large variation across different methods in the computation time and classification performance in response to changing the input features and the number of cells. Our results highlight the general-purpose support vector machine (*SVM*) classifier as the best performer overall.

Results

Benchmarking automatic cell identification methods (intra-dataset evaluation)

We benchmarked the performance and computation time of all 22 classifiers (Table 1) across 11 datasets used for intra-dataset evaluation (Table 2). Classifiers were divided into two categories: (1) supervised methods which require a training dataset labeled with the corresponding cell populations in order to train the classifier or (2) prior-knowledge methods, for which either a marker gene file is required as an input or a pretrained classifier for specific cell populations is provided.

The datasets used in this study vary in the number of cells, genes, and cell populations (annotation level), in order to represent different levels of challenges in the classification task and to evaluate how each classifier performs in each case (Table 2). They include relatively typical sized scRNA-seq datasets (1500–8500 cells), such as the 5 pancreatic datasets (Baron Mouse, Baron Human, Muraro, Segerstolpe, and Xin), which include both mouse and human pancreatic cells and vary in the sequencing protocol used. The Allen Mouse Brain (AMB) dataset is used to evaluate how the classification performance changes when dealing with different levels of cell population annotation as the AMB dataset contains three levels of annotations for each cell (3, 16, or 92 cell populations), denoted as AMB3, AMB16, and AMB92, respectively. The Tabula Muris (TM) and Zheng 68K datasets represent relatively large scRNA-seq datasets (> 50,000 cells) and are used to assess how well the classifiers scale with large datasets. For all previous datasets, cell populations were obtained through clustering. To assess how the classifiers perform when dealing with sorted populations, we included the CellBench dataset and the Zheng sorted dataset, representing sorted populations for lung cancer cell lines and peripheral blood mononuclear cells (PBMC), respectively. Including the Zheng sorted and Zheng 68K datasets allows the benchmarking of 4 prior-knowledge classifiers, since the marker gene files or pretrained classifiers are available for the 4 classifiers for PBMCs.

All classifiers perform well in intra-dataset experiments

Generally, all classifiers perform well in the intra-dataset experiments, including the general-purpose classifiers (Fig. 1). However, *Cell-BLAST* performs poorly for the

Table 1 Automatic cell identification methods included in this study

Name	Version	Language	Underlying classifier	Prior knowledge	Rejection option	Reference
Garnett	0.1.4	R	Generalized linear model	Yes	Yes	[14]
Moana	0.1.1	Python	SVM with linear kernel	Yes	No	[15]
DigitalCellSorter	GitHub version: e369a34	Python	Voting based on cell type markers	Yes	No	[16]
SCINA	1.1.0	R	Bimodal distribution fitting for marker genes	Yes	No	[17]
scVI	0.3.0	Python	Neural network	No	No	[18]
Cell-BLAST	0.1.2	Python	Cell-to-cell similarity	No	Yes	[19]
ACTINN	GitHub version: 563bcc1	Python	Neural network	No	No	[20]
LAmbDA	GitHub version: 3891d72	Python	Random forest	No	No	[21]
scmapcluster	1.5.1	R	Nearest median classifier	No	Yes	[22]
scmapcell	1.5.1	R	kNN	No	Yes	[22]
scPred	0.0.0.9000	R	SVM with radial kernel	No	Yes	[23]
CHETAH	0.99.5	R	Correlation to training set	No	Yes	[24]
CaSTLe	GitHub version: 258b278	R	Random forest	No	No	[25]
SingleR	0.2.2	R	Correlation to training set	No	No	[26]
scID	0.0.0.9000	R	LDA	No	Yes	[27]
singleCellNet	0.1.0	R	Random forest	No	No	[28]
LDA	0.19.2	Python	LDA	No	No	[29]
NMC	0.19.2	Python	NMC	No	No	[29]
RF	0.19.2	Python	RF (50 trees)	No	No	[29]
SVM	0.19.2	Python	SVM (linear kernel)	No	No	[29]
$SVM_{rejection}$	0.19.2	Python	SVM (linear kernel)	No	Yes	[29]
kNN	0.19.2	Python	kNN ($k = 9$)	No	No	[29]

Baron Mouse and Segerstople pancreatic datasets. Further, *scVI* has low performance on the deeply annotated datasets TM (55 cell populations) and AMB92 (92 cell populations), and *kNN* produces low performance for the Xin and AMB92 datasets.

For the pancreatic datasets, the best-performing classifiers are *SVM*, $SVM_{rejection}$, *scPred*, *scmapcell*, *scmapcluster*, *scVI*, *ACTINN*, *singleCellNet*, *LDA*, and *NMC*. *SVM* is the only classifier to be in the top five list for all five pancreatic datasets, while *NMC*, for example, appears only in the top five list for the Xin dataset. The Xin dataset contains only four pancreatic cell types (alpha, beta, delta, and gamma) making the classification task relatively easy for all classifiers, including *NMC*. Considering the median F1-score alone to judge the classification performance can be misleading since some classifiers incorporate a rejection option (e.g., $SVM_{rejection}$, *scmapcell*, *scPred*), by which a cell is assigned as "unlabeled" if the classifier is not confident enough. For example, for the Baron Human dataset, the median F1-score for $SVM_{rejection}$, *scmapcell*, *scPred*, and *SVM* is 0.991, 0.984, 0.981, and 0.980, respectively (Fig. 1a). However, $SVM_{rejection}$, *scmapcell*, and *scPred* assigned 1.5%, 4.2%, and 10.8% of the cells, respectively, as unlabeled while *SVM* (without rejection) classified 100% of the cells with a median F1-score of 0.98 (Fig. 1b). This shows an overall better performance for *SVM* and $SVM_{rejection}$, with higher performance and less unlabeled cells.

The CellBench 10X and CEL-Seq2 datasets represent an easy classification task, where the five sorted lung cancer cell lines are quite separable [34]. All classifiers have an almost perfect performance on both CellBench datasets (median F1-score ≈ 1).

For the TM dataset, the top five performing classifiers are $SVM_{rejection}$, *SVM*, *scmapcell*, *Cell-BLAST*, and *scPred* with a median F1-score > 0.96, showing that these classifiers can perform well and scale to large scRNA-seq datasets with a deep level of annotation. Furthermore, *scmapcell* and *scPred* assigned 9.5% and 17.7% of the cells, respectively, as unlabeled, which shows a superior performance for $SVM_{rejection}$ and *SVM*, with a higher median F1-score and 2.9% and 0% unlabeled cells, respectively.

Performance evaluation across different annotation levels

We used the AMB dataset with its three different levels of annotations, to evaluate the classifiers' performance behavior with an increasing number of smaller cell populations within the same dataset. For AMB3, the classification task is relatively easy, differentiating between three major brain cell types (inhibitory neurons, esxcitatory neurons, and non-neuronal). All classifiers perform

Table 2 Overview of the datasets used during this study

Dataset	No. of cells	No. of genes	No. of cell populations (> 10 cells)	Description	Protocol	Reference
Baron (Mouse)[a]	1886	14,861	13 (9)	Mouse pancreas	inDrop	[30]
Baron (Human)[a,b]	8569	17,499	14 (13)	Human pancreas	inDrop	[30]
Muraro[a,b]	2122	18,915	9 (8)	Human pancreas	CEL-Seq2	[31]
Segerstolpe[a,b]	2133	22,757	13 (9)	Human pancreas	SMART-Seq2	[32]
Xin[a,b]	1449	33,889	4 (4)	Human pancreas	SMARTer	[33]
CellBench 10X[a,b]	3803	11,778	5 (5)	Mixture of five human lung cancer cell lines	10X chromium	[34]
CellBench CEL-Seq2[a,b]	570	12,627	5 (5)	Mixture of five human lung cancer cell lines	CEL-Seq2	[34]
TM[a]	54,865	19,791	55 (55)	Whole *Mus musculus*	SMART-Seq2	[6]
AMB[a]	12,832	42,625	4/22/110 (3/16/92)	Primary mouse visual cortex	SMART-Seq v4	[35]
Zheng sorted[a]	20,000	21,952	10 (10)	FACS-sorted PBMC	10X CHROMIUM	[36]
Zheng 68K[a]	65,943	20,387	11 (11)	PBMC	10X CHROMIUM	[36]
VISp[b] (Mouse)	12,832	42,625	3/36 (3/34)	Primary visual cortex	SMART-Seq v4	[35]
ALM[b] (Mouse)	8758	42,461	3/37 (3/34)	Anterior lateral motor area	SMART-Seq v4	[35]
MTG[b] (Human)	14,636	16,161	3/35 (3/34)	Middle temporal gyrus	SMART-Seq v4	[37]
PbmcBench pbmc1.10Xv2[b]	6444	33,694	9 (9)	PBMC	10X version 2	[38]
PbmcBench pbmc1.10Xv3[b]	3222	33,694	8 (8)	PBMC	10X version 3	[38]
PbmcBench pbmc1.CL[b]	253	33,694	7 (7)	PBMC	CEL-Seq2	[38]
PbmcBench pbmc1.DR[b]	3222	33,694	9 (9)	PBMC	Drop-Seq	[38]
PbmcBench pbmc1.iD[b]	3222	33,694	7 (7)	PBMC	inDrop	[38]
PbmcBench pbmc1.SM2[b]	253	33,694	6 (6)	PBMC	SMART-Seq2	[38]
PbmcBench pbmc1.SW[b]	3176	33,694	7 (7)	PBMC	Seq-Well	[38]
PbmcBench pbmc2.10Xv2[b]	3362	33,694	9 (9)	PBMC	10X version 2	[38]
PbmcBench pbmc2.CL[b]	273	33,694	5 (5)	PBMC	CEL-Seq2	[38]
PbmcBench pbmc2.DR[b]	3362	33,694	6 (6)	PBMC	Drop-Seq	[38]
PbmcBench pbmc2.iD[b]	3362	33,694	9 (9)	PBMC	inDrop	[38]
PbmcBench pbmc2.SM2[b]	273	33,694	6 (6)	PBMC	SMART-Seq2	[38]
PbmcBench pbmc2.SW[b]	551	33,694	4 (4)	PBMC	Seq-Well	[38]

[a]Used for intra-dataset evaluation
[b]Used for inter-dataset evaluation

almost perfectly with a median F1-score > 0.99 (Fig. 1a). For AMB16, the classification task becomes slightly more challenging and the performance of some classifiers drops, especially *kNN*. The top five classifiers are $SVM_{rejection}$, *scmapcell*, *scPred*, *SVM*, and *ACTINN*, where $SVM_{rejection}$, *scmapcell*, and *scPred* assigned 1.1%, 4.9%, and 8.4% of the cells as unlabeled, respectively. For the deeply annotated AMB92 dataset, the performance of all classifiers drops further, specially for *kNN* and *scVI*, where the median F1-score is 0.130 and zero, respectively. The top five classifiers are $SVM_{rejection}$, *scmapcell*, *SVM*, *LDA*, and *scmapcluster*, with $SVM_{rejection}$ assigning less cells as unlabeled compared to *scmapcell* (19.8% vs 41.9%), and once more, $SVM_{rejection}$ shows improved performance over *scmapcell* (median F1-score of 0.981 vs 0.906). These results show an overall superior performance for general-purpose classifiers ($SVM_{rejection}$, *SVM*, and *LDA*) compared to other scRNA-seq-specific classifiers across different levels of cell population annotation.

Instead of only looking at the median F1-score, we also evaluated the F1-score per cell population for each classifier (Additional file 1: Figure S1). We confirmed previous conclusions that *kNN* performance drops with deep annotations which include smaller cell populations (Additional file 1: Figure S1B-C), and *scVI* poorly performs on the deeply annotated AMB92 dataset. Additionally, we observed that some cell populations are much harder to classify compared to other populations. For example, most classifiers had a low performance on the *Serpinf1* cells in the AMB16 dataset.

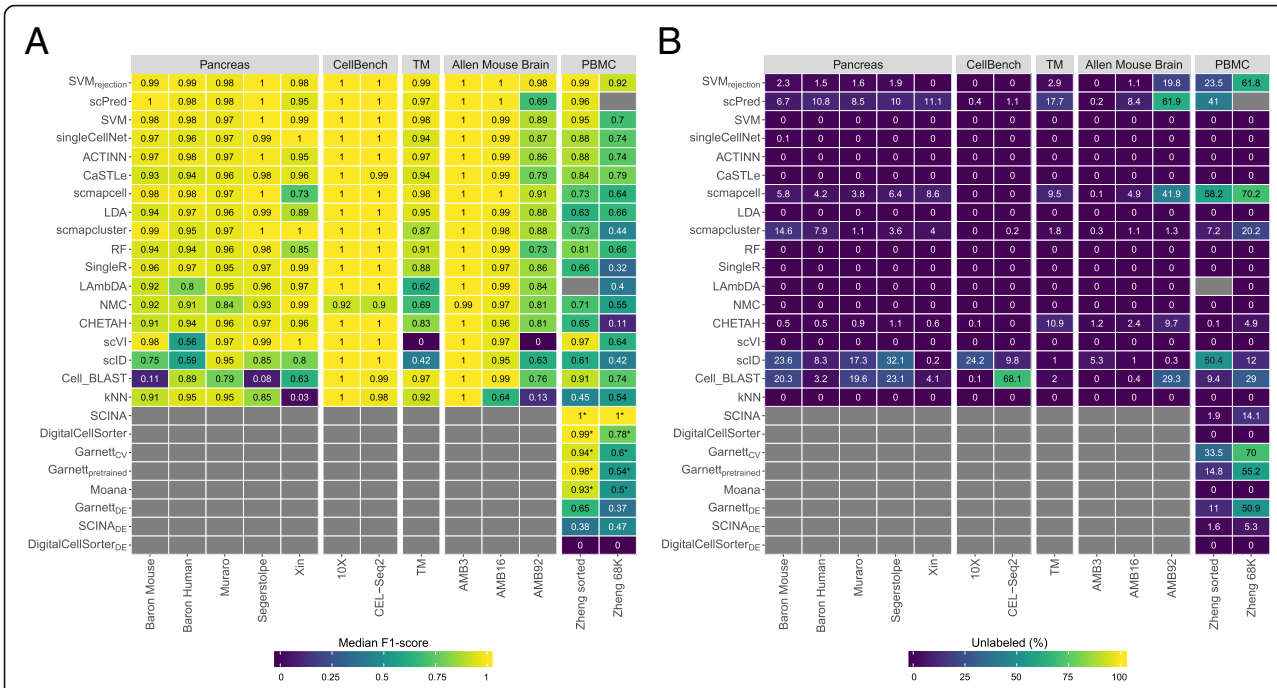


Fig. 1 Performance comparison of supervised classifiers for cell identification using different scRNA-seq datasets. Heatmap of the **a** median F1-scores and **b** percentage of unlabeled cells across all cell populations per classifier (rows) per dataset (columns). Gray boxes indicate that the corresponding method could not be tested on the corresponding dataset. Classifiers are ordered based on the mean of the median F1-scores. Asterisk (*) indicates that the prior-knowledge classifiers, *SCINA*, *DigitalCellSorter*, $Garnett_{CV}$, $Garnett_{pretrained}$, and *Moana*, could not be tested on all cell populations of the PBMC datasets. $SCINA_{DE}$, $Garnett_{DE}$, and $DigitalCellSorter_{DE}$ are versions of *SCINA*, $Garnett_{CV}$, and *DigitalCellSorter*, the marker genes are defined using differential expression from the training data. Different numbers of marker genes, 5, 10, 15, and 20, were tested, and the best result is shown here. *SCINA*, *Garnett*, and *DigitalCellSorter* produced the best result for the Zheng sorted dataset using 20, 15, and 5 markers, and for the Zheng 68K dataset using 10, 5, and 5 markers, respectively

Incorporating prior-knowledge does not improve intra-dataset performance on PBMC data

For the two PBMC datasets (Zheng 68K and Zheng sorted), the prior-knowledge classifiers *Garnett*, *Moana*, *DigitalCellSorter*, and *SCINA* could be evaluated and benchmarked with the rest of the classifiers. Although the best-performing classifier on Zheng 68K is *SCINA* with a median F1-score of 0.998, this performance is based only on 3, out of 11, cell populations (Monocytes, B cells, and NK cells) for which marker genes are provided. Additional file 1: Table S1 summarizes which PBMC cell populations can be classified by the prior-knowledge methods. Interestingly, none of the prior-knowledge methods showed superior performance compared to other classifiers, despite the advantage these classifiers have over other classifiers given they are tested on fewer cell populations due to the limited availability of marker genes. *Garnett*, *Moana*, and *DigitalCellSorter* could be tested on 7, 7, and 5 cell populations, respectively (Additional file 1: Table S1). Besides *SCINA*, the top classifiers for the Zheng 68K dataset are *CaSTLe*, *ACTINN*, *singleCellNet*, and *SVM*. $SVM_{rejection}$ and *Cell-BLAST* show high performance, at the expense of a high rejection rate of 61.8% and 29%, respectively (Fig. 1). Moreover, *scPred* failed when tested on the Zheng 68K dataset. Generally, all classifiers show relatively lower performance on the Zheng 68K dataset compared to other datasets, as the Zheng 68K dataset contains 11 immune cell populations which are harder to differentiate, particularly the T cell compartment (6 out of 11 cell populations). This difficulty of separating these populations was previously noted in the original study [36]. Also, the confusion matrices for *CaSTLe*, *ACTINN*, *singleCellNet*, and *SVM* clearly indicate the high similarity between cell populations, such as (1) monocytes with dendritic cells, (2) the 2 CD8+ T populations, and (3) the 4 CD4+ T populations (Additional file 1: Figure S2).

The classification of the Zheng sorted dataset is relatively easier compared to the Zheng 68K dataset, as almost all classifiers show improved performance (Fig. 1), with the exception that *LAmbDA* failed while being tested on the Zheng sorted dataset. The prior-knowledge methods show high performance (median F1-score > 0.93), which is still comparable to other classifiers such as $SVM_{rejection}$, *scVI*, *scPred*, and *SVM*. Yet, the supervised classifiers do not require any marker genes, and they can predict more (all) cell populations.

The performance of prior-knowledge classifiers strongly depends on the selected marker genes

Some prior-knowledge classifiers, *SCINA*, *DigitalCellSorter*, and $Garnett_{CV}$, used marker genes to classify the cells. For the PBMC datasets, the number of marker genes per cell population varies across classifiers (2–161 markers) and the marker genes show very little overlap. Only one B cell marker gene, CD79A, is shared by all classifiers while none of the marker genes for the other cell populations is shared by the three classifiers. We analyzed the effect of the number of marker genes, mean expression, dropout rate, and the specificity of each marker gene (beta score, see the "Methods" section) on the performance of the classifier (Additional file 1: Figure S3). The dropout rate and marker specificity (beta-score) are strongly correlated with the median F1-score, highlighting that the performance does not only depend on biological knowledge, but also on technical factors.

The difference between the marker genes used by each method underscores the challenge of marker gene selection, especially for smaller cell populations. Moreover, public databases of cell type markers (e.g., PanglaoDB [39] and CellMarker [40]) often provide different markers for the same population. For example, CellMarker provides 33 marker genes for B cells, while PanglaoDB provides 110 markers, with only 11 marker genes overlap between the two databases.

Given the differences between "expert-defined" markers and the correlation of classification performance and technical dataset-specific features (e.g., dropout rate), we tested if the performance of prior-knowledge methods can be improved by automatically selecting marker genes based on differential expression. Through the cross-validation scheme, we used the training folds to select the marker genes of each cell population based on differential expression (see the "Methods" section) and later used these markers to evaluate the classifiers' performance on the testing fold. We tested this approach on the two PBMC datasets, Zheng sorted and Zheng 68K for different numbers of marker genes (5, 10, 15, and 20 markers). In Fig. 1, the best result across the number of markers for $SCINA_{DE}$, $Garnett_{DE}$, and $DigitalCellSorter_{DE}$ are shown.

The median F1-score obtained using the differential expression-defined markers is significantly lower compared to the original versions of classifiers using the markers defined by the authors. This lower performance is in part due to the low performance on challenging populations, such as subpopulations of CD4+ and CD8+ T cell populations (F1-score ≤ 0.68) (Additional file 1: Figure S4). These challenging populations are not identified by the original classifiers since the markers provided by the authors only considered annotations at a higher level (Additional file 1: Table S1). For example, the median F1-score of $_{SCINADE}$ on Zheng sorted is 0.38, compared to a median F1-score of 1.0 for *SCINA* (using the original markers defined by the authors). However, *SCINA* only considers three cell populations: CD14+ monocytes, CD56+ NK cells, and CD19+ B cells. If we only consider these cell populations for $SCINA_{DE}$, this results in a median F1-score of 0.95.

We observed that the optimal number of marker genes varies per classifier and dataset. For the Zheng sorted dataset, the optimal number of markers is 5, 15, and 20 for $DigitalCellSorter_{DE}$, $Garnett_{DE}$, and $SCINA_{DE}$, respectively, while for Zheng 68K, this is 5, 5, and 10. All together, these results illustrate the dependence of the classification performance on the careful selection of marker genes which is evidently a challenging task.

Classification performance depends on dataset complexity

A major aspect affecting the classification performance is the complexity of the dataset at hand. We described the complexity of each dataset in terms of the pairwise similarity between cell populations (see the "Methods" section) and compared the complexity to the performance of the classifiers and the number of cell populations in a dataset (Fig. 2). When the complexity and/or the number of cell populations of the dataset increases, the performance generally decreases. The performance of all classifiers is relatively low on the Zheng 68K dataset, which can be explained by the high pairwise correlations between the mean expression profiles of each cell population (Additional file 1: Figure S5). These correlations are significantly lower for the TM and AMB92 datasets, justifying the higher performance of the classifiers on these two datasets (Additional file 1: Figures S6–S7). While both TM and AMB92 have more cell populations (55 and 92, respectively) compared to Zheng 68K (11 populations), these populations are less correlated to one another, making the task easier for all the classifiers.

Performance evaluation across datasets (inter-dataset evaluation)

While evaluating the classification performance within a dataset (intra-dataset) is important, the realistic scenario in which a classifier is useful requires cross-dataset (i.e., inter-dataset) classification. We used 22 datasets (Table 2) to test the classifiers' ability to predict cell identities in a dataset that was not used for training. First, we tested the classifiers' performance across different sequencing protocols, applied to the same samples within the same lab using the two CellBench datasets. We evaluated the classification performance when training on one protocol and testing on the other. Similar to the intra-dataset evaluation result, all classifiers performed well in this case (Additional file 1: Figure S8).

Second, we tested the classification performance on the PbmcBench datasets, which represent a more extensive protocol comparison. PbmcBench consists of 2

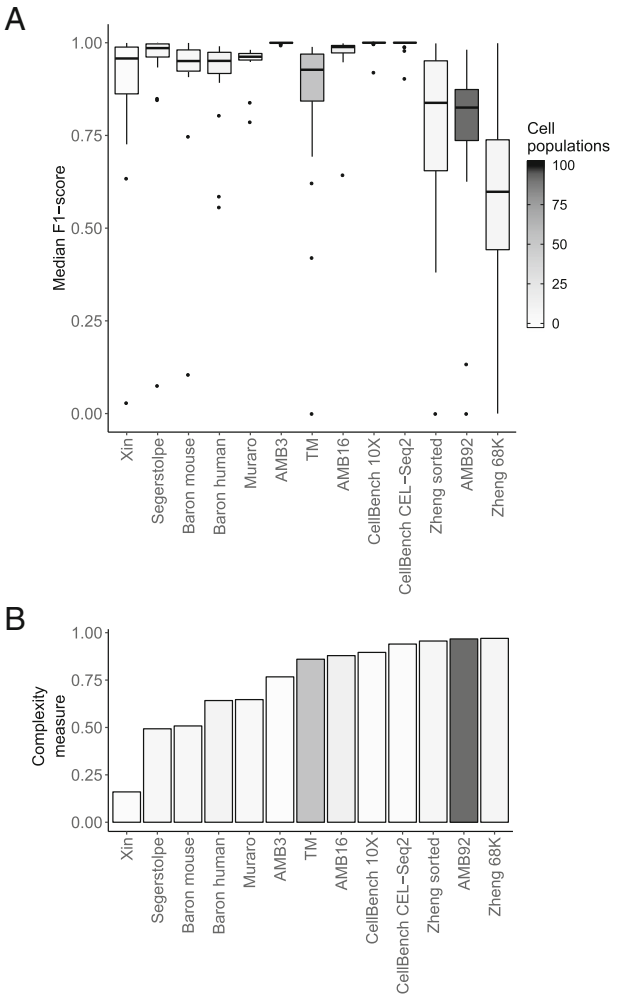


Fig. 2 Complexity of the datasets compared to the performance of the classifiers. **a** Boxplots of the median F1-scores of all classifiers for each dataset used during the intra-dataset evaluation. **b** Barplots describing the complexity of the datasets (see the "Methods" section). Datasets are ordered based on complexity. Box- and bar plots are colored according to the number of cell populations in each dataset

samples (pbmc1 and pbmc2), sequenced using 7 different protocols (Table 2) with the exception that 10Xv3 was not applied to the pbmc2 sample. We used the pbmc1 datasets to evaluate the classification performance of all pairwise train-test combinations between the 7 protocols (42 experiments, see the "Methods" section). Moreover, we extended the evaluation to include comparisons across different samples for the same protocol, using pbmc1 and pbmc2 (6 experiments, see the "Methods" section). All 48 experiment results are summarized in Fig. 3. Overall, several classifiers performed well including $SCINA_{DE}$ using 20 marker genes, *singleCellNet*, *scmapcell*, *scID*, and *SVM*, with an average median F1-score > 0.75 across all 48 experiments (Fig. 3a, Additional file 1: Figure S9A). $SCINA_{DE}$, $Garnett_{DE}$, and $DigitalCellSorter_{DE}$ were tested using 5, 10, 15, and 20 marker genes; Fig. 3a shows the best result for each classifier, where $SCINA_{DE}$ and $Garnett_{DE}$ performed best using 20 and 5 marker genes, respectively, while $DigitalCellSorter_{DE}$ had a median F1-score of 0 during all experiments using all different numbers of marker genes. $DigitalCellSorter_{DE}$ could only identify B cells in the test sets, usually with an F1-score between 0.8 and 1.0, while the F1-score for all other cell populations was 0.

We also tested the prior-knowledge classifiers on all 13 PbmcBench datasets. The prior-knowledge classifiers showed lower performance compared to other classifiers (average median F1-score < 0.6), with the exception of *SCINA* which was only tested on three cell populations (Fig. 3b, Additional file 1: Figure S9B). These results are in line with our previous conclusions from the Zheng sorted and Zheng 68K datasets in the intra-dataset evaluation.

Comparing the performance of the classifiers across the different protocols, we observed a higher performance for all classifiers for specific pairs of protocols. For example, all classifiers performed well when trained on 10Xv2 and

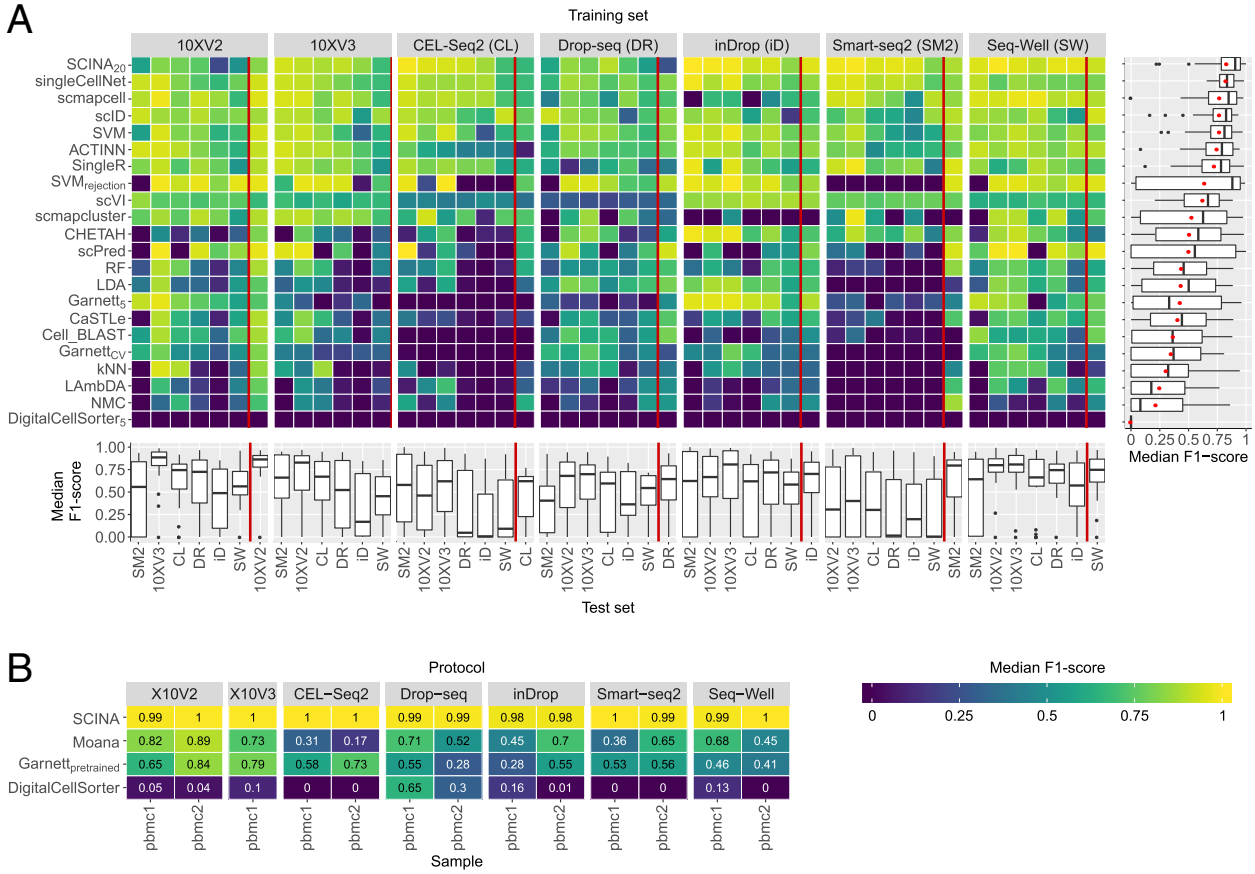


Fig. 3 Classification performance across the PbmcBench datasets. **a** Heatmap showing the median F1-scores of the supervised classifiers for all train-test pairwise combination across different protocols. The training set is indicated in the gray box on top of the heatmap, and the test set is indicated using the column labels below. Results shown to the left of the red line represent the comparison between different protocols using sample pbmc1. Results shown to the right of the red line represent the comparison between different samples using the same protocol, with pbmc 1 used for training and pbmc2 used for testing. Boxplots on the right side of the heatmap summarize the performance of each classifier across all experiments. The mean of the median F1-scores, also used to order the classifiers, is indicated in the boxplots using a red dot. Boxplots underneath the heatmap summarize the performance of the classifiers per experiment. For $SCINA_{DE}$, $Garnett_{DE}$, and $DigitalCellSorter_{DE}$, different numbers of marker genes were tested. Only the best result is shown here. **b** Median F1-score of the prior-knowledge classifiers on both samples of the different protocols. The protocol is indicated in the gray box on top of the heatmap, and the sample is indicated with the labels below. Classifiers are ordered based on their mean performance across all datasets

tested on 10Xv3, and vice versa. On the other hand, other pairs of protocols had a good performance only in one direction, training on Seq-Well produced good predictions on 10Xv3, but not the other way around. Compared to all other protocols, the performance of all classifiers was low when they were either trained or tested on Smart-seq2 data. This can, in part, be due to the fact that Smart-seq2 data does not contain unique molecular identifier (UMI), in contrast to all other protocols.

We also tested the classification performance using the 3 brain datasets, VISp, ALM, and MTG (Table 2), which allowed us to compare the performances across species (mouse and human) as well as single-cell RNA-seq (used in VISp and ALM) vs single-nucleus RNA-seq (used in MTG). We tested all possible train-test combinations for both levels of annotation, three major brain cell types (inhibitory neurons, excitatory neurons, and non-neuronal cells), and the deeper annotation level with 34 cell populations (18 experiments, see the "Methods" section). Prediction of the three major cell types was easy, where almost all classifiers showed high performance (Fig. 4a) with some exceptions. For example, *scPred* failed the classification task completely when testing on the MTG dataset, producing 100% unlabeled cells (Additional file 1: Figure S10A). Predicting the 34 cell populations turned out to be a more challenging task, especially when the MTG human dataset is included either as training or testing data, resulting in significantly lower performance across all classifiers (Fig. 4b). Across all nine experiments at the deeper annotation, the top-performing classifiers were *SVM*, *ACTINN*, *singleCellNet*, *SingleR*, and *LAmbDA*, with almost 0% unlabeled cells (Additional file 1: Figure S10B).

Finally, to evaluate the classification performance across different protocols and different labs, we used the four human pancreatic datasets: Baron Human, Muraro, Segerstople, and Xin (see the "Methods" section, Additional file 1: Table S2). We tested four combinations by training on three datasets and test on one dataset, in which case the classification performance can be affected by batch differences between the datasets. We evaluated the performance of the classifiers when trained using the original data as well as aligned data using the mutual nearest neighbor (MNN) method [41]. Additional file 1: Figure S11 shows UMAPs [42] of the combined dataset before and after alignment, demonstrating better grouping of pancreatic cell types after alignment.

For the original (unaligned) data, the best-performing classifiers across all four experiments are *scVI*, *SVM*, *ACTINN*, *scmapcell*, and *SingleR* (Fig. 5a, Additional file 1: Figure S12A). For the aligned data, the best-performing classifiers are *kNN*, $SVM_{rejection}$, *singleCellNet*, *SVM*, and *NMC* (Fig. 5b, Additional file 1: Figure S12B). Some classifiers benefit from aligning datasets such as $SVM_{rejection}$, *kNN*, *NMC*, and *singleCellNet*, resulting in higher median F1-scores (Fig. 5). On the other hand, some other classifiers failed the classification task completely, such as *scmapcell* which labels all cells as unlabeled. Some other classifiers failed to run over the aligned datasets, such as *ACTINN*, *scVI*, *Cell-BLAST*, *scID*, *scmapcluster*, and *scPred*. These classifiers work only with positive gene expression data, while the aligned datasets contain positive and negative gene expression values.

Rejection option evaluation

Classifiers developed for scRNA-seq data often incorporate a rejection option to identify cell populations in the test set that were not seen during training. These populations cannot be predicted correctly and therefore should remain unassigned. To test whether the classifiers indeed leave these unseen populations unlabeled, we applied two different experiments using negative controls of different tissues and using unseen populations of the same tissue.

First, the classifiers were trained on a data set from one tissue (e.g., pancreas) and used to predict cell populations of a completely different tissue (e.g., brain) [22]. The methods should thus reject all (100%) of the cells in the test dataset. We carried out four different negative control experiments (see the "Methods" section, Fig. 6a). *scmapcluster* and *scPred* have an almost perfect score for all four combinations, rejecting close 100% of the cells. Other top-performing methods for this task, $SVM_{rejection}$ and *scmapcell*, failed when trained on mouse pancreatic data and tested on mouse brain data. All labeled cells of the AMB16 dataset are predicted to be beta cells in this case. The prior-knowledge classifiers, *SCINA*, $Garnett_{pre-trained}$, and *DigitalCellSorter*, could only be tested on the Baron Human pancreatic dataset. $Garnett_{CV}$ could, on top of that, also be trained on the Baron Human dataset and tested on the Zheng 68K dataset. During the training phase, $Garnett_{CV}$ tries to find representative cells for the cell populations described in the marker gene file. Being trained on Baron Human using the PBMC marker gene file, it should not be able to find any representatives, and therefore, all cells in the Zheng 68K dataset should be unassigned. Surprisingly, $Garnett_{CV}$ still finds representatives for PBMC cells in the pancreatic data, and thus, the cells in the test set are labeled. However, being trained on the PBMC dataset and tested on the pancreatic dataset, it does have a perfect performance.

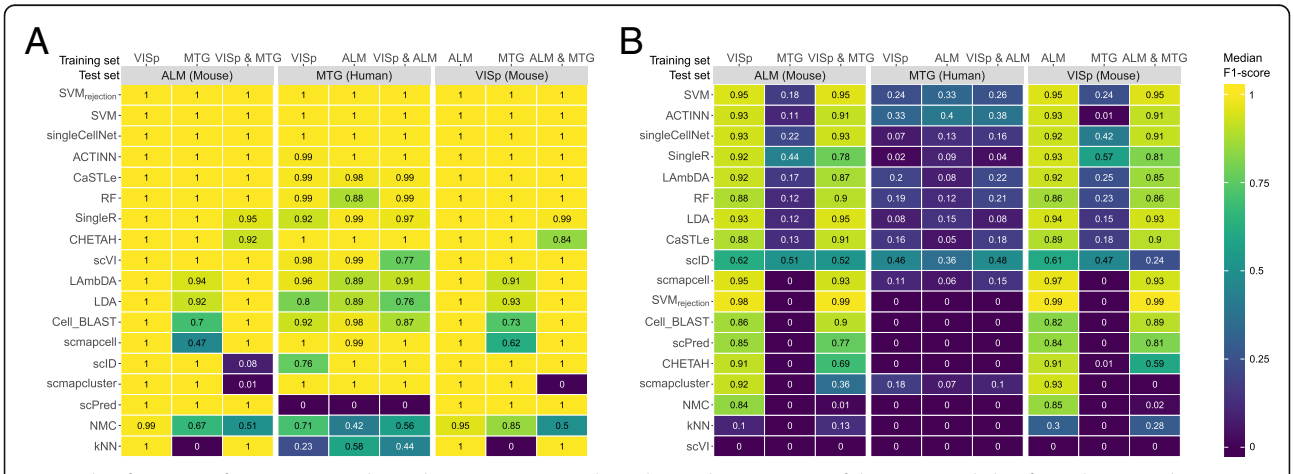


Fig. 4 Classification performance across brain datasets. Heatmaps show the median F1-scores of the supervised classifiers when tested on **a** major lineage annotation with three cell populations and **b** deeper level of annotation with 34 cell populations. The training sets are indicated using the column labels on top of the heatmap. The test set is indicated in the gray box. In each heatmap, the classifiers are ordered based on their mean performance across all experiments

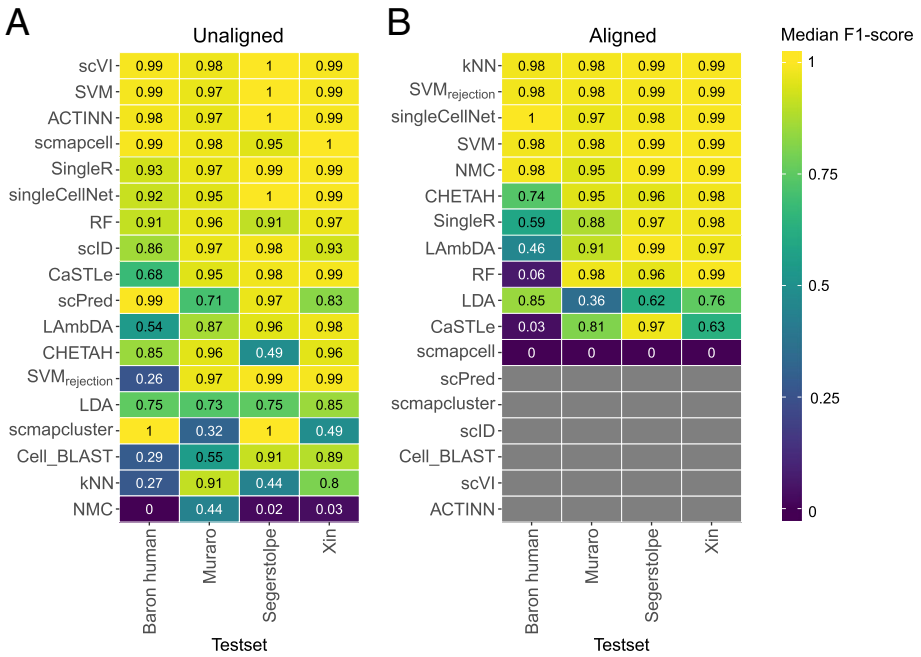


Fig. 5 Classification performance across pancreatic datasets. Heatmaps showing the median F1-score for each classifier for the **a** unaligned and **b** aligned datasets. The column labels indicate which of the four datasets was used as a test set, in which case the other three datasets were used as training. Gray boxes indicate that the corresponding method could not be tested on the corresponding dataset. In each heatmap, the classifiers are ordered based on their mean performance across all experiments

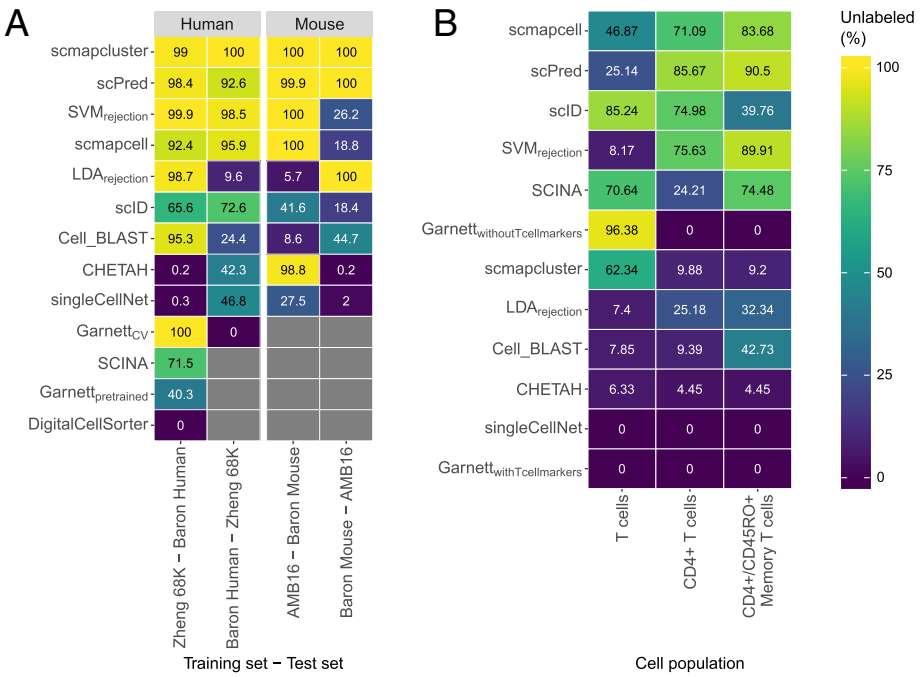


Fig. 6 Performance of the classifiers during the rejection experiments. **a** Percentage of unlabeled cells during the negative control experiment for all the classifiers with a rejection option. The prior-knowledge classifiers could not be tested on all datasets, and this is indicated with a gray box. The species of the dataset is indicated in the gray box on top. Column labels indicate which datasets are used for training and testing. **b** Percentage of unlabeled cells for all classifiers with a rejection option when a cell population was removed from the training set. Column labels indicate which cell population was removed. This cell population was used as a test set. In both **a** and **b**, the classifiers are sorted based on their mean performance across all experiments

To test the rejection option in a more realistic and challenging scenario, we trained the classifiers on some cell populations from one dataset and used the held out cell populations in the test set (see the "Methods" section). Since the cell populations in the test set were not seen during training, they should remain unlabeled. Here, the difficulty of the task was gradually increased (Additional file 1: Table S3). First, all the T cells were removed from the training set. Next, only the CD4+ T cells were removed. Finally, only CD4+/CD45RO+ memory T cells, a subpopulation of the CD4+ T cells, were removed. The top-performing methods for this task are *scmapcell*, *scPred*, *scID*, $SVM_{rejection}$, and *SCINA* (Fig. 6b). We expected that rejecting T cells would be a relatively easy task as they are quite distinct from all other cell populations in the dataset. It should thus be comparable to the negative control experiment. Rejecting CD4+/CD45RO+ memory T cells, on the other hand, would be more difficult as they could easily be confused with all other subpopulations of CD4+ T cells. Surprisingly, almost all classifiers, except for *scID* and *scmapcluster*, show the opposite.

To better understand this unexpected performance, we analyzed the labels assigned by $SVM_{rejection}$. In the first task (T cells removed from the training set), $SVM_{rejection}$ labels almost all T cells as B cells. This can be explained by the fact that $SVM_{rejection}$, and most classifiers for that matter, relies on the classification posterior probabilities to assign labels but ignores the actual similarity between each cell and the assigned population. In task 2 (CD4+ T cells were removed), there were two subpopulations of CD8+ T cells in the training set. In that case, two cell populations are equally similar to the cells in the test set, resulting in low posterior probabilities for both classes and thus the cells in the test set remain unlabeled. If one of these CD8+ T cell populations was removed from the training set, only 10.53% instead of 75.57% of the CD4+ T cells were assigned as unlabeled by $SVM_{rejection}$. All together, our results indicate that despite the importance of incorporating a rejection option in cell identity classifiers, the implementation of this rejection option remains challenging.

Performance sensitivity to the input features

During the intra-datasets cross-validation experiment described earlier, we used all features (genes) as input to the classifiers. However, some classifiers suffer from overtraining when too many features are used. Therefore, we tested the effect of feature selection on the performance of the classifiers. While different strategies for feature selection in scRNA-seq classification experiments exist, selecting genes with a higher number of dropouts compared to the expected number of dropouts has been shown to outperform other methods [22, 43]. We selected subsets of features from the TM dataset using the dropout method. In the experiments, we used the top 100, 200, 500, 1000, 2000, 5000, and 19,791 (all) genes. Some classifiers include a built-in feature selection method which is used by default. To ensure that all methods use the same set of features, the built-in feature selection was turned off during these experiments.

Some methods are clearly overtrained when the number of features increases (Fig. 7a). For example, *scmapcell* shows the highest median F1-score when using less features, and the performance drops when the number of features increases. On the other hand, the performance of other classifiers, such as *SVM*, keeps improving when the number of features increases. These results indicate that the optimal number of features is different for each classifier.

Looking at the median F1-score, there are several methods with a high maximal performance. *Cell-BLAST*, *ACTINN*, *scmapcell*, *scPred*, $SVM_{rejection}$, and *SVM* all have a median F1-score higher than 0.97 for one or more of the feature sets. Some of these well-performing methods, however, leave many cells unlabeled. *scmapcell* and *scPred*, for instance, yield a maximum median F1-score of 0.976 and 0.982, respectively, but 10.7% and 15.1% of the cells are assigned as unlabeled (Fig. 7b). On the other hand, $SVM_{rejection}$ has the highest median F1-score (0.991) overall with only 2.9% unlabeled. Of the top-performing classifiers, only *ACTINN* and *SVM* label all the cells. Overall *SVM* shows the third highest performance with a score of 0.979.

Scalability: performance sensitivity to the number of cells

scRNA-seq datasets vary significantly across studies in terms of the number of cells analyzed. To test the influence of the size of the dataset on the performance of the classifier, we downsampled the TM dataset in a stratified way (i.e., preserving population frequencies) to 1, 5, 10, 20, 50, and 100% of the original number of 45,469 cells (see the "Methods" section) and compared the performance of the classifiers (Fig. 7c, d). Using less than 500 cells in the dataset, most classifiers have a relatively high performance. Only *scID*, *LAmbDA*, *CaSTLe*, and *Cell-BLAST* have a median F1-score below 0.85. Surprisingly, $SVM_{rejection}$ has almost the same median F1-score when using 1% of the data as when using all data (0.993 and 0.994). It must be noted here, however, that the percentage of unlabeled cells decreases significantly (from 28.9% to 1.3%). Overall, the performance of all classifiers stabilized when tested on ≥ 20% (9099 cells) of the original data.

Running time evaluation

To compare the runtimes of the classification methods and see how they scale when the number of cells increases, we compared the number of cells in each dataset with the computation time of the classifiers (Additional file 1: Figure

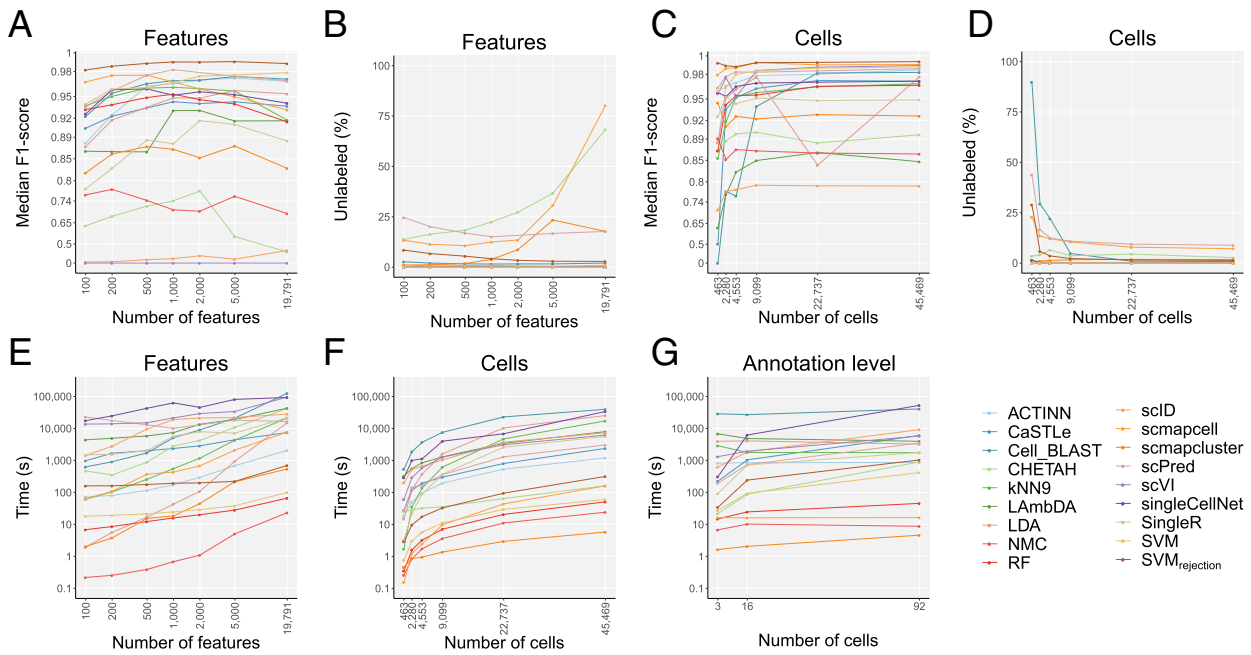


Fig. 7 Computation time evaluation across different numbers of features, cells, and annotation levels. Line plots show **a** the median F1-score, **b** percentage of unlabeled cells, and **e** computation time of each classifier applied to the TM dataset with the top 100, 200, 500, 1000, 2000, 5000, and 19,791 (all) genes as input feature sets. Genes were ranked based on dropout-based feature selection. **c** The median F1-score, **d** percentage of unlabeled cells, and **f** computation time of each classifier applied to the downsampled TM datasets containing 463, 2280, 4553, 9099, 22,737, and 45,469 (all) cells. **g** The computation time of each classifier is plotted against the number of cell populations. Note that the *y*-axis is 100^x scaled in **a** and **c** and log-scaled in **e**–**g**. The *x*-axis is log-scaled in **a**–**f**

S13). Overall, big differences in the computation time can be observed when comparing the different methods. *SingleR* showed the highest computation time overall. Running *SingleR* on the Zheng 68K dataset took more than 39 h, while *scmapcluster* was finished within 10 s on this dataset. Some of the methods have a high runtime for the small datasets. On the smallest dataset, Xin, all classifiers have a computation time < 5 min, with most classifiers finishing within 60 s. *Cell-BLAST*, however, takes more than 75 min. In general, all methods show an increase in computation time when the number of cells increases. However, when comparing the second largest (TM) and the largest (Zheng 68K) datasets, not all methods show an increase in computation time. Despite the increase in the number of cells between the two datasets, *CaSTLe*, *CHETAH*, and *SingleR* have a decreasing computation time. A possible explanation could be that the runtime of these methods also depends on the number of genes or the number of cell populations in the dataset. To evaluate the run time of the methods properly, we therefore investigated the effect of the number of cells, features, and cell populations separately (Fig. 7e–g).

To assess the effect of the number of genes on the computation time, we compared the computation time of the methods during the feature selection experiment (Fig. 7e). Most methods scale linearly with the number of genes. However, *LDA* does not scale very well when the number of genes increases. If the number of features is higher than the number of cells, the complexity of *LDA* is O(*g*^3), where *g* is the number of genes [44].

The effect of the number of cells on the timing showed that all methods increase in computation time when the number of cells increases (Fig. 7f). The differences in runtime on the largest dataset are larger. *scmapcluster*, for instance, takes 5 s to finish, while *Cell-BLAST* takes more than 11 h.

Finally, to evaluate the effect of the number of cell populations, the runtime of the methods on the AMB3, AMB16, and AMB92 datasets was compared (Fig. 7g). For most methods, this shows an increase in runtime when the number of cell populations increases, specially *singleCellNet*. For other methods, such as *ACTINN* and *scmapcell*, the runtime remains constant. Five classifiers, *scmapcell*, *scmapcluster*, *SVM*, *RF*, and *NMC*, have a computation time below 6 min on all the datasets.

Discussion

In this study, we evaluated the performance of 22 different methods for automatic cell identification using 27 scRNA-seq datasets. We performed several experiments to cover different levels of challenges in the classification task and to test specific aspects of the classifiers such as the feature selection, scalability, and rejection experiments. We

summarize our findings across the different experiments (Fig. 8) and provide a detailed summary of which dataset was used for each experiment (Additional file 1: Table S4). This overview can be used as a user guide to choose the most appropriate classifier depending on the experimental setup at hand. Overall, several classifiers performed accurately across different datasets and experiments, particularly $SVM_{rejection}$, *SVM*, *singleCellNet*, *scmapcell*, *scPred*, *ACTINN*, and *scVI*. We observed relatively lower performance for the inter-dataset setup, likely due to the technical and biological differences between the datasets, compared to the intra-dataset setup. $SVM_{rejection}$, *SVM*, and *singleCellNet* performed well for both setups, while *scPred* and *scmapcell* performed better in the intra-dataset setup, and *scVI* and *ACTINN* had a better performance in the inter-dataset setup (Fig. 8). Of note, we evaluated all classifiers using the default settings. While adjusting these settings for a specific dataset might improve the performances, it increases the risk of overtraining.

Considering all three evaluation metrics (median F1-score, percentage of unlabeled cells, and computation time), $SVM_{rejection}$ and *SVM* are overall the best-performing classifiers for the scRNA-seq datasets used. Although *SVM* has a shorter computation time, the high accuracy of the rejection option of $SVM_{rejection}$, which allows flagging new cells and assigning them as unlabeled, results in an improved performance compared to *SVM*. Our results show that $SVM_{rejection}$ and *SVM* scale well to large datasets as well as deep annotation levels. In addition, they did not suffer from the large number of features (genes) present in the data, producing the highest performance on the TM dataset using all genes, due to the incorporated L2 regularization. The comparable or higher overall performance of a general-purpose classier such as *SVM* warrants caution when designing scRNA-seq-specific classifiers that they do not introduce unnecessary complexity. For example, deep learning methods, such as *ACTINN* and *scVI*, showed overall lower performance compared to *SVM*, supporting recent observations by Köhler et al. [45].

scPred (which is based on an SVM with a radial kernel), *LDA*, *ACTINN*, and *singleCellNet* performed well on most datasets, yet the computation time is long for large datasets. *singleCellNet* also becomes slower with a large number of cell populations. Additionally, in some cases, *scPred* and *scmapcell/cluster* reject higher proportions of cells as unlabeled compared to $SVM_{rejection}$, without a substantial improvement in the accuracy. In general, incorporating a rejection option with classification is a good practice to allow the detection of potentially novel cell populations (not present in the training data) and improve the performance for the classified cells with high confidence. However, for the datasets used in this study, the performance of classifiers with a rejection option, except for $SVM_{rejection}$, did not show substantial improvement compared to other classifiers. Furthermore, our results indicate that designing a proper rejection option can be challenging for complex datasets (e.g., PBMC) and that relying on the posterior probabilities alone might not yield optimal results.

For datasets with deep levels of annotation (i.e., large number) of cell populations, the classification performance of all classifiers is relatively low, since the classification task is more challenging. *scVI*, in particular, failed to scale with deeply annotated datasets, although it works well for datasets with a relatively small number of cell populations. Further, applying the prior-knowledge classifiers becomes infeasible for deeply annotated datasets, as the task of defining the marker genes becomes even more challenging.

We evaluated the performance of the prior-knowledge methods (marker-based and pretrained) on PBMC datasets only, due to the limited availability of author-provided marker genes. For all PBMC datasets, the prior-knowledge methods did not improve the classification performance over supervised methods, which do not incorporate such prior knowledge. We extended some prior-knowledge methods such that the marker genes were defined in a data-driven manner using differential expression which did not improve the performance of these classifiers, except for $SCINA_{DE}$ (with 20 marker genes) for the PbmcBench datasets. The data-driven selection of markers allows the prediction of more cell populations compared to the number of populations for which marker genes were originally provided. However, this data-driven selection violates the fundamental assumption in prior-knowledge methods that incorporating expert-defined markers improves classification performance. Further, several supervised classifiers which do not require markers to be defined a priori (e.g., *scPred* and *scID*) already apply a differential expression test to find the best set of genes to use while training the model. The fact that prior-knowledge methods do not outperform other supervised methods and given the challenges associated with explicit marker definition indicate that incorporating prior knowledge in the form of marker genes is not beneficial, at least for PBMC data.

In the inter-dataset experiments, we tested the ability of the classifiers to identify populations across different scRNA-seq protocols. Our results show that some protocols are more compatible with one another (e.g., 10Xv2 and 10Xv3), Smart-Seq2 is distinct from the other UMI-based methods, and CEL-Seq2 suffers from low replicability of cell populations across samples. These results can serve as a guide in order to choose the best set of protocols that can be used in studies where more than one protocol is used.

The intra-dataset evaluation included the Zheng sorted dataset, which consists of 10 FACS-sorted cell populations based on the expression of surface protein markers. Our results show relatively lower classification performance

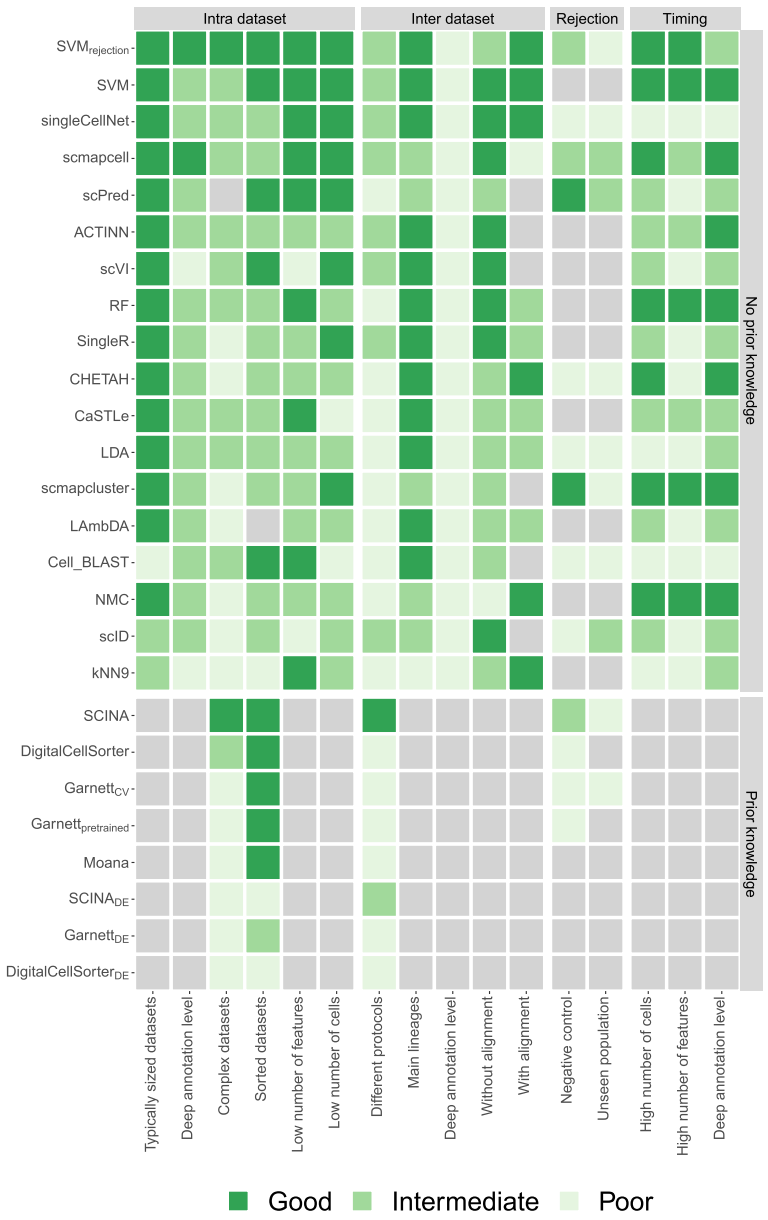


Fig. 8 Summary of the performance of all classifiers during different experiments. For each experiment, the heatmap shows whether a classifier performs good, intermediate, or poor. Light gray indicates that a classifier could not be tested during an experiment. The gray boxes to the right of the heatmap indicate the four different categories of experiments: intra-dataset, inter-dataset, rejection, and timing. Experiments itself are indicated using the row labels. Additional file 1: Table S4 shows which datasets were used to score the classifiers exactly for each experiment. Gray boxes above the heatmap indicate the two classifier categories. Within these two categories, the classifiers are sorted based on their mean performance on the intra- and inter-dataset experiments

compared to other datasets, except the Zheng 68K dataset. The poor correlation between the expression levels of these protein markers and their coding genes mRNA levels [46] might explain this low performance.

Overall, we observed that the performance of almost all methods was relatively high on various datasets, while some datasets with overlapping populations (e.g., Zheng 68K dataset) remain challenging. The inter-dataset comparison requires extensive development in order to deal with technical differences between protocols, batches, and labs, as well as proper matching between different cell population annotations. Further, the pancreatic datasets are known to project very well across studies, and hence, using them to evaluate inter-dataset performance can be misleading. We recommend considering other challenging tissues and cell populations.

Conclusions

We present a comprehensive evaluation of automatic cell identification methods for single-cell RNA sequencing data. Generally, all classifiers perform well across all datasets, including the general-purpose classifiers. In our experiments, incorporating prior knowledge in the form of marker genes does not improve the performance (on PBMC data). We observed large differences in the performance between methods in response to changing the input features. Furthermore, the tested methods vary considerably in their computation time which also varies differently across methods based on the number of cells and features.

Taken together, we recommend the use of the general-purpose $SVM_{rejection}$ classifier (with a linear kernel) since it has a better performance compared to the other classifiers tested across all datasets. Other high-performing classifiers include *SVM* with a remarkably fast computation time at the expense of losing the rejection option, *singleCellNet*, *scmapcell*, and *scPred*. To support the future extension of this benchmarking work with new classifiers and datasets, we provide a Snakemake workflow to automate the performed benchmarking analyses (https://github.com/tabdelaal/scRNAseq_Benchmark/).

Methods

Classification methods

We evaluated 22 scRNA-seq classifiers, publicly available as R or Python packages or scripts (Table 1). This set includes 16 methods developed specifically for scRNA-seq data as well as 6 general-purpose classifiers from the scikit-learn library in Python [29]: linear discriminant analysis (*LDA*), nearest mean classifier (*NMC*), *k*-nearest neighbor (*kNN*), support vector machine (*SVM*) with linear kernel, SVM with rejection option ($SVM_{rejection}$), and random forest (*RF*). The following functions from the scikit-learn library were used respectively: LinearDiscriminantAnalysis(), NearestCentroid(), KNeighborsClassifier(n_neighbors=9), LinearSVC(), LinearSVC() with CalibratedClassifierCV() wrapper, and RandomForestClassifier(n_estimators=50). For *kNN*, 9 neighbors were chosen. After filtering the datasets, only cell populations consisting of 10 cells or more remained. Using 9 neighbors would thus ensure that this classifier could also predict very small populations. For $SVM_{rejection}$, a threshold of 0.7 was used on the posterior probabilities to assign cells as "unlabeled." During the rejection experiments, also an LDA with rejection was implemented. In contrast to the LinearSVC(), the LinearDiscriminantAnalysis() function can output the posterior probabilities, which was also thresholded at 0.7.

scRNA-seq-specific methods were excluded from the evaluation if they did not return the predicted labels for each cell. For example, we excluded *MetaNeighbor* [47] because the tool only returns the area under the receiver operator characteristic curve (AUROC). For all methods, the latest (May 2019) package was installed or scripts were downloaded from their GitHub. For *scPred*, it should be noted that it is only compatible with an older version of Seurat (v2.0). For *CHETAH*, it is important that the R version 3.6 or newer is installed. For *LAmbDA*, instead of the predicted label, the posterior probabilities were returned for each cell population. Here, we assigned the cells to the cell population with the highest posterior probability.

During the benchmark, all methods were run using their default settings, and if not available, we used the settings provided in the accompanying examples or vignettes. As input, we provided each method with the raw count data (after cell and gene filtering as described in the "Data preprocessing" section) according to the method documentation. The majority of the methods have a built-in normalization step. For the general-purpose classifiers, we provided log-transformed counts, $log_2(count + 1)$.

Some methods required a marker gene file or pretrained classifier as an input (e.g., *Garnett*, *Moana*, *SCINA*, *DigitalCellSorter*). In this case, we use the marker gene files or pretrained classifiers provided by the authors. We did not attempt to include additional marker gene files for all datasets, and hence, the evaluation of those methods is restricted to datasets where a marker gene file for cell populations is available.

Datasets

A total of 27 scRNA-seq datasets were used to evaluate and benchmark all classification methods, from which 11 datasets were used for intra-dataset evaluation using a cross-validation scheme, and 22 datasets were used for inter-dataset evaluation, with 6 datasets overlapping for both tasks as described in Table 2. Datasets vary across species (human and mouse), tissue (brain, pancreas, PBMC, and whole mouse), and the sequencing protocol used. The brain datasets, including Allen Mouse Brain (AMB), VISp, ALM (GSE115746), and MTG (phs001790), were downloaded from the Allen Institute Brain Atlas http://celltypes.brain-map.org/rnaseq. All 5 pancreatic datasets were obtained from https://hemberg-lab.github.io/scRNA.seq.datasets/ (Baron Mouse: GSE84133, Baron Human: GSE84133, Muraro: GSE85241, Segerstolpe: E-MTAB-5061, Xin: GSE81608). The CellBench 10X dataset was obtained from (GSM3618014), and the CellBench CEL-Seq2 dataset was obtained from 3 datasets (GSM3618022, GSM3618023, GSM3618024) and concatenated into 1 dataset. The Tabula Muris (TM) dataset was downloaded from https://tabula-muris.ds.czbiohub.org/ (GSE109774). For the Zheng sorted datasets, we downloaded the 10 PBMC-sorted populations (CD14+ monocytes, CD19+ B cells, CD34+ cells, CD4+ helper T cells, CD4+/CD25+ regulatory T cells, CD4+/CD45RA+/CD25− naive T cells, CD4+/CD45RO+ memory T cells,

CD56+ natural killer cells, CD8+ cytotoxic T cells, CD8+/CD45RA+ naive cytotoxic T cells) from https://support.10xgenomics.com/single-cell-gene-expression/datasets; next, we downsampled each population to 2000 cells obtaining a dataset of 20,000 cells in total. For the Zheng 68K dataset, we downloaded the gene-cell count matrix for the "Fresh 68K PBMCs" [36] from https://support.10xgenomics.com/single-cell-gene-expression/datasets (SRP073767). All 13 PbmcBench datasets, 7 different sequencing protocols applied on 2 PBMC samples, were downloaded from the Broad Institute Single Cell portal https://portals.broadinstitute.org/single_cell/study/SCP424/single-cell-comparison-pbmc-data. The cell population annotation for all datasets was provided with the data, except the Zheng 68K dataset, for which we obtained the cell population annotation from https://github.com/10XGenomics/single-cell-3prime-paper/tree/master/pbmc68k_analysis. These annotations were used as a "ground truth" during the evaluation of the cell population predictions obtained from the classification methods.

Data preprocessing

Based on the manual annotation provided in the datasets, we started by filtering out cells that were labeled as doublets, debris, or unlabeled cells. Next, we filtered genes with zero counts across all cells. For cells, we calculated the median number of detected genes per cell, and from that, we obtained the median absolute deviation (MAD) across all cells in the log scale. We filtered out cells when the total number of detected genes was below three MAD from the median number of detected genes per cell. The number of cells and genes in Table 2 represent the size of each dataset after this stage of preprocessing.

Moreover, before applying cross-validation to evaluate each classifier, we excluded cell populations with less than 10 cells across the entire dataset; Table 2 summarizes the number of cell populations before and after this filtration step for each dataset.

Intra-dataset classification

For the supervised classifiers, we evaluated the performance by applying a 5-fold cross-validation across each dataset after filtering genes, cells, and small cell populations. The folds were divided in a stratified manner in order to keep equal proportions of each cell population in each fold. The training and testing folds were exactly the same for all classifiers.

The prior-knowledge classifiers, *Garnett*, *Moana*, *DigitalCellSorter*, and *SCINA*, were only evaluated on the Zheng 68K and Zheng sorted datasets, for which the marker gene files or the pretrained classifiers were available, after filtering genes and cells. Each classifier uses the dataset and the marker gene file as inputs and outputs the cell population label corresponding to each cell. No cross-validation is applied in this case, except for *Garnett* where we could either use the pretrained version ($Garnett_{pretrained}$) provided from the original study, or train our own classifier using the marker gene file along with the training data ($Garnett_{CV}$). In this case, we applied 5-fold cross-validation using the same train and test sets described earlier. Additional file 1: Table S1 shows the mapping of cell populations between the Zheng datasets and each of the prior-knowledge classifiers. For *Moana*, a pretrained classifier was used, this classifier also predicted cells to be memory CD8+ T cells and CD16+ monocytes, while these cell populations were not in the Zheng datasets.

Evaluation of marker genes

The performance and choice of the marker genes per cell population per classifier were evaluated by comparing the F1-score of each cell population with four different characteristics of the marker genes across the cells for that particular cell population: (1) the number of marker genes, (2) the mean expression, (3) the average dropout rate, and (4) the average beta of the marker genes [37]. Beta is a score developed to measure how specific a marker gene for a certain cell population is based on binary expression.

Selecting marker genes using differential expression

Using the cross-validation scheme, training data of each fold was used to select sets of 5, 10, 15, and 20 differentially expressed (DE) marker genes. First, if the data was not already normalized, a CPM read count normalization was applied to the data. Next, the data was log-transformed using $log_2(count + 1)$, and afterwards, the DE test could be applied. As recommended in [48], MAST was used to find the DE genes [49]. The implementation of MAST in the FindAllMarkers() function of Seurat v2.3.0 was used to do a one-vs-all differential expression analysis [50]. Genes returned by Seurat were sorted, and the top 5, 10, 15, or 20 significant genes with a positive fold change were selected as marker genes. These marker genes were then used for population prediction of the test data of the corresponding fold. These marker gene lists can be used by prior-knowledge classifiers such as *SCINA*, $Garnett_{CV}$, and *DigitalCellSorter*, by modifying the cell type marker gene file required as an input to these classifiers. Such modification cannot be applied to the pretrained classifiers of $Garnett_{pretrained}$ and *Moana*.

Dataset complexity

To describe the complexity of a dataset, the average expression of all genes for each cell population (avg_{C_i}) in the dataset was calculated, representing the prototype of each cell population in the full gene space. Next, the pairwise Pearson correlation between these centroids

was calculated $\mathrm{corr}_{\forall i,j}(\mathrm{avg}_{C_i}, \mathrm{avg}_{C_j})$. For each cell population, the highest correlation to another cell population was recorded. Finally, the mean of these per cell population maximum correlations was taken to describe the complexity of a dataset.

$$\text{Complexity} = \text{mean}\left(\max_{\forall i, i \neq j} \underset{\forall i,j}{\text{corr}}\left(\text{avg}_{C_i}, \text{avg}_{C_j}\right)\right)$$

Inter-dataset classification

CellBench

Both CellBench datasets, 10X and CEL-Seq2, were used once as training data and once as test data, to obtain predictions for the five lung cancer cell lines. The common set of detected genes by both datasets was used as features in this experiment.

PbmcBench

Using pbmc1 sample only, we tested all train-test pairwise combinations between all 7 protocols, resulting in 42 experiments. Using both pbmc1 and pbmc2 samples, for the same protocol, we used pbmc1 as training data and pbmc2 as test data, resulting in 6 additional experiments (10Xv3 was not applied for pbmc2). As we are now dealing with PBMC data, we evaluated all classifiers, including the prior-knowledge classifiers, as well as the modified versions of *SCINA*, $Garnett_{CV}$, and *DigitalCellSorter*, in which the marker genes are obtained through differential expression from the training data as previously described. Through all these 48 experiments, genes that are not expressed in the training data were excluded from the feature space. Also, as these PbmcBench datasets differ in the number of cell populations (Table 2), only the cell populations provided by the training data were used for the test data prediction evaluation.

Brain

We used the three brain datasets, VISp, ALM, and MTG with two levels of annotations, 3 and 34 cell populations. We tested all possible train-test combinations, by either using one dataset to train and test on another (6 experiments) or using two concatenated datasets to train and test on the third (3 experiments). A total of 9 experiments were applied for each annotation level. We used the common set of detected genes between the datasets involved in each experiment as features.

Pancreas

We selected the four major endocrine pancreatic cell types (alpha, beta, delta, and gamma) across all four human pancreatic datasets: Baron Human, Muraro, Segerstolpe, and Xin. Additional file 1: Table S2 summarizes the number of cells in each cell type across all datasets. To account for batch effects and technical variations between different protocols, datasets were aligned using MNN [41] from the scran R package (version 1.1.2.0). Using both the raw data (unaligned) and the aligned data, we applied leave-one-dataset-out cross-validation where we train on three datasets and test on the left out dataset.

Performance evaluation metrics

The performance of the methods on the datasets is evaluated using three different metrics: (1) For each cell population in the dataset, the F1-score is reported. The median of these F1-scores is used as a measure for the performance on the dataset. (2) Some of the methods do not label all the cells. These unassigned cells are not considered in the F1-score calculation. The percentage of unlabeled cells is also used to evaluate the performance. (3) The computation time of the methods is also measured.

Feature selection

Genes are selected as features based on their dropout rate. The method used here is based on the method described in [22]. During feature selection, a sorted list of the genes is made. Based on this list, the top n number of genes can be easily selected during the experiments. First, the data is normalized using $log_2(count + 1)$. Next, for each gene, the percentage of dropouts, d, and the mean, m, of the normalized data are calculated. Genes that have a mean or dropout rate of 0 are not considered during the next steps. These genes will be at the bottom of the sorted list. For all other genes, a linear model is fitted to the mean and $\log2(d)$. Based on their residuals, the genes are sorted in descending order and added to the top of the list.

Scalability

For the scalability experiment, we used the TM dataset. To ensure that the dataset could be downsampled without losing cell populations, only the 16 most abundant cell populations were considered during this experiment. We downsampled these cell populations in a stratified way to 1, 5, 10, 20, 50, and 100% of its original size (45,469 cells).

Rejection

Negative control

Two human datasets, Zheng 68K and Baron Human, and two mouse datasets, AMB16 and Baron Mouse, were used. The Zheng 68K dataset was first stratified downsampled to 11% of its original size to reduce computation time. For each species, two different experiments were applied by using one dataset as a training set and the other as a test set and vice versa.

Unseen cell populations

Zheng 68K dataset was stratified downsampled to 11% of its original size to reduce computation time. Three

different experiments were conducted. First, all cell populations that are a subpopulation of T cells were considered the test set. Next, the test set consisted of all subpopulations of CD4+ T cells. Last, only the CD4+/CD45RO+ memory T cells were in the test set. Each time, all cell populations that were not in the test set were part of the training set. Additional file 1: Table S3 gives an exact overview of the populations per training and test set.

Benchmarking pipeline

In order to ensure reproducibility and support the future extension of this benchmarking work with new classification methods and benchmarking datasets, a Snakemake [51] workflow for automating the performed benchmarking analyses was developed with an MIT license (https://github.com/tabdelaal/scRNAseq_Benchmark/). Each tool (license permitting) is packaged in a Docker container (https://hub.docker.com/u/scrnaseqbenchmark) alongside the wrapper scripts and their dependencies. These images will be used through Snakemake's singularity integration to allow the workflow to be run without the requirement to install specific methods and to ensure reproducibility. Documentation is also provided to execute and extend this benchmarking workflow to help researchers to further evaluate interested methods.

Additional files

Additional file 1 Supplementary data, Tables S1-S4 and Figures S1–13. (PDF 12800 kb)

Additional file 2 Review history. (DOCX 42 kb)

Acknowledgements
Not applicable.

Review history
The review history is available as Additional file 2.

Authors' contributions
TA, LM, MJTR, and AM conceived the study and designed the experiments. TA and LM performed the experiments. DH, DC, and HM designed and developed the Snakemake workflow. MJTR and AM supervised the experiments. TA, LM, HM, and AM wrote the manuscript. All authors reviewed and approved the manuscript.

Funding
This work was supported by the European Commission of a H2020 MSCA award under proposal number [675743] (ISPIC).

Availability of data and materials
The filtered datasets analyzed during the current study can be downloaded from Zenodo (https://doi.org/10.5281/zenodo.3357167). The source code is available in the GitHub repository, at https://github.com/tabdelaal/scRNAseq_Benchmark [52], and in the Zenodo repository, at https://doi.org/10.5281/zenodo.3369158 [53]. The source code is released under MIT license. Datasets accession numbers: AMB, VISp, and ALM [35] (GSE115746), MTG [31] (phs001790), Baron Mouse [30] (GSE84133), Baron Human [30] (GSE84133), Muraro [31] (GSE85241), Segerstolpe [32] (E-MTAB-5061), Xin [33] (GSE81608), CellBench 10X [34] (GSM3618014), CellBench CEL-Seq2 [34] (GSM3618022, GSM3618023, GSM3618024), TM [6] (GSE109774), and Zheng sorted and Zheng 68K [36] (SRP073767). The PbmcBench datasets [38] are not yet uploaded to any data repository.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Leiden Computational Biology Center, Leiden University Medical Center, Einthovenweg 20, 2333 ZC Leiden, The Netherlands. [2]Delft Bioinformatics Laboratory, Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands. [3]Sequencing Analysis Support Core, Department of Biomedical Data Sciences, Leiden University Medical Center, Einthovenweg 20, 2333 ZC Leiden, The Netherlands.

Received: 18 May 2019 Accepted: 17 August 2019
Published online: 09 September 2019

References
1. Svensson V, Vento-Tormo R, Teichmann SA. Exponential scaling of single-cell RNA-seq in the past decade. Nat Protoc. 2018;13:599–604. https://doi.org/10.1038/nprot.2017.149.
2. Plass M, Solana J, Wolf FA, Ayoub S, Misios A, Glažar P, et al. Cell type atlas and lineage tree of a whole complex animal by single-cell transcriptomics. Science. 2018;360. https://doi.org/10.1126/science.aaq1723.
3. Cao J, Packer JS, Ramani V, Cusanovich DA, Huynh C, Daza R, et al. Comprehensive single-cell transcriptional profiling of a multicellular organism. Science. 2017;357:661–667. https://doi.org/10.1126/science.aam8940.
4. Fincher CT, Wurtzel O, de Hoog T, Kravarik KM, Reddien PW. Cell type transcriptome atlas for the planarian. Science. 2018;360. https://doi.org/10.1126/science.aaq1736.
5. Han X, Wang R, Zhou Y, Fei L, Sun H, Lai S, et al. Mapping the Mouse Cell Atlas by Microwell-Seq. Cell. 2018;173:1307. https://doi.org/10.1016/j.cell.2018.05.012.
6. Schaum N, Karkanias J, Neff NF, May AP, Quake SR, Wyss-Coray T, et al. Single-cell transcriptomics of 20 mouse organs creates a Tabula Muris. Nature. 2018;562:367–372. https://doi.org/10.1038/s41586-018-0590-4.
7. Cao J, Spielmann M, Qiu X, Huang X, Ibrahim DM, Hill AJ, et al. The single-cell transcriptional landscape of mammalian organogenesis. Nature. 2019;566:496–502. https://doi.org/10.1038/s41586-019-0969-x.
8. Henry VJ, Bandrowski AE, Pepin A-S, Gonzalez BJ, Desfeux A. OMICtools: an informative directory for multi-omic data analysis. Database. 2014;2014. https://doi.org/10.1093/database/bau069.
9. Zappia L, Phipson B, Oshlack A. Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database. PLoS Comput Biol. 2018;14:e1006245. https://doi.org/10.1371/journal.pcbi.1006245.
10. Saelens W, Cannoodt R, Todorov H, Saeys Y. A comparison of single-cell trajectory inference methods. Nat Biotechnol. 2019;37:547–554. https://doi.org/10.1038/s41587-019-0071-9.
11. Duò A, Robinson MD, Soneson C. A systematic performance evaluation of clustering methods for single-cell RNA-seq data. F1000Res. 2018;7:1141. https://doi.org/10.12688/f1000research.15666.2.
12. Soneson C, Robinson MD. Bias, robustness and scalability in single-cell differential expression analysis. Nat Methods. 2018;15:255–261. https://doi.org/10.1038/nmeth.4612.
13. Diaz-Mejia JJ, Javier Diaz-Mejia J, Meng EC, Pico AR, MacParland SA, Ketela T, et al. Evaluation of methods to assign cell type labels to cell clusters from single-cell RNA-sequencing data. 2019. https://doi.org/10.1101/562082.
14. Pliner HA, Shendure J, Trapnell C. Supervised classification enables rapid annotation of cell atlases. bioRxiv. 2019; 538652. https://doi.org/10.1101/538652.
15. Wagner F, Yanai I. Moana: A robust and scalable cell type classification framework for single-cell RNA-Seq data. bioRxiv. 2018; 456129. https://doi.org/10.1101/456129.
16. Domanskyi S, Szedlak A, Hawkins NT, Wang J, Paternostro G, Piermarocchi C. Polled Digital Cell Sorter (p-DCS): automatic identification of hematological

cell types from single cell RNA-sequencing clusters. bioRxiv. 2019; 539833. https://doi.org/10.1101/539833.
17. Zhang Z, Luo D, Zhong X, Choi JH, Ma Y, Mahrt E, et al. SCINA: semi-supervised analysis of single cells in silico. bioRxiv. 2019; 559872. https://doi.org/10.1101/559872.
18. Lopez R, Regier J, Cole MB, Jordan MI, Yosef N. Deep generative modeling for single-cell transcriptomics. Nat Methods. 2018;15:1053–1058. https://doi.org/10.1038/s41592-018-0229-2.
19. Cao Z-J, Wei L, Lu S, Yang D-C, Gao G. Cell BLAST: searching large-scale scRNA-seq databases via unbiased cell embedding. bioRxiv. 2019; 587360. https://doi.org/10.1101/587360.
20. Ma F, Pellegrini M. Automated identification of cell types in single cell RNA sequencing. bioRxiv. 2019; 532093. https://doi.org/10.1101/532093.
21. Johnson TS, Wang T, Huang Z, Yu CY, Wu Y, Han Y, et al. LAmbDA: label ambiguous domain adaptation dataset integration reduces batch effects and improves dsubtype detection. Bioinformatics. 2019. https://doi.org/10.1093/bioinformatics/btz295.
22. Kiselev VY, Yiu A, Hemberg M. scmap: projection of single-cell RNA-seq data across data sets. Nat Methods. 2018;15:359. https://doi.org/10.1038/nmeth.4644.
23. Alquicira-Hernandez J, Nguyen Q, Powell JE. scPred: scPred: cell type prediction at single-cell resolution. bioRxiv. 2018; 369538. https://doi.org/10.1101/369538.
24. Kanter JK de, Lijnzaad P, Candelli T, Margaritis T, Holstege F. CHETAH: a selective, hierarchical cell type identification method for single-cell RNA sequencing. bioRxiv. 2019; 558908. https://doi.org/10.1101/558908.
25. Lieberman Y, Rokach L, Shay T. CaSTLe – classification of single cells by transfer learning: harnessing the power of publicly available single cell RNA sequencing experiments to annotate new experiments. PLoS One. 2018;13:e0205499. https://doi.org/10.1371/journal.pone.0205499.
26. Aran D, Looney AP, Liu L, Wu E, Fong V, Hsu A, et al. Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage. Nat Immunol. 2019;20:163–172. https://doi.org/10.1038/s41590-018-0276-y.
27. Boufea K, Seth S, Batada NN. scID: identification of equivalent transcriptional cell populations across single cell RNA-seq data using discriminant analysis. https://doi.org/10.1101/470203.
28. Tan Y, Cahan P. SingleCellNet: a computational tool to classify single cell RNA-Seq data across platforms and across species. bioRxiv. 2018; 508085. https://doi.org/10.1101/508085.
29. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. JMLR. 2011;12:2825–30.
30. Baron M, Veres A, Wolock SL, Faust AL, Gaujoux R, Vetere A, et al. A single-cell transcriptomic map of the human and mouse pancreas reveals inter- and intra-cell population structure. Cell Syst. 2016;3:346–60.e4. https://doi.org/10.1016/j.cels.2016.08.011.
31. Muraro MJ, Dharmadhikari G, Grün D, Groen N, Dielen T, Jansen E, et al. A single-cell transcriptome atlas of the human pancreas. Cell Syst. 2016;3:385–94.e3. https://doi.org/10.1016/j.cels.2016.09.002.
32. Segerstolpe Å, Palasantza A, Eliasson P, Andersson E-M, Andréasson A-C, Sun X, et al. Single-cell transcriptome profiling of human pancreatic islets in health and type 2 diabetes. Cell Metab. 2016;24:593–607. https://doi.org/10.1016/j.cmet.2016.08.020.
33. Xin Y, Kim J, Okamoto H, Ni M, Wei Y, Adler C, et al. RNA sequencing of single human islet cells reveals type 2 diabetes genes. Cell Metab. 2016;24:608–615. https://doi.org/10.1016/j.cmet.2016.08.018.
34. Tian L, Dong X, Freytag S, Lê Cao K-A, Su S, JalalAbadi A, et al. Benchmarking single cell RNA-sequencing analysis pipelines using mixture control experiments. Nat Methods. 2019;16:479–487. https://doi.org/10.1038/s41592-019-0425-8.
35. Tasic B, Yao Z, Graybuck LT, Smith KA, Nguyen TN, Bertagnolli D, et al. Shared and distinct transcriptomic cell types across neocortical areas. Nature. 2018;563:72–78. https://doi.org/10.1038/s41586-018-0654-5.
36. Zheng GXY, Terry JM, Belgrader P, Ryvkin P, Bent ZW, Wilson R, et al. Massively parallel digital transcriptional profiling of single cells. Nat Commun. 2017;8:14049. https://doi.org/10.1038/ncomms14049.
37. Hodge RD, Bakken TE, Miller JA, Smith KA, Barkan ER, Graybuck LT, et al. Conserved cell types with divergent features between human and mouse cortex. Nature. 2019. https://doi.org/10.1038/s41586-019-1506-7.
38. Ding J, Adiconis X, Simmons SK, Kowalczyk MS, Hession CC, Marjanovic ND, et al. Systematic comparative analysis of single cell RNA-sequencing methods. bioRxiv. 2019; 632216. https://doi.org/10.1101/632216.
39. Franzén O, Gan L-M, Björkegren JLM. PanglaoDB: a web server for exploration of mouse and human single-cell RNA sequencing data. Database. 2019;2019. https://doi.org/10.1093/database/baz046.
40. Zhang X, Lan Y, Xu J, Quan F, Zhao E, Deng C, et al. CellMarker: a manually curated resource of cell markers in human and mouse. Nucleic Acids Res. 2019;47:D721–D728. https://doi.org/10.1093/nar/gky900.
41. Haghverdi L, Lun ATL, Morgan MD, Marioni JC. Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. Nat Biotechnol. 2018;36:421–427. https://doi.org/10.1038/nbt.4091.
42. McInnes L, Healy J, Melville JUMAP. Uniform manifold approximation and projection for dimension reduction. arXiv [stat.ML]. 2018; http://arxiv.org/abs/1802.03426.
43. Andrews TS, Hemberg M. M3Drop: dropout-based feature selection for scRNASeq. Bioinformatics. 2018. https://doi.org/10.1093/bioinformatics/bty1044.
44. D. Cai, X. He, J. Han. Training linear discriminant analysis in linear time. 2008. https://doi.org/10.1109/ICDE.2008.4497429.
45. Köhler ND, Büttner M, Theis FJ. Deep learning does not outperform classical machine learning for cell-type annotation. bioRxiv. 2019; 653907. https://doi.org/10.1101/653907.
46. van den Berg PR, Budnik B, Slavov N, Semrau S. Dynamic post-transcriptional regulation during embryonic stem cell differentiation. bioRxiv. 2017; 123497. https://doi.org/10.1101/123497.
47. Crow M, Paul A, Ballouz S, Huang ZJ, Gillis J. Characterizing the replicability of cell types defined by single cell RNA-sequencing data using MetaNeighbor. Nat Commun. 2018;9:884. https://doi.org/10.1038/s41467-018-03282-0.
48. Luecken MD, Theis FJ. Current best practices in single-cell RNA-seq analysis: a tutorial. Mol Syst Biol. 2019;15:e8746. https://doi.org/10.15252/msb.20188746.
49. Finak G, McDavid A, Yajima M, Deng J, Gersuk V, Shalek AK, et al. MAST: a flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data. Genome Biol. 2015;16:278. https://doi.org/10.1186/s13059-015-0844-5.
50. Butler A, Hoffman P, Smibert P, Papalexi E, Satija R. Integrating single-cell transcriptomic data across different conditions, technologies, and species. Nat Biotechnol. 2018;36:411–420. https://doi.org/10.1038/nbt.4096.
51. Köster J, Rahmann S. Snakemake—a scalable bioinformatics workflow engine. Bioinformatics. 2018;34:3600–3600. https://doi.org/10.1093/bioinformatics/bty350.
52. Abdelaal T, Michielsen L, Cats D, Hoogduin D, Mei H, Reinders MJT, et al. scRNA-seq classification benchmarking source code. Github. 2019. https://github.com/tabdelaal/scRNAseq_Benchmark.
53. Abdelaal T, Michielsen L, Cats D, Hoogduin D, Mei H, Reinders MJT, et al. scRNA-seq classification benchmarking source code: Zenodo; 2019. https://doi.org/10.5281/zenodo.3369158.

Publisher's Note

Supplementary Data for

A comparison of automatic cell identification methods for single-cell RNA-sequencing data

Tamim Abdelaal[1,2†] (t.r.m.abdelaal-1@tudelft.nl)

Lieke Michielsen[1,2†] (l.c.m.michielsen@student.tudelft.nl)

Davy Cats[3] (d.cats@lumc.nl)

Dylan Hoogduin[3] (ddhoogduin@gmail.com)

Hailiang Mei[3] (H.Mei@lumc.nl)

Marcel J.T. Reinders[1,2] (m.j.t.reinders@tudelft.nl)

Ahmed Mahfouz[1,2*] (a.mahfouz@lumc.nl)

[1] Leiden Computational Biology Center, Leiden University Medical Center, Einthovenweg 20, 2333ZC, Leiden, The Netherlands

[2] Delft Bioinformatics Lab, Delft University of Technology, Van Mourik Broekmanweg 6, 2628XE, Delft, The Netherlands

[3] Sequencing Analysis Support Core, Department of Biomedical Data Sciences, Einthovenweg 20, 2333ZC, Leiden University Medical Center, Leiden, The Netherlands

[†] Equal contribution

* Corresponding author (a.mahfouz@lumc.nl)

Table S1. Mapping of true cell population labels from PBMC datasets to cell population labels of the prior-knowledge classifiers.

True Labels	**Garnett**	**DigitalCellSorter**	**Moana**	**SCINA**
CD14+ Monocyte	CD14+ Monocyte	CD14+ Monocyte	CD14+ Monocyte	CD14+ Monocyte
Dendritic	Dendritic	Dendritic	Dendritic	
CD34+	CD34+			
CD56+ NK	CD56+ NK	CD56+ NK	CD56+ NK	CD56+ NK
CD19+ B	CD19+ B	CD19+ B	CD19+ B	CD19+ B
CD4+ T Helper 2	CD4+ T cell	T cell		
CD4+/CD25 T Reg				
CD4+/CD45RA+/CD25- Naïve T			Naïve CD4+ T cells	
CD4+/CD45RO+ Memory			Memory CD4+ T cells	
CD8+ Cytotoxic T	CD8+ T cell			
CD8+/CD45RA+ Naïve Cytotoxic			Naïve CD8+ T cells	
			Memory CD8+ T cells	
			CD16+ Monocytes	

Table S2. Cell type size for each pancreatic dataset used in the across dataset performance evaluation.

Dataset	alpha	beta	delta	gamma	Total
Baron (Human)	2326	2525	601	255	**5707**
Muraro	812	448	193	101	**1554**
Segerstolpe	872	263	110	195	**1440**
Xin	855	466	46	82	**1449**
Total	**4865**	**3702**	**950**	**633**	**10150**

Table S3. Cell populations in the training and test set during the rejection experiment applied on the Zheng 68K dataset.

	Training set	**Test set**
T cells	CD19+ B CD56+ NK Dendritic CD14+ Monocyte CD34+	CD8+ Cytotoxic T CD8+/CD45RA+ Naive Cytotoxic T CD4+/CD25 T reg CD4+/CD45RO+ Memory T CD4+/CD45RA+/CD25- Naive T CD4+ T Helper
CD4+ T cells	CD19+ B CD56+ NK Dendritic CD14+ Monocyte CD34+ CD8+ Cytotoxic T CD8+/CD45RA+ Naive Cytotoxic T	CD4+/CD25 T reg CD4+/CD45RO+ Memory T CD4+/CD45RA+/CD25- Naive T CD4+ T Helper
CD4+/CD45RO+ Memory T cells	CD19+ B CD56+ NK Dendritic CD14+ Monocyte CD34+ CD8+ Cytotoxic T CD8+/CD45RA+ Naive Cytotoxic T CD4+/CD25 T reg CD4+/CD45RA+/CD25- Naive T CD4+ T Helper	CD4+/CD45RO+ Memory T

Table S4. Datasets used to score the performance of the classifiers per experiment in Figure 8.

Category	Experiment	Datasets
Intra dataset	Typically sized datasets	Baron Human Baron Mouse Xin Muraro Segerstolpe
	Deep annotation level	AMB92
	Complex datasets	Zheng 68K
	Sorted datasets	Zheng sorted
	Low number of features	TM with 100 features
	Low number of cells	TM downsampled to 1% (463 cells)
Inter dataset	Different protocols	Across PBMC benchmark
	Main lineages	Across brain - 3 levels of annotations
	Deep annotation level	Across brain - 34 levels of annotations
	Without alignment	Across pancreas - unaligned
	With alignment	Across pancreas - aligned
Rejection	Negative control	Downsampled Zheng 68K Baron Human AMB16 Baron Mouse
	Unseen population	Downsampled Zheng 68K
Timing	High number of cells	TM with most abundant 16 populations (45,469 cells)
	High number of features	TM with all features
	Deep annotation level	AMB92

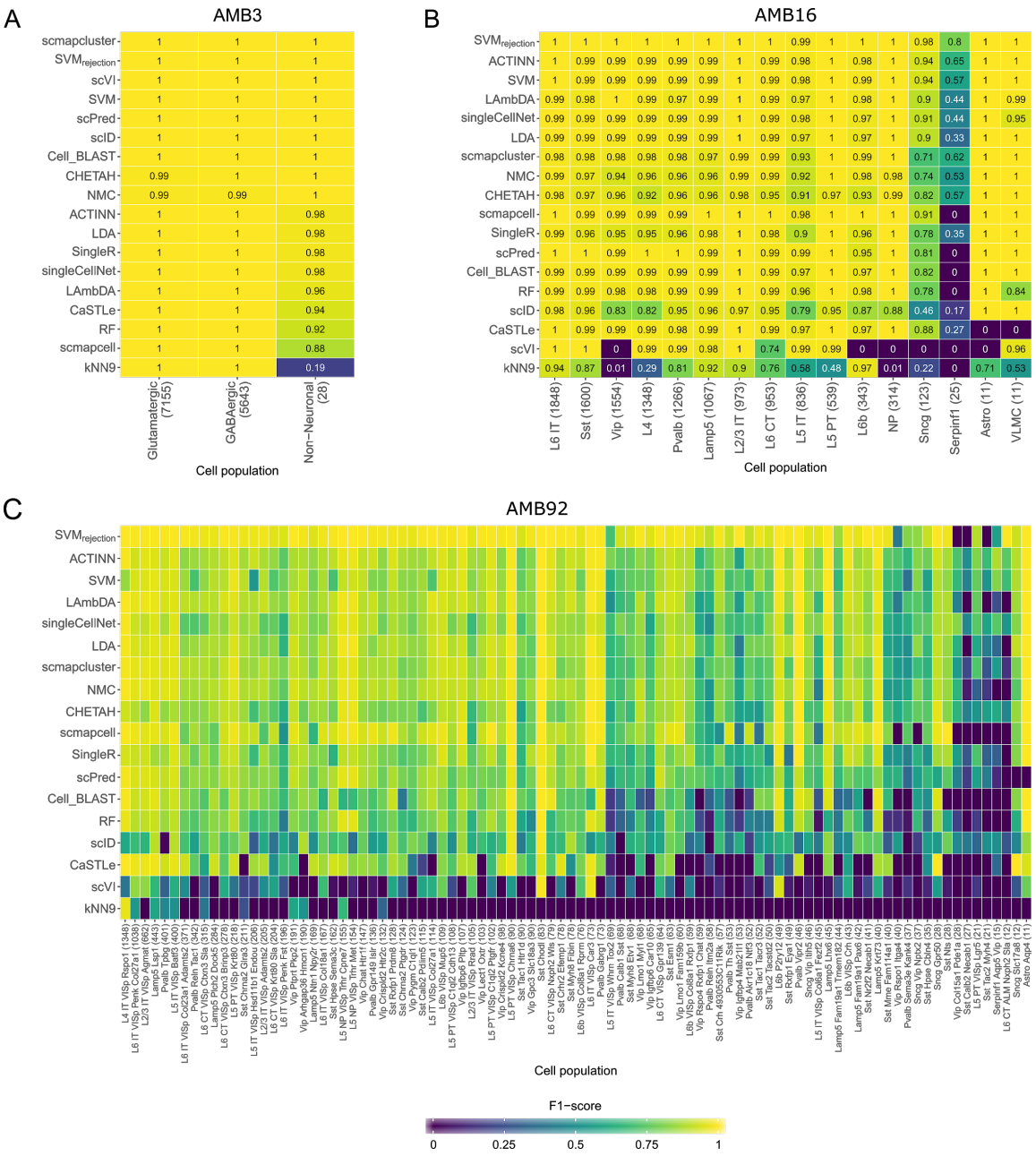


Figure S1. Classification performance across different annotation levels in the Allen Mouse Brain dataset. Heatmaps show the F1-scores of each classifier for each cell population in the **(A)** AMB3, **(B)** AMB16, and **(C)** AMB92 datasets. The cell populations are sorted from left-to-right in descending order according to their size (i.e. number of cells). The size of each population is indicated between brackets. In each heatmap, the classifiers are sorted according to their mean performance across all cell populations.

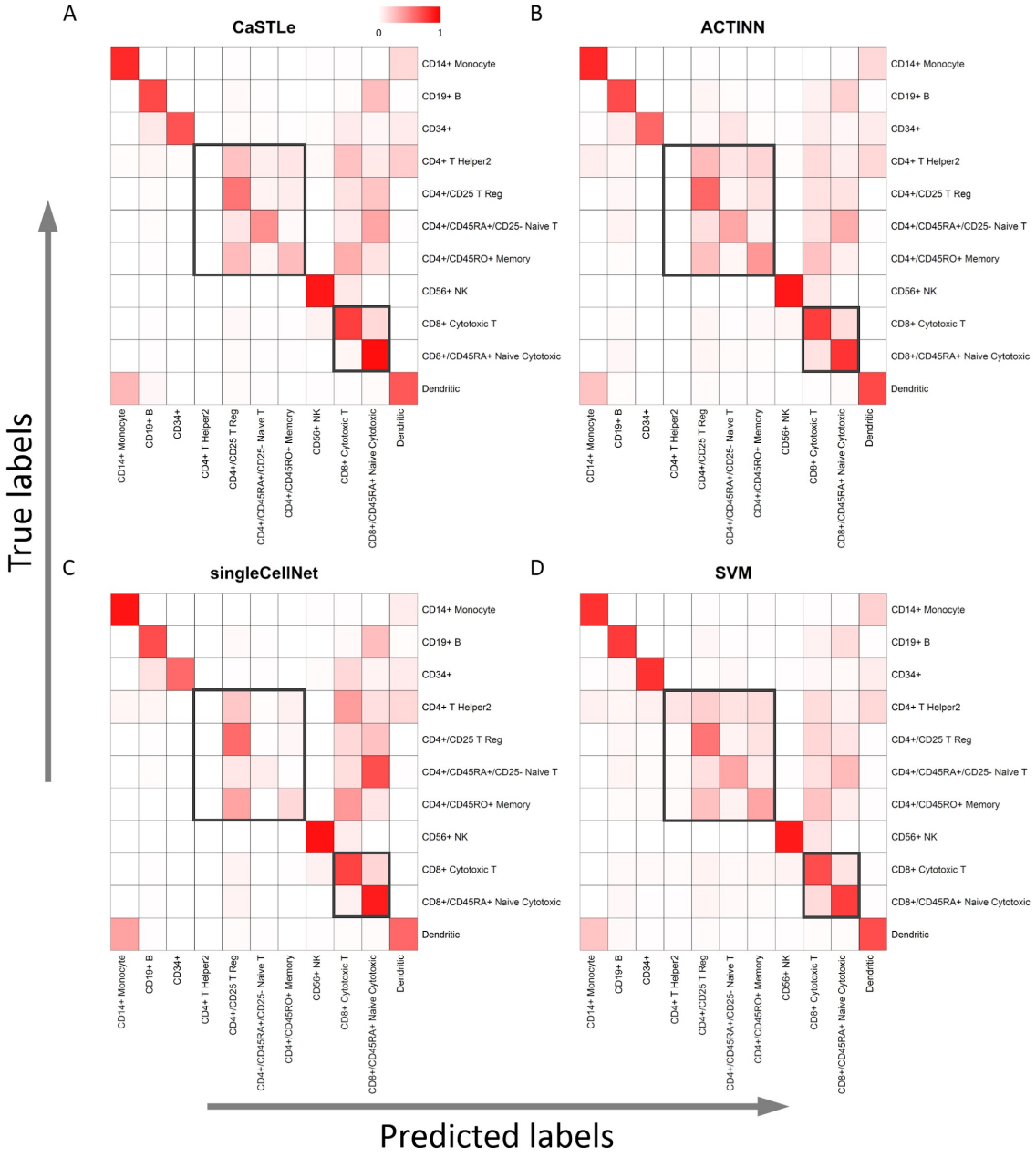


Figure S2. Confusion matrices for the Zheng 68K dataset. Results of four classifiers, **(A)** *CaSTLe*, **(B)** *ACTINN*, **(C)** *singleCellNet*, and **(D)** *SVM*, are shown. Rows indicate the true labels and columns indicate the predicted labels. Each cell in the heatmap is colored according to the percentage of overlapping cells between the true and predicted cell population. Black boxes highlight the four subpopulations of CD4 and the two subpopulations of CD8 T-cells.

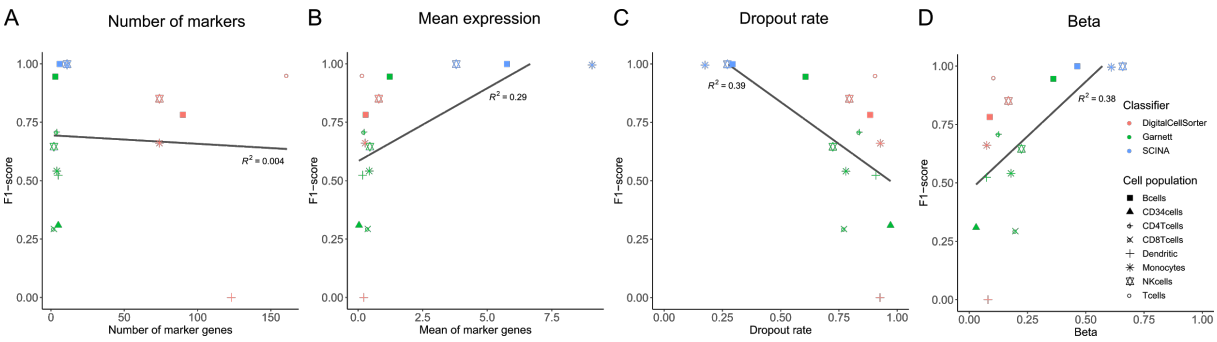


Figure S3. Effect of marker-genes on the performance of the classifiers. Scatterplots compare the **(A)** number of marker-genes, **(B)** mean expression, **(C)** dropout rate, and **(D)** beta, a measure for the specificity, with the performance of the marker based classifiers. Different classifiers are indicated with different colors, different cell populations with different shapes.

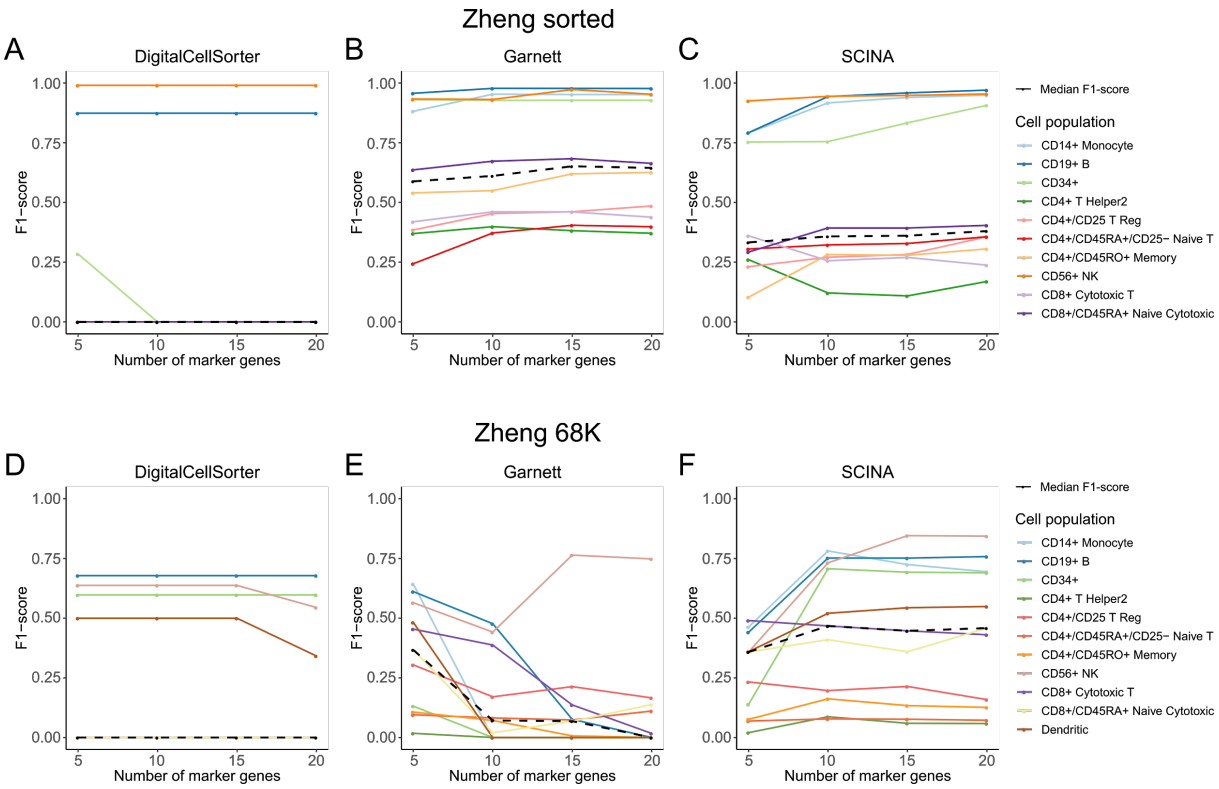


Figure S4. Performance of marker-based classifiers using differentially expressed genes. Line plots show the performance of marker-based classifiers using different number of marker-genes on the **(A-C)** Zheng sorted and **(D-F)** Zheng 68K dataset. marker-genes were selected using differential expression. Different cell populations are indicated using different colors. The median F1-score of the classifier is indicated using a dashed black line.

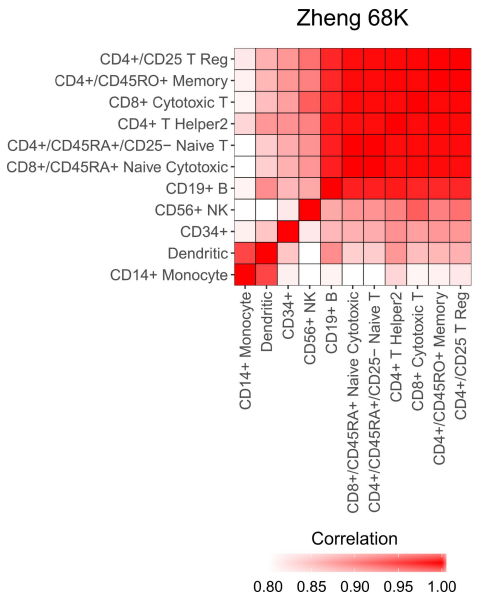


Figure S5. Correlation between cell populations in the Zheng 68K dataset. Heatmap showing the pairwise Pearson correlation between the different cell populations in the Zheng 68K dataset.

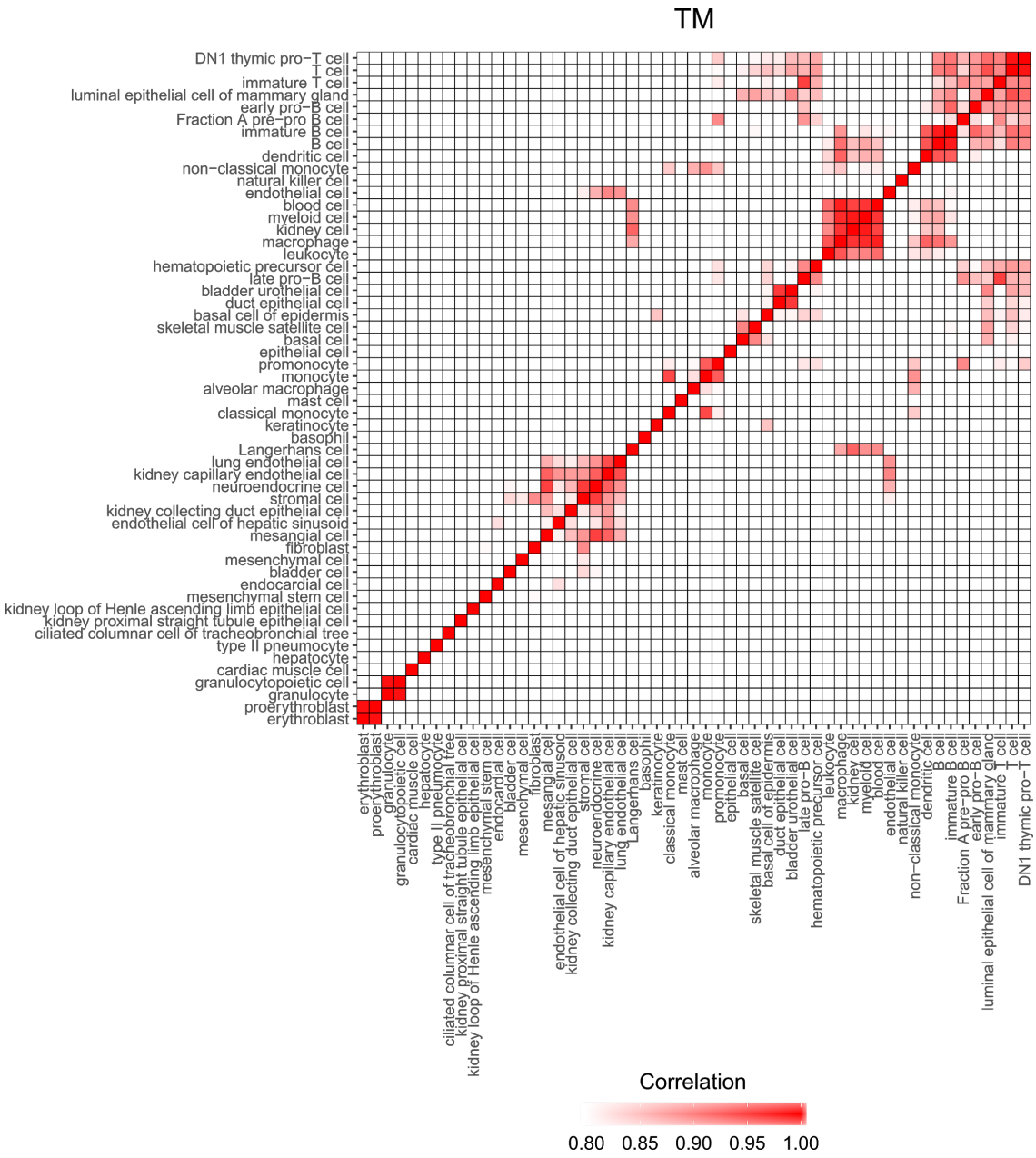


Figure S6. Correlation between cell populations in the TM dataset. Heatmap showing the pairwise Pearson correlation between the different cell populations in the TM dataset.

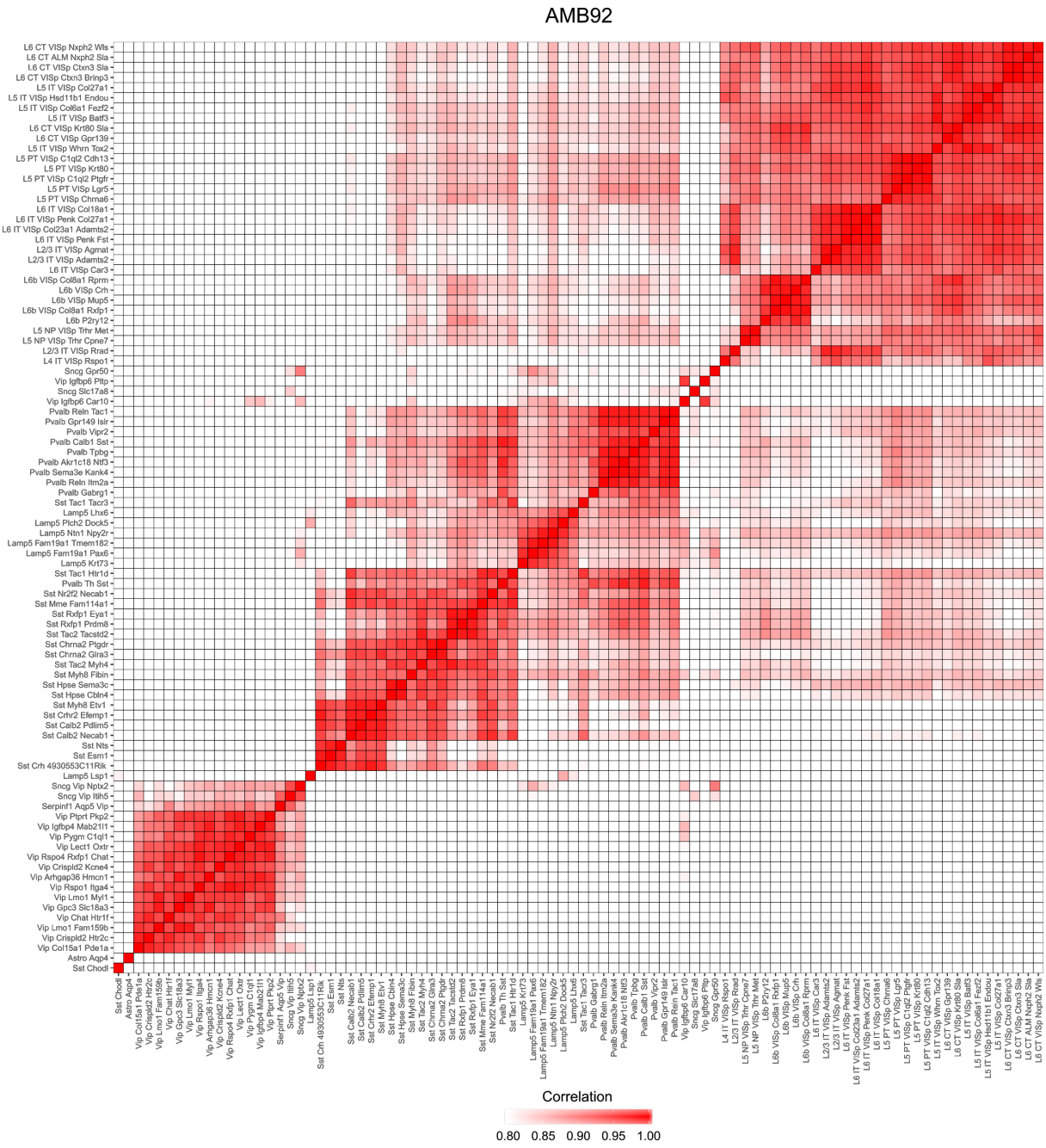


Figure S7. Correlation between cell populations in the AMB92 dataset. Heatmap showing the pairwise Pearson correlation between the different cell populations in the AMB92 dataset.

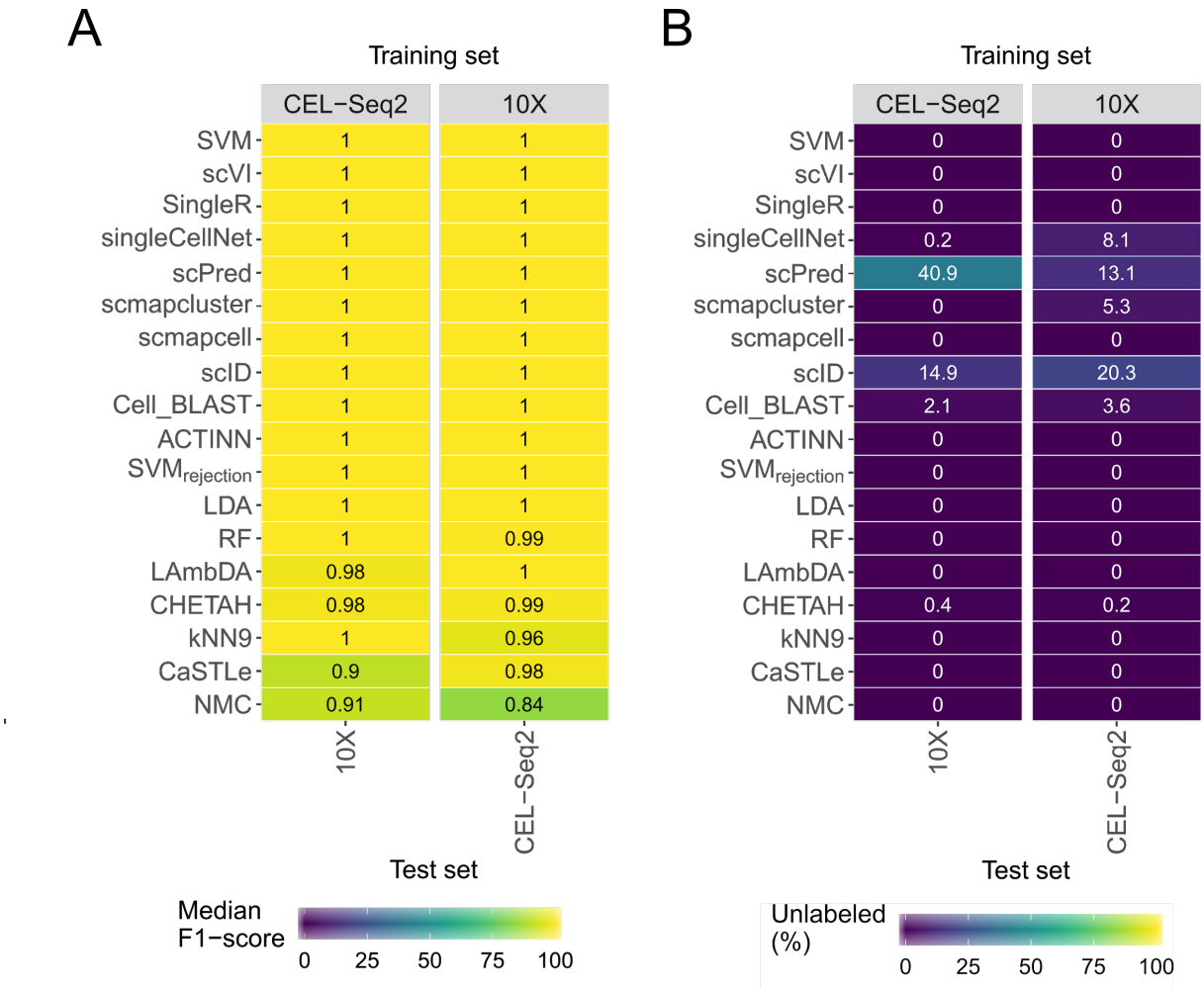


Figure S8. Classification performance across the CellBench datasets. Heatmaps show the **(A)** median F1-score and **(B)** percentage of unlabeled cells across the CellBench datasets. The training set is indicated above the heatmap, the test set below. Classifiers are sorted based on their mean performance in **(A).**

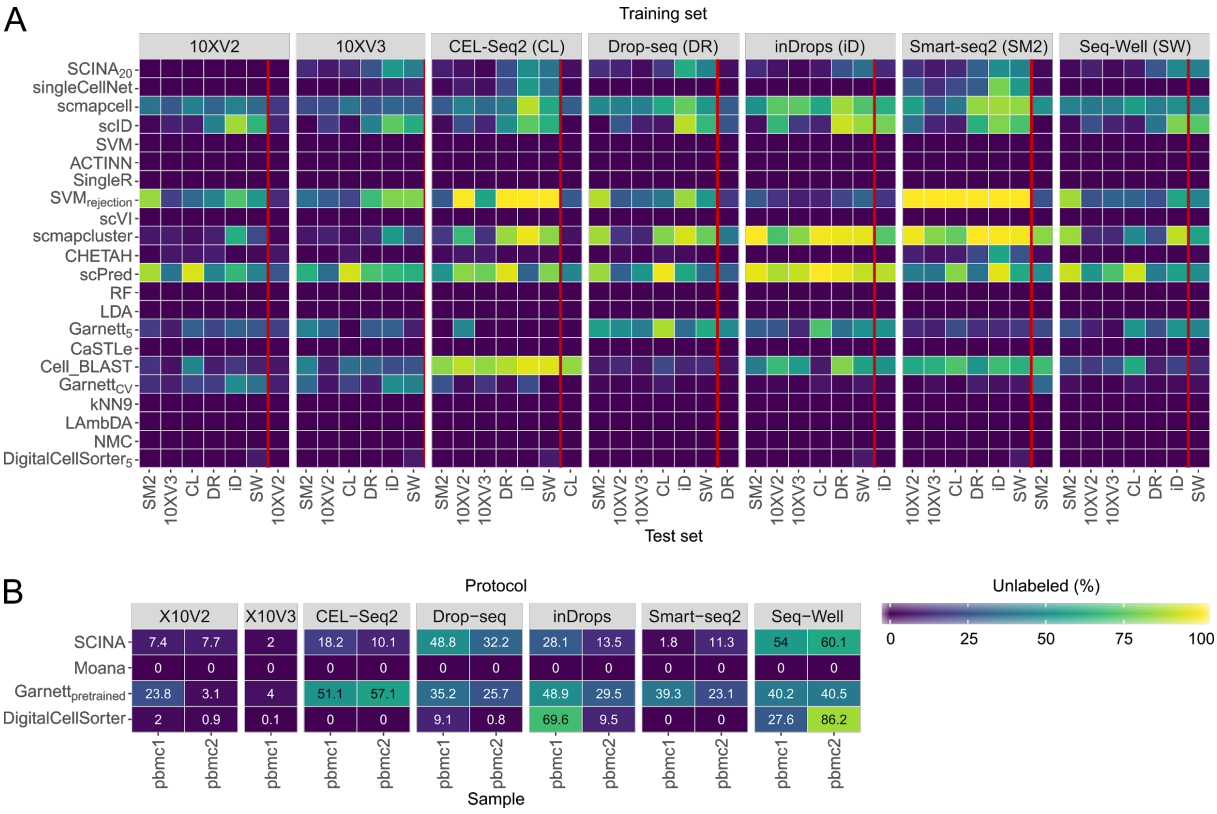


Figure S9. Percentage of unlabeled cells across the PbmcBench datasets. (A) Heatmap showing the median F1-score of the supervised classifiers for all train-test pairwise combination across different protocols. The training set is indicated in the grey box on top of the heatmap, the test set is indicated using the column labels below. Results showed to the left of the red line represent the comparison between different protocol using sample pbmc1. Results showed to the right of the red line represent the comparison between different samples using the same protocol, with pbmc1 used for training and pbmc2 used for testing. For *SCINA*,, *Garnett*$_{DE}$, and *DigitalCellSorter*$_{DE}$ different numbers of marker-genes were tested. Only the best result is shown here. **(B)** Percentage of unlabeled of the prior-knowledge classifiers on both samples of the different protocols. The protocol is indicated in the grey box on top of the heatmap, the sample is indicated with the labels below. Classifiers in the heatmaps are ordered based on their mean performance in Figure 3.

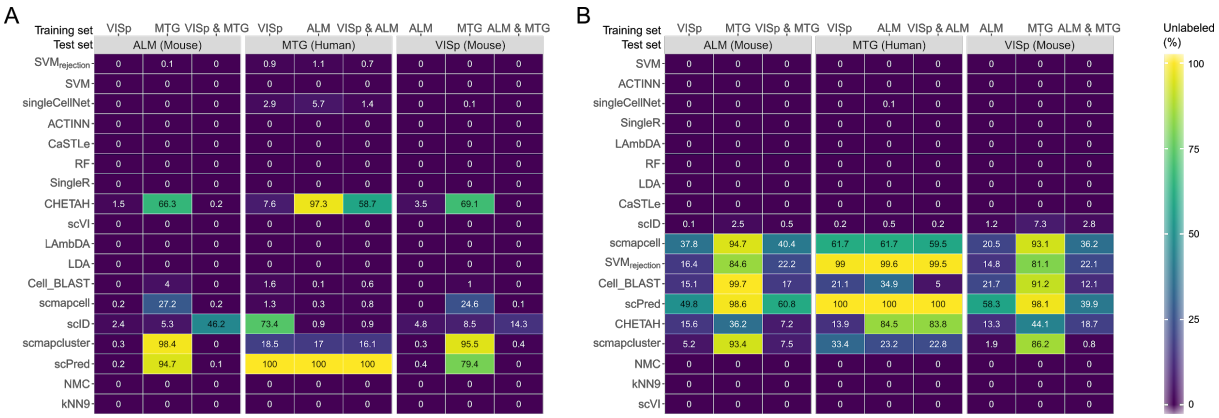


Figure S10. Percentage of unlabeled across brain datasets. Heatmaps show the percentage of unlabeled of the classifiers on **(A)** major lineage annotation with three cell populations, and **(B)** deeper level of annotation with 34 cell populations. The training set(s) are indicated using the column labels on top of the heatmap. The test set is indicated in the grey box. In each heatmap the classifiers are ordered based on their mean performance in Figure 4.

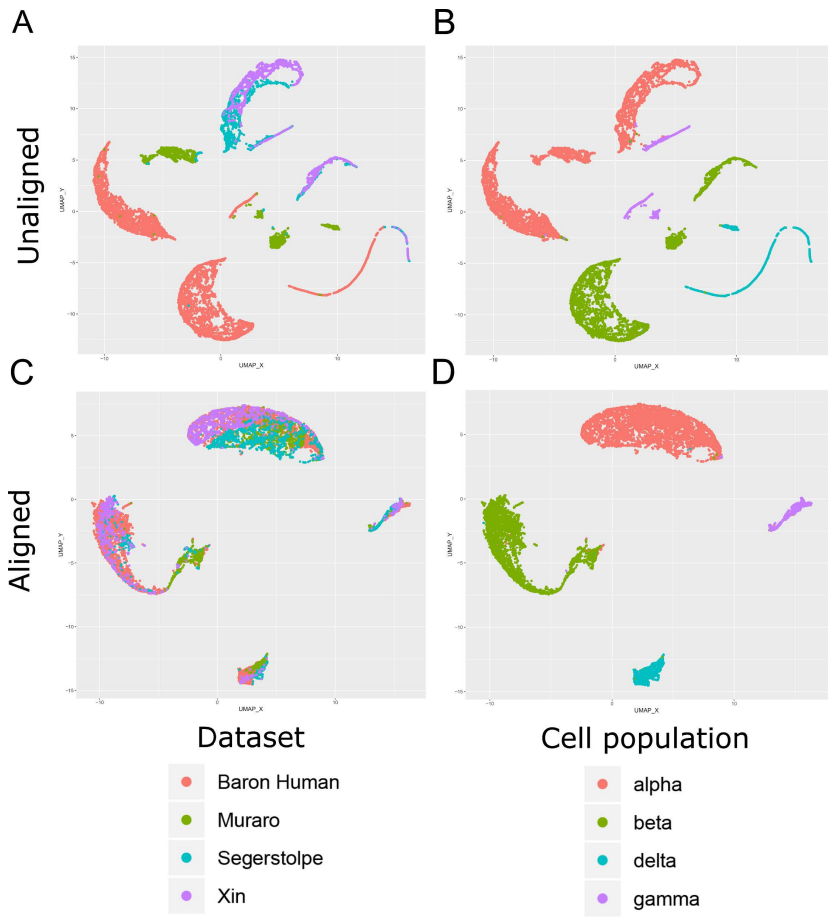


Figure S11. UMAP plots of the four pancreatic datasets used in the inter-dataset experiment. (A-B) UMAP plots before and **(C-D)** after alignment using MNN. In **(A, C)** the cells are colored by dataset and in **(B, D)** the cells are colored by cell population.

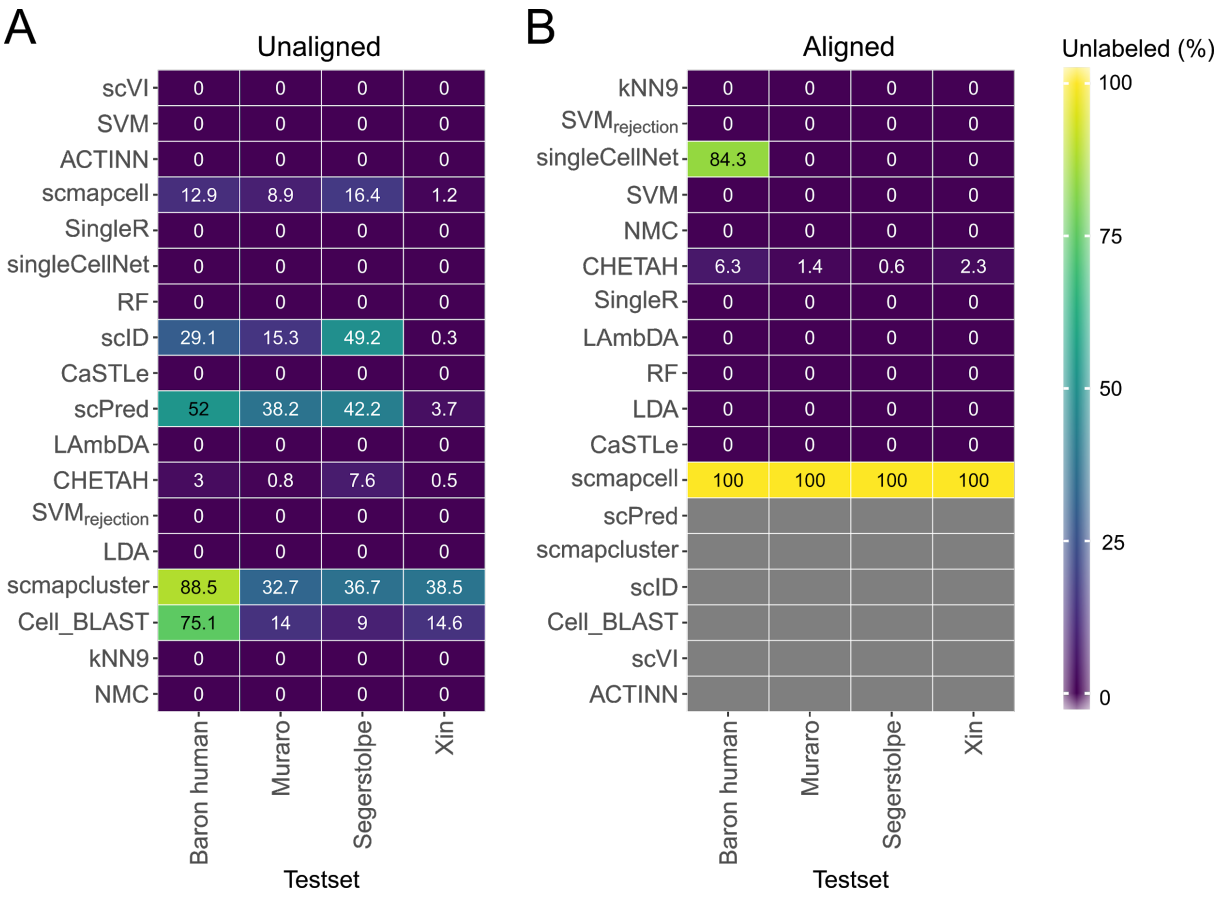


Figure S12. Percentage of unlabeled cells across different pancreatic datasets. Heatmaps showing the percentage of unlabeled for each classifier for the **(A)** unaligned and **(B)** aligned datasets. The column labels indicate which of the four datasets was used as a test set, in which case the other three sets were used as training data. Grey boxes indicate that the corresponding method could not be tested on the corresponding dataset. In each heatmap, the classifiers are ordered based on their mean performance in Figure 5.

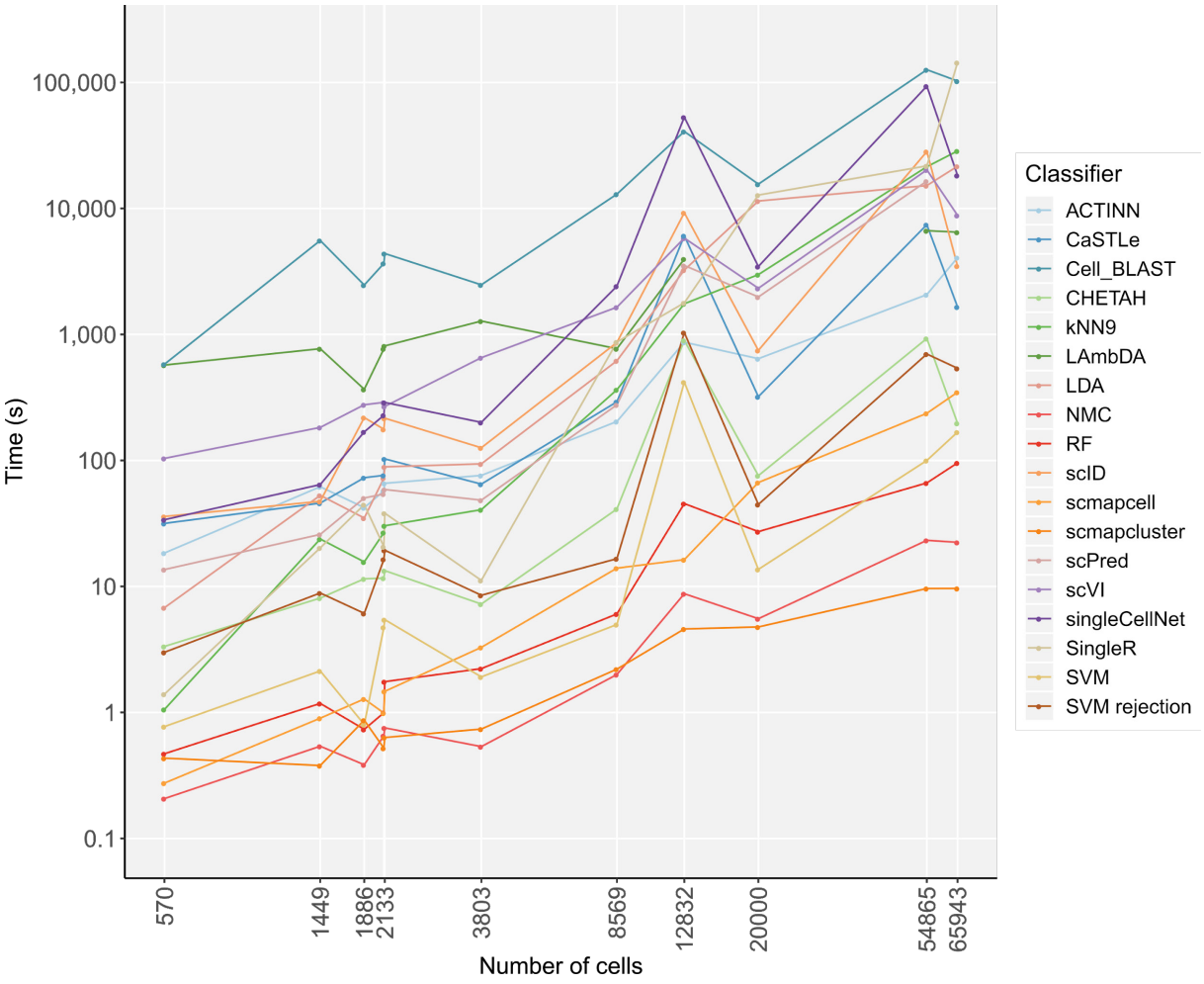


Figure S13. Computation time across different datasets. Line plots showing the computation time of the classifiers with the number of cells in all datasets. Classifiers are indicated using different colors. Both axes in the plot are log-scaled.

B

Cell identification in single-cell RNA-sequencing datasets using hierarchical progressive learning

Cell identification in single-cell RNA-sequencing datasets using hierarchical progressive learning

Lieke Michielsen (4282361)

January 18, 2020

Abstract

Since the revolution of single-cell RNA-sequencing, the number of available datasets has increased enormously. In these datasets, cell identification is mainly done manually, which is subjective and time-consuming. As a consequence, most datasets are annotated at a different resolution. This is not surprising as cell types form a hierarchy, but it can be problematic for downstream analysis or comparison of datasets. Several supervised methods have already been developed to overcome the drawbacks of unsupervised learning. None of these, however, combines the information found in multiple datasets and preserves the definition of cell populations in each dataset, while this consistency is necessary for downstream analysis. Furthermore, a supervised classifier should be able to detect new cell populations in an unlabeled dataset. Here, we introduce a hierarchical progressive learning pipeline with a one-class classifier to face these challenges. Using this pipeline, it is possible to construct a hierarchical classification tree by combining the information of multiple datasets. If datasets are annotated at a different resolution, their cell populations will be at different levels in the tree and all definitions are thus preserved. By using a one-class classifier for each cell population, it is also possible to have a correctly working rejection option and discover new cell populations. In this paper, we show that it is possible to construct a classification tree for simulated data and immune cells. When comparing the pipeline with a one-class to a linear classifier, we show that a one-class classifier indeed improves the rejection option. Using a linear classifier, on the other hand, results in a higher accuracy. Choosing between a one-class and a linear classifier is a trade-off between the ability of discovering new cell populations and a higher accuracy.

1 Introduction

In the past, cells could only be studied using a microscope. Based on morphological features, such as shape or size, cells were divided into different groups, so called cell types. As only a few features could be measured, classification into cell types was quite straightforward, but a precise definition was never formulated. Development of new measurement techniques, including single-cell RNA-sequencing (scRNA-seq), led to an increase in the number of features that could be measured in a cell. Using scRNA-seq the whole transcriptome, the products of more than 10,000 genes, can be sequenced in a single-cell [1]. That the missing definition of a cell type is becoming apparent now is illustrated with the following example. In 2016 and 2018, Tasic et al. sequenced 1,679 and 23,288 brain cells respectively [2, 3]. In the first study, they revealed 49 cell types, while more extensive sampling lead to the discovery of 133 cell types. Most of the cell types found in 2016 were split into subpopulations that could be seen now, but interestingly, also some cell types were concatenated into one big cell type containing high variance.

Currently, already more than 500 scRNA-seq datasets exist [4]. Due to improving technologies, this number and also the number of cells sequenced per dataset keeps increasing. At the moment, the biggest dataset, the mouse organogenesis cell atlas, consists of approximately two million cells [5]. An ongoing project, the human cell atlas, is even trying to sequence all cells in the human body [6]. These atlases can teach us a lot about, among others, the development of cells, relationships between them, and cellular dysregulation in human diseases.

In scRNA-seq datasets, cells are primarily annotated using clustering and visualisation techniques. Cells are first clustered into populations and these are identified based on the expression of marker genes. This, however, is time-consuming and subjective [7]. The number of cell populations identified in a dataset, for example, is strongly correlated with the number of cells analyzed [4]. Such disagreements between clusterings might be difficult during cohort studies. Van Der Wijst et al., for instance, identified expression quantitative trait loci (eQTLs), genomic loci explaining variation in expression levels of mRNAs, per cell type [8]. In order to compare these results to different studies, the cells should be annotated consistently such that the cell types are preserved across studies.

Inconsistency across datasets can occur when they are annotated at a different resolution. This is also illustrated by the two brain datasets sequenced by Tasic et al. [2, 3]. In the first dataset, less cells were sequenced and less cell populations were identified compared to the second dataset. Cell populations of the first dataset were mainly split into multiple populations in the second dataset. These populations in the second dataset could be seen as subpopulations of cell types in the first dataset, which indicates a hierarchy of cell populations in the brain.

Immune cells are another well-known example of a hierarchy and as a consequence most datasets are annotated at a different level. T-cells for example can be subdivided into CD4+ T-cells and CD8+ T-cells, which are for example found in the PbmcBench datasets [9]. In the FACS-sorted PBMC dataset, however, subtypes of CD4+ T-cells are annotated [10].

As unsupervised learning is thus subjective and time-consuming, supervised learning could be a solution. Using a hierarchical classifier to profit from this hierarchy of cell populations could have several advantages over flat classification. When using a flat classification approach a classifier needs to be able to distinguish a large number of classes, while if we exploit this hierarchy, the problem is divided into smaller sub-problems. Another advantage of hierarchical classification is that the cell type definition of multiple datasets can be combined, which gives consistency.

Currently, some classifiers, Garnett, CHETAH, and Moana, already exploit this hierarchy in the data [11, 12, 13]. Garnett and Moana both depend on prior knowledge. Marker genes have to be defined for each cell population. Especially for deeper annotated datasets, it can be difficult to define marker genes that replicate across scRNA-seq datasets [14, 15]. Benchmarking several supervised and prior knowledge classifiers also showed that adding prior knowledge is not

beneficial [16]. CHETAH, on the contrary, constructs the classification tree based on one dataset by hierarchically clustering the reference profiles of the cell types. CHETAH classifies cells based on the similarity of a new cell to the reference profile of that cell type. The performance of CHETAH, however, is mediocre compared to flat scRNA-seq classifiers [16]. A successful strategy to exploit this hierarchy is still missing.

As correspondence of cell types between current datasets is missing, it would be most straightforward to use a reference atlas containing all possible cell types to train a classifier. This atlas, however, does not exist yet and might never be fully complete. Aberrant cell types in particular might be missing as a huge number of diseases exist. Furthermore, mutations could always cause new cell types.

Ideally, the classifier should thus be able to combine the information of multiple datasets and continue learning. Each time a new cell population is found in a dataset, it should be added to the knowledge of the classifier. This could be done using progressive learning, a learning strategy inspired by humans. Human learning is a continuous process that never ends [17]. Using progressive learning, the task complexity is gradually increased, for instance, by adding more classes, but the knowledge of the previous classes is preserved [18, 19]. This way, we can combine the information of multiple existing datasets and retain the possibility to add more datasets afterwards. This strategy, however, cannot be simply applied to scRNA-seq datasets. A constant terminology to describe cell types is missing, so straightforward identification of new cell types based on the name is not possible.

Even if multiple datasets are combined, there might be unseen populations in a new unlabeled dataset. It is important that a classifier can identify these instead of mislabeling them as another known population. This could for example be done using a rejection option. Previously, this was mainly implemented by setting a threshold on the posterior probability [11, 16, 20, 21]. If the highest posterior probability did not exceed a threshold, a cell was rejected. By looking at the posterior, the actual similarity between a cell and the cell type is ignored.

Recently, many classifiers have been developed to identify cell types in scRNA-seq datasets, but none of them solves these problems of the ever-growing datasets and the fact that we might never have a complete reference atlas. Moreover, the classifiers cannot handle the inconsistency across datasets and struggle with identifying new populations. Additionally, current classifiers cannot successfully exploit the hierarchical relationship between classes and do not preserve the cell types if they are retrained which is problematic for downstream analysis.

Here, we will use hierarchical progressive learning with a one-class classifier to face these challenges (Figure 1). During the training phase, we will exploit the different resolutions of multiple datasets (Figure 1a). Our pipeline is slightly different than the normal progressive learning pipeline since we cannot simply add new cell populations to the classifier. We will discover which cell populations are new in a dataset by reciprocal matching the labels of multiple datasets using the following steps. For all datasets a flat classification tree is constructed (Figure 1b). Using this tree and the corresponding dataset, a classifier is trained for each node in the tree, except for the root (Figure 1c). These classifiers are used to predict the labels of another dataset (Figure 1d). We use reciprocal matching, so the labels of dataset 1 are predicted using the classifier trained on dataset 2 and vice versa. By comparing the cluster labels with the predicted labels, matches between the populations of two datasets can be found. These matches are used to update the classification tree (Figure 1e). In dataset 2, for example, subpopulations of 'Class 1' of dataset 1 are found. Therefore, these classes, 'Class A' and 'Class B', are added as children to 'Class 1' (Figure 1e). By constructing this hierarchy instead of removing 'Class 1', we preserve their old definition. This might have been used in downstream analysis of the first dataset, so the updated classifier should be able to reconstruct this. The updated tree and both datasets are used to retrain the classifier. If there is a third dataset, we can again match the labels and update the tree. This continues until there are no more labeled datasets.

For each node in the tree, a separate classifier is trained. By using a one-class classifier, we can overcome the problem of the rejection option. When training a one-class classifier only examples of the positive class are used and the classifier tries to fit a tight decision boundary around these (Figure 2a) [22]. Figure 2 illustrates the improved rejection option of a one-class classifier compared to a linear one-vs-all classifier. When training a one-class and a linear one-vs-all classifier on three classes and test them on a new unseen class, the linear classifier does not reject any of the cells, but the one-class classifier rejects all new cells correctly.

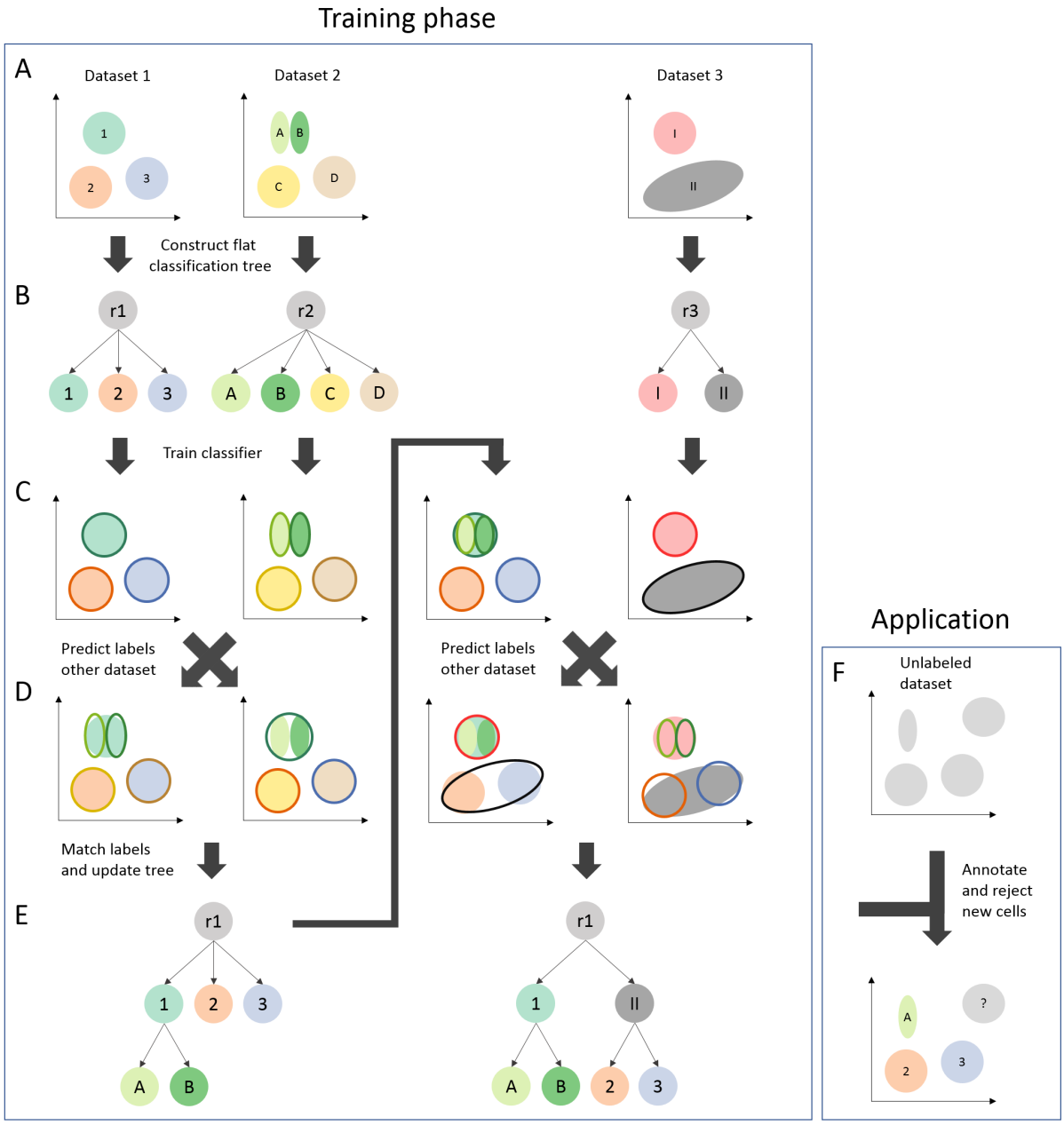


Figure 1: Schematic overview of the hierarchical progressive learning pipeline. (a) Multiple datasets are combined. These datasets can be annotated at a different resolution. (b) A flat classification tree is constructed for each dataset. (c) This classification tree is used to train a classifier on the dataset. (d) The trained classifier is used to predict the labels of another dataset. (e) The predicted labels are compared to the cluster labels to found matching labels. The classification tree of the first dataset is updated based on these matches. If there is a third dataset, this updated tree can be used to predict these labels and vice versa to update the tree further. (f) If there are no more labeled datasets to add, the final tree can be used to predict the labels of an unlabeled dataset. If there is an unseen cell population, this population is rejected instead of annotated.

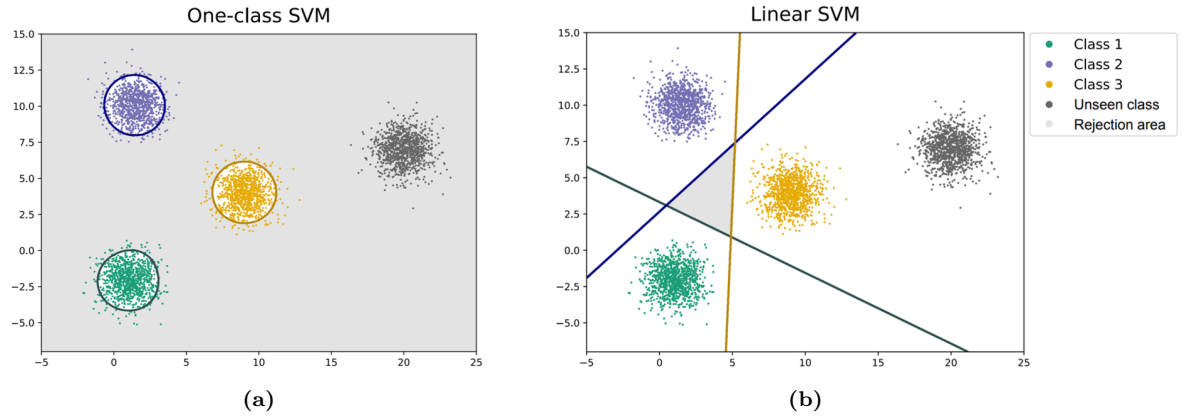


Figure 2: Visualized rejection option for a (a) one-class and (b) one-vs-all linear support vector machine (SVM). Three Gaussian distributions, 'Class 1', 'Class 2', and 'Class 3', were simulated as training data. The decision boundaries of the one-class SVM and one-vs-all linear SVM are visualized for each class. The rejection area is visualized in light-grey. Next, a fourth Gaussian distribution, a new unseen class, was simulated as test data. Using the one-class SVM this new class is correctly rejected, while using the linear SVM all test points are predicted to be 'Class 3'.

After the training phase, the classification tree can be used to annotate a new unlabeled dataset (Figure 1f). If there are new unseen populations in this dataset, the classifier can reject these cells. We have trained a one-class classifier for each node in the tree, so if a new cell falls outside all decision boundaries, it is rejected.

In this paper, we use simulated, peripheral blood mononuclear cells (PBMC), and brain data, to test our hierarchical progressive learning pipeline. First, we will evaluate the classification performance on predefined hierarchies and compare this performance to existing methods. Next, we will verify whether we can construct these classification trees using progressive learning. Finally, we evaluated the rejection option to test the ability of discovering new cell populations.

2 Methods

2.1 Hierarchical progressive learning

In the following sections we will explain the hierarchical progressive learning pipeline in more detail. We will focus on how we train the hierarchical classifier, predict labels, match the labels of two datasets, and update the tree.

2.1.1 Training the hierarchical classifier

Training the hierarchical classifier is the same for every tree. This tree could be a flat classification tree as in Figure 1b, but also a predefined tree. We train a local classifier for each node in the tree except for the root node. This local classifier is either a one-class or a binary classifier. We use a recursive algorithm to train these local classifiers and start with training the classifier for one of the leaf nodes. When training the classifier, the positive training samples include samples from the node itself and samples from its child nodes. Negative training samples are selected using the siblings policy [23]. Sibling nodes include all nodes that have the same ancestor, excluding the ancestor itself. Similar to the positive samples, the negative samples also include the samples of the children of siblings (Figure S1).

One-class classifier

We chose to use a one-class support vector machine (SVM) as one-class classifier. To train this classifier, only the positive samples were needed. We used the `svm.OneClassSVM(nu = 0.05)`

function from the scikit-learn library in Python [24]. During prediction, the one-class SVM gives each sample a score. If this score is higher than the offset, a sample is labeled positive. During the experiments, four different offsets were tried to determine the optimal value. First, we tried the default offset, which is the offset determined by the scikit-learn function based on the `nu` parameter. Here, `nu` is a lower bound on the fraction of support vectors and an upper bound on the fraction of training errors. Figure 2 shows that using this default offset, the decision boundary is quite tight around the training samples. By decreasing the offset, we try to make this boundary less tight and reduce the training error. The other offsets were determined based on the scores of the positive training samples. The minimum offset is the minimum score of the training samples. We also tested the 1^{st} percentile (perc.) and the 0.5^{th} perc. offset. These are the 1^{st} perc. and 0.5^{th} perc. of the scores of the positive samples respectively. As the offset is determined based on the scores of the positive samples, it is different for every node in the tree. These offsets were also saved and could later on be used during prediction.

Binary classifier

Benchmarking existing classifiers showed that $\text{SVM}_{\text{rejection}}$ was the best classifier for scRNA-seq data at the moment [16]. This classifier is a multiclass, linear SVM using the `svm.LinearSVC()` function from the scikit-learn library in Python [24]. Therefore, we decided to also use this linear SVM, but than in a binary setting, here. $\text{SVM}_{\text{rejection}}$ has a rejection option based on the posterior probability. We did not implement this for the linear SVM. We trained this classifier on the positive versus the negative samples. Cells can thus still be rejected by the linear SVM Figure 2.

Dimensionality reduction

In scRNA-seq experiments, more than 10,000 genes are sequenced. Dimensionality reduction is important as using all features might lead to overfitting the classifier. When using the linear SVM, L2-regularization was applied by default, so no extra dimensionality reduction techniques were used. For the one-class SVM this was not implemented and thus we reduced the dimensions ourselves. First, we did a principal component analysis (PCA) on the data to select the principal components (PCs) which explain 90% of the variance of the data. Next, we select informative PCs for each node separately, i.e. PCs where the positive and negative samples show a difference. These PCs are selected by doing a two-sided two-sample t-test with $\alpha = 0.05$. We also applied Bonferroni multiple testing correction, so all p-values were multiplied by the number of genes. In some cases, this correction was too strict and no PCs were selected. In these rare cases, the five PCs with the smallest p-values were selected.

2.1.2 Predicting the labels

When we want to predict the label of a new sample, we start at the root node, which we will call the parent node now. We use the local classifiers of its descendants to predict whether a sample belongs to one of them. All local classifiers predict the label of the cell and give a score to it. These scores represent the signed distance of a cell to the decision boundary. When using the linear SVM, we use the `predict()` function to predict whether the sample is positive or negative and the `decision_function()` function to obtain the score for a cell. For the one-class SVM, we use the `score_samples()` function. If this score is higher than the offset we determined in the training phase, the cell is predicted to be positive. When comparing the predictions and scores of the local classifiers, three different scenarios can happen:

1. All child nodes label the new sample negative. In this case, we label the sample with the name of the parent node. If this is the root node, the sample is rejected and remains unlabeled. Otherwise, the sample is thus labeled as an internal node.

2. One child node labels the sample positive. If this node is a leaf node, the sample is labeled as this node. If this node is an internal node, this node becomes the new parent and we start to look at its children.

3. Multiple child nodes label the sample positive. In this scenario, we compare the scores of the nodes and only consider the node with the highest score. Similar to scenario two; if this node is a leaf node, the sample is labeled as this node. If this node still has children, this node becomes the new parent and we start to look at its children.

2.1.3 Matching labels and updating the tree

As shown in Figure 1d, we used classifier 1 to predict the labels of dataset 2, $D2$, and classifier 2 to predict the labels of dataset 1, $D1$. To match the labels and update the classification tree, we construct confusion matrices, $C1$ and $C2$, for $D1$ and $D2$ respectively. Here, $C1_{ij}$ indicates how many cells of cell type i of $D1$ are predicted to be cell type j of $D2$. This prediction can be a leaf node, internal node or a rejection. We normalize each matrix such that the sum of each row is one, which gives $NC1$ and $NC2$ (Equation (1)).

$$NC1_{ij} = \frac{C1_{ij}}{\sum_{\forall j} C1_{ij}} \tag{1}$$

Next, we convert $NC1$ and $NC2$ to binary confusion matrices $BC1$ and $BC2$ to find matches between the cell types. Here, the high values in $NC1$ and $NC2$ are most interesting as these indicate that many cells of a cell type of $D1$ are predicted to be cell type of $D2$ or vice versa. We use the notations shown in Equations (2) and (3) to indicate (the index of) these values.

First, we want to find the best match for each cell type. Remember that the true cell types were represented by the rows and the predictions by the columns. For each row, the index of the highest value of $NC1_i$ and $NC2_i$ will be set to 1 in $BC1$ and $BC2$ respectively (line 3, Algorithm 1). This column namely indicates the best match to this cell type i.

Next, we look if the cell type has more matches (lines 5-7). If the difference between the highest value and the second highest value of a row is smaller than a threshold, and if this second highest value is also higher than 0.1, it is also set to 1. If so, the third highest value is compared with the second highest value etc. The default value of the threshold is 0.25.

$$\max(a, b) : b^{th} \text{ highest value of } a \tag{2}$$

$$\arg\max(a, b) : \text{index of } b^{th} \text{ highest value of } a \tag{3}$$

Algorithm 1 Pseudo code to fill a binary confusion matrix BC.

```
1: function FillBinary(matrix NC, threshold t)
2:     matrix BC ← 0
3:     BC_{i,argmax(NC_i,1)} ← 1
4:     counter x ← 2
5:     while max(NC_i, x) − max(NC_i, x − 1) < t and max(NC_i, x) > 0.1 do
6:         BC_{i,argmax(NC_i,x)} ← 1
7:         x + +
8:     end while
9:     return BC
```

Next, we combine $BC1$ and $BC2$ into the added matrix X (Equation (4)). The columns in the X represent the cell populations of $D1$, while the rows in the added matrix represent the cell populations of $D2$. If $X_{ij} = 2$, this indicates a reciprocal match between cell population i from $D2$ and j from $D1$. If $X_{ij} = 1$, this is indicates a one-sided match. A 0 indicates that there is no match between the two cell populations. We will use X to find matches between the labels and update the tree according to this. When we update the tree, we will always update the tree belonging to $D1$ (Figure 1e).

$$X = B1' + B2 \tag{4}$$

When matching the labels, many different scenarios can happen. Here, we will explain the four most common scenarios: a perfect match, splitting nodes, merging nodes, and impossible matching. All other scenarios are explained in Supplementary Note 1.

Perfect match

In this scenario, we find a unique match between a node from $D1$ and a node from $D2$ (Figure 3). In the corresponding row and column, no other matches are found. We can identify this scenario as there is only one non-zero value in the corresponding row and colunn in X (Figure 3b). This indicates a perfect match, so the tree is not updated. The labels of $D2$ have to be updated. If, for instance, 'Class 1' of $D1$ matches 'Class A' from $D2$ perfectly, 'Class A' will not be in the updated tree as 'Class 1' is already in there. To ensure that the 'Class A' samples of $D2$ are also used during the next training phase, we rename them to 'Class 1'.

Splitting nodes

In this scenario, a node from $D1$ matches multiple nodes from $D2$ (Figure 4). We can easily identify this as there are multiple non-zeros in one column in X (Figure 4b). We consider the nodes from $D2$ subpopulations of the node from $D1$, so we add these subpopulations as children to the corresponding node from $D1$.

Merging nodes

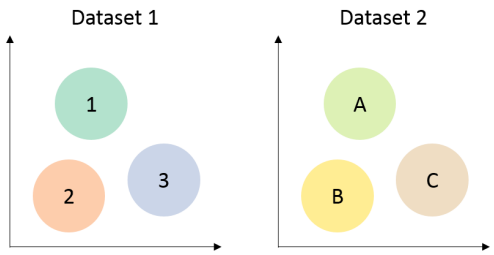
This scenario is quite similar to multiple rows scenario. Here, however, multiples nodes from $D1$ match one node of $D2$ (Figure 5). There are thus multiple non-zeros in one row in X (Figure 5b). We consider the nodes from $D1$ subpopulations of the node from $D2$, so we need to merge these subpopulations into the node from $D2$.

Impossible matching

Sometimes, it could be impossible to match the labels from two datasets (Figure 6). This happens if there are multiple non-zero values in the corresponding row and the corresponding column (Figure 6b). Something could have gone wrong during the clustering, e.g. 'Class 1' and 'Class 2' from $D1$ match 'Class A' from $D2$, but 'Class 2' also matches 'Class C' from $D2$. Here, 'Class A' and 'Class C' should be merged into 'Class 2', but 'Class A' should also be split into 'Class 1' and 'Class 2'. 'Class 2', however, cannot be added to the tree twice.

We will iterate over X to find the different scenarios and update the tree. If we encounter a non-zero value, we check whether there are other non-zero values in the corresponding row and column to identify the scenario. Each time that the tree is updated, the corresponding values are removed from X. If the matching is impossible, the corresponding values are thus not removed. Once we have iterated over the whole matrix and if there are still values in X, we will change this matrix into a strict matrix. This means that we will only consider reciprocal

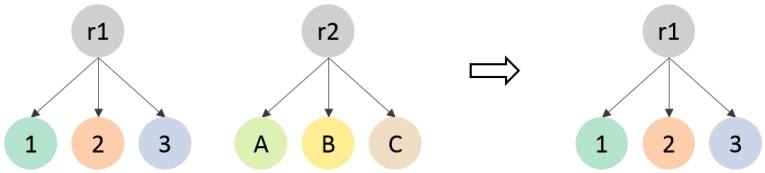
matches now, so all '1's are turned into a '0'. We again iterate over the matrix to see if we can solve these impossible situations now.



(a)

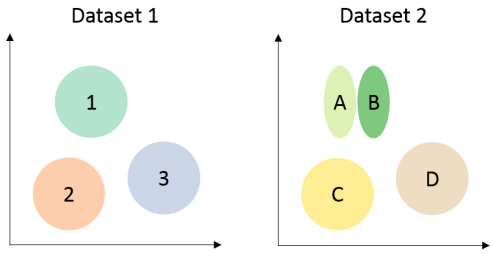
	1	2	3	root1
A	**2**	0	0	0
B	0	**2**	0	0
C	0	0	**2**	0
root2	0	0	0	0

(b)



(c)

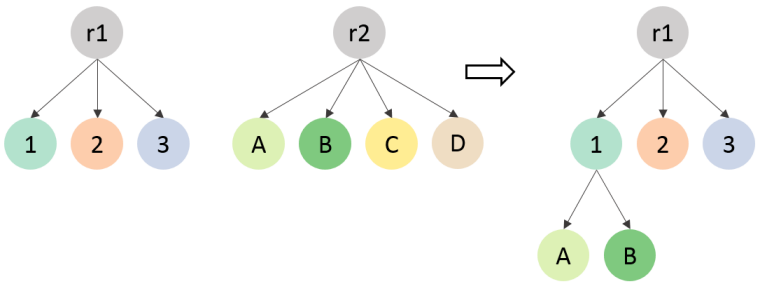
Figure 3: Example of two datasets that have a perfect match. (a) Schematic drawing of the two datasets. (b) Matrix X beloning to these datasets. (c) Classification trees belonging to $D1$ and $D2$ and the updated tree.



(a)

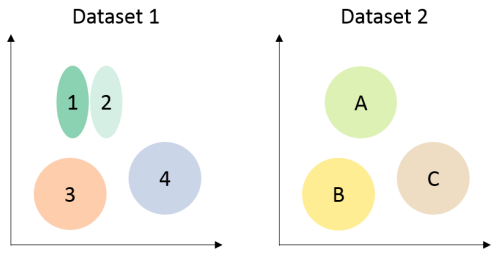
	1	2	3	root1
A	**2**	0	0	0
B	**2**	0	0	0
C	0	**2**	0	0
D	0	0	**2**	0
root2	0	0	0	0

(b)



(c)

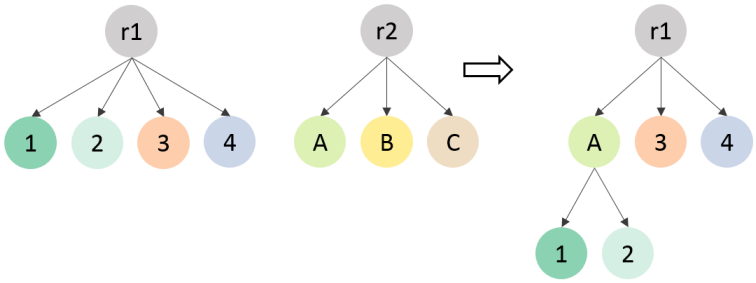
Figure 4: Example of a split scenario. (a) Schematic drawing of the two datasets. (b) Matrix X beloning to these datasets. Here, we see that 'Class A' matches both 'Class 1' and 'Class 2'. The other cell populations, 'Class 3' with 'Class B' and 'Class 4' with 'Class C', have a perfect match. (c) Classification trees belonging to $D1$ and $D2$ and the updated tree.



(a)

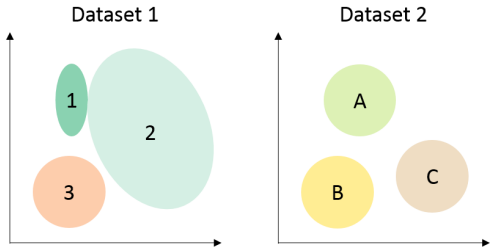
	1	2	3	4	root1
A	**2**	**2**	0	0	0
B	0	0	**2**	0	0
C	0	0	0	**2**	0
root2	0	0	0	0	0

(b)



(c)

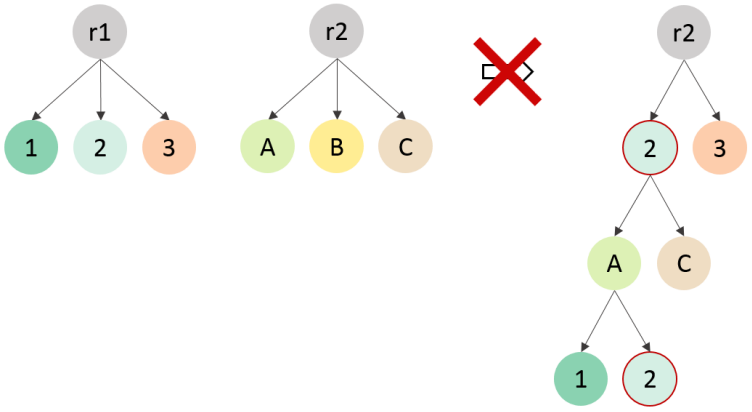
Figure 5: Example of a merging scenario. (a) Schematic drawing of the two datasets. (b) Matrix X beloning to these datasets. Here, we see that 'Class 1' matches both 'Class A' and 'Class B'. The other cell populations, 'Class 2' with 'Class C' and 'Class 3' with 'Class D', have a perfect match. (c) Classification trees belonging to $D1$ and $D2$ and the updated tree



(a)

	1	2	3	root1
A	**2**	**2**	0	0
B	0	0	**2**	0
C	0	**2**	0	0
root2	0	0	0	0

(b)



(c)

Figure 6: Example of two datasets that are impossible to match. (a) Schematic drawing of the two datasets. (b) Matrix X beloning to these datasets. (c) Classification trees belonging to $D1$ and $D2$ and the updated tree.

2.2 Classification performance evaluation

2.2.1 Hierarchical F1-score

We use the hierarchical F1-score (HF1-score) to evaluate the performance of the classifiers [25]. For each cell i in the test set, the HF1-score compares the predicted set P_i with the true set T_i. P_i contains the predicted cell type at the highest resolution and all the ancestors and T_i contains the true cell type at the highest resolution and all the ancestors. $|P_i \cup T_i|$ is the overlap between these two sets. These sets are used to calculate the hierarchical precision (hP) and hierarchical recall (hR). The HF1-score is the harmonic mean of those (Equation (5)).

If, for example, a correct leaf node is predicted for a cell, both hP and hR will be 1. If, however, a correct internal node is predicted, while the true label is a leaf node, hP will still be 1, but hR will be lower.

$$HF1 = \frac{2 * hP * hR}{hP + hR}, hP = \frac{\sum_i |P_i \cup T_i|}{\sum_i |P_i|}, hR = \frac{\sum_i |P_i \cup T_i|}{\sum_i |T_i|} \tag{5}$$

2.2.2 Median F1-score

We use the median F1-score to compare the classification performance to flat classifiers. The F1-score is calculated for each cell population in the dataset and afterwards the median of these scores is taken. Rejected and internal labeled cells are not considered when calculating the median F1-score.

2.3 Classification tree evaluation

We evaluated the constructed tree by constructing multiple trees and counting how often each link occurred. These trees were drawn and compared to the ground truth.

2.4 Datasets

2.4.1 Simulated data

We used the R-package Splatter version 1.6.1 to simulate a hierachical scRNA-seq dataset that represents the tree shown in Figure 7a [26]. As Splatter is originally not developed to simulate hierarchical data, we create this dataset by concatenating three separately simulated datasets. These three datasets all consist of 9000 cells and 3000 genes. We use the `group.prob` option to simulate three different groups, 'Group12', 'Group3', and 'Group456' with ratios 0.334, 0.167, and 0.499 in the first dataset. In second dataset, four groups, 'Group1', 'Group2', 'Group3', and 'Group4' all have a probability of 0.167, while 'Group56' has a probability of 0.332. In the third dataset all groups, 'Group1', 'Group2', 'Group3', 'Group4', 'Group5', and 'Group6' have an equal probability.

We stack these three datasets columnwise (Figure S2). This thus results in 9000 genes per cell. When stacking the datasets, we have to ensure that the labels of the three datasets match. A cell from 'Group6' of dataset 3 can, for instance, only be combined with a cell of 'Group56' of dataset 2 and 'Group456' of dataset 1. Based on the probabilities defined above, we would expect 4491 cells of 'Group456' in the first dataset and 1503 cells of 'Group4' and 2988 cells of 'Group56' in the second dataset. Based on these numbers, part of the 'Group456' cells could match 'Group4' and part could match 'Group56'. As this are probabilities, however, the exact number of cells is slightly different. Because of this, some cells, 161 in total, could not be matched and are thus removed. The final dataset consists of 8839 cells and 9000 genes. Each cell in the final stacked dataset gets the label from the third dataset. As a preprocessing step, we did a log-transformation on the final dataset, $log_2(x + 1)$. A UMAP of this final simulated dataset shows it indeed represents the desired hierarchy (Figure 7b).

We did a 10 fold cross-validation to test the the classification performance. The classifiers were trained on the tree shown in Figure 7a.

To test the tree construction on this dataset, we split the simulated dataset into a train and test set using a 10 fold cross-validation. Next, we randomly split this training set into three different batches, Batch 1, Batch 2, and Batch 3, to simulate the idea of different datasets. We changed the labels of some groups in these batches to get different resolutions (Table 1). When constructing the trees, we try all different orders of the batches. We can for example start with Batch 1, add Batch 2 and finally add Batch 3, but also vice versa. In total there are six different possibilities. Combining this with the 10 fold cross-validation, there are 60 trees constructed in total.

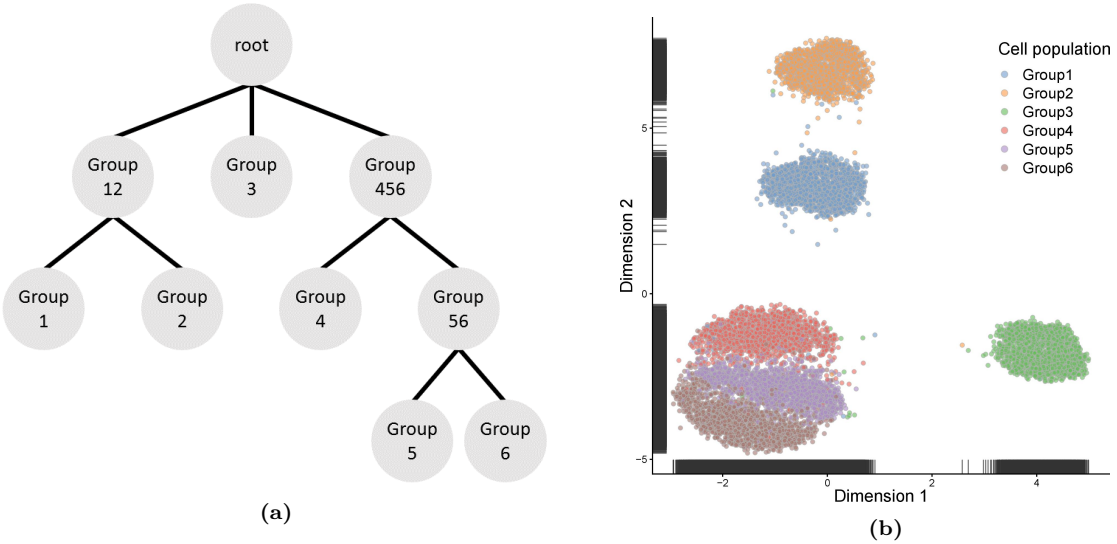


Figure 7: (a) Hierarchical tree and (b) UMAP of the simulated data

Table 1: Labels of the simulated dataset when testing tree construction.

Original cell population	Label Batch 1	Label Batch 2	Label Batch 3
Group1	Group12	Group1	Group1
Group2	Group12	Group2	Group2
Group3	Group3	Group3	Group3
Group4	Group456	Group4	Group4
Group5	Group456	Group56	Group5
Group6	Group456	Group56	Group6

2.4.2 PBMC data

We used the downsampled FACS-sorted PBMC dataset constructed in [16]. The cells were first FACS-sorted into ten different cell populations (CD14+ monocytes, CD19+ B cells, CD34+ cells, CD4+ helper T-cells, CD4+/CD25+ regulatory T-cells, CD4+/CD45RA+/ CD25− naive

T-cells, CD4+/CD45RO+ memory T-cells, CD56+ natural killer cells, CD8+ cytotoxic T-cells, CD8+/ CD45RA+ naive cytotoxic T-cells) and sequenced using 10X chromium [10]. Afterwards, the dataset was preprocessed. First, genes with zero counts across all cells were removed. Next, the median number of detected genes per cell was calculated for each cell. Cells were filtered out if the number of detected genes was 3 median absolute deviations (MAD) in log scale lower than the median number of detected genes per cells. Furthermore, we did a log-transformation on the dataset, $log_2(x+1)$. Finally, each cell population was downsampled to 2,000 cells. This resulted in a total dataset of 20,000 cells and 21,952 genes.

We did a 10 fold cross-validation to test the the classification performance. The classifiers were trained on a classification tree based on hematopoietic differentation (Figure S3). During the cross-validation the original labels were used. Some of these labels, CD4+ T-cells and CD8+ T-cells, are thus internal nodes instead of leaf nodes.

To test the tree construction on this dataset, we split the dataset into a train and test set using 10 fold cross-validation. Next, we randomly split this training set into three different batches, to simulate the idea of different datasets. We renamed, merged, or removed some cell populations in the batches to ensure that each batch is annotated at a different resolution (Table 2). When constructing the trees, we try all different ordering of the batches. This gives six possibilities and we have 10 fold cross-validation, so in total 60 trees are constructed.

Table 2: Labels of the PBMC dataset when testing the tree construction.

Original cell population	Label Batch 1	Label Batch 2	Label Batch 3
CD14+ monocytes	CD14+ monocytes	CD14+ monocytes	CD14+ monocytes
CD19+ B cells	CD19+ B cells	CD19+ B cells	CD19+ B cells
CD34+ cells	CD34+ cells	CD34+ cells	CD34+ cells
CD56+ natural killer cells	CD56+ natural killer cells	CD56+ natural killer cells	CD56+ natural killer cells
CD4+ helper T-cells	T-cells	CD4+ T-cells	-
CD4+/CD25+ regulatory T-cells	T-cells	-	CD4+/CD25+ regulatory T-cells
CD4+/CD45RA+/ CD25- naïve T-cells	T-cells	-	CD4+/CD45RA+/ CD25- naïve T-cells
CD4+/CD45RO+ memory T-cells	T-cells	-	CD4+/CD45RO+ memory T-cells
CD8+ cytotoxic T-cells	T-cells	CD8+ T-cells	-
CD8+/CD45RA+ naïve cytotoxic T-cells	T-cells	-	CD8+/CD45RA+ naïve cytotoxic T-cells

2.4.3 Allen Mouse Brain data

We used the Allen Mouse Brain (AMB) data to look at different resolutions of cell types in the brain [3]. This dataset consists of 12,771 cells and 42,625 genes sequenced using SMART-Seq V4. We used the filtered dataset created in [16]. This dataset was preprocessed in the same way as the PBMC dataset. After this preprocessing, cell populations smaller than 10 cells were removed.

Cells in this dataset were annotated at three different levels. When testing the classification performance on a predefined tree, we used these three annotation levels to construct the tree. We build the tree by attaching the unique labels of the first level (GABAergic, glutamatergic, and non-neuronal cells) to the root. Next we attached the fifteen cell populations from the second annotation to their corresponding parent node. Finally we attached the cell populations from the finest resolution to their corresponding parent node. We also removed the redundant nodes. The 'Astro Aqp4' leaf node, for instance, was the only node attached to the 'Astro' population, which was the only node attached to the 'Non-Neuronal' population. Therefore, we removed the 'Non-Neuronal' and 'Astro' nodes and connected 'Astro Aqp4' to the root directly. The final tree is shown in Figure S4. The tree is thus constructed using all three levels on annotations. When training the classifier, we use this tree and only the deepest annotation level. Here, we did a five fold cross-validation when testing the classification performance. These folds were the same as used in [16].

2.5 Comparison to other classifiers

We compared the results of our classifiers on the AMB dataset to two classifiers that were benchmarked on this dataset, $\text{SVM}_{\text{rejection}}$ and CHETAH [16]. For the comparison we used the results as generated in [16].

2.6 Implementation

The hierarchical classification pipeline is implemented in Python 3.7. R was used to simulate the data.

3 Results

When evaluating the hierarchical progressive learning pipeline, there are three different requirements that should be fulfilled: a high classification performance, a correctly constructed classification tree, and a working rejection option. By testing the classification performance, we can verify whether new cells are labeled correctly and if using a hierarchical over a flat classifier indeed improves the performance. Besides this correct labeling, it is also important that our progressive learning pipeline works, so that indeed the correct tree is constructed. Finally, we test the ability to detect unseen cell populations.

3.1 Classification performance

3.1.1 Decreasing the offset of the one-class SVM improves classification performance on simulated data

We first tested the classification performance of the classifiers on a simulated dataset since this is an easy hierarchical dataset. The classifiers were trained on the tree shown in Figure 7a. The one-class SVM was implemented with different offsets to investigate the effect of this parameter. Decreasing the offset of the one-class SVM leads to an increase in performance (Figure 8). In the simulated dataset, the different cell populations are widely separated (Figure 7b). This is also supported by comparing the scores of the positive and negative samples for each cell population.

The distributions of the scores of the positive and negative samples show no overlap, which makes it relatively easy to distinguish them (Figure S5). If we compare the confusion matrices of the one-class SVM with minimum and default offset, we also notice that the only difference is the number of rejected and internal node annotations (Tables S1 and S2). The one-class SVM with default offset rejects more cells or labels them as an internal node instead of a leaf node compared to the minimum offset (6.4% vs. 0.1%). Both classifiers have no wrong predictions.

If we compare the performance of the one-class with the linear SVM, we see that the one-class SVM with minimum offset and the linear SVM both have a median HF1-score of 1. Looking at the rejected and internal node annotations, we notice a small difference in the performance though. The one-class SVM has one rejected and three internal node annotations, while the linear SVM classifies everything correct.

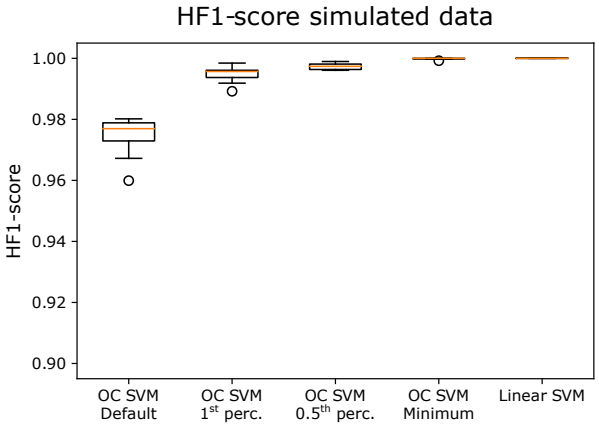


Figure 8: Boxplots showing the HF1-score of the one-class (OC) with different offsets and the linear SVM on the simulated data.

3.1.2 Cell populations in real data overlap, so decreasing the offset does not improve the performance

As the simulated dataset is a relatively easy dataset with non-overlapping cell populations, we were interested how the classifiers would perform on real data. Therefore, we did the same experiment on the PBMC dataset. The classifiers are trained on the tree shown in Figure S3.

For the simulated data, decreasing the offset of the one-class SVM led to an increased performance. The results on the PBMC dataset contradict these results (Figure 9). Here, decreasing the offset leads to a decreasing performance. For the simulated data, the score distributions of the negative and positive samples were widely separated, while for the PBMC data, this is only true for the B-cells (Figure S6c). For all other populations, the score distributions of the positive and negative samples overlap (Figure S6). Some cell populations, the CD4+ and CD4+ naive T-cells, have positive samples with extremely low scores (Figures S6g and S6i). Especially the minimum offset is affected by these outliers. Due to the low score of these outliers, the offset becomes lower and prediction of the CD4+ and CD4+ naive T-cells is favored over other cell populations. Therefore, many other cells are also predicted to be this class. If we look at the confusion matrix of the one-class SVM with minimum offset, we see that indeed almost all T-cells are predicted to be CD4+ T-cells and CD4+ naive T-cells in particular, while this is not the case for the default offset (Tables S3 and S4). In case of overlapping cell populations, it is thus better to rely on the default offset.

Comparing the linear SVM to the one-class SVM with default offset, we can see that the linear SVM has a better performance: a median HF1-score of 0.986 vs. 0.940. This indicates that the linear SVM can handle overlapping cell populations better than the one-class SVM.

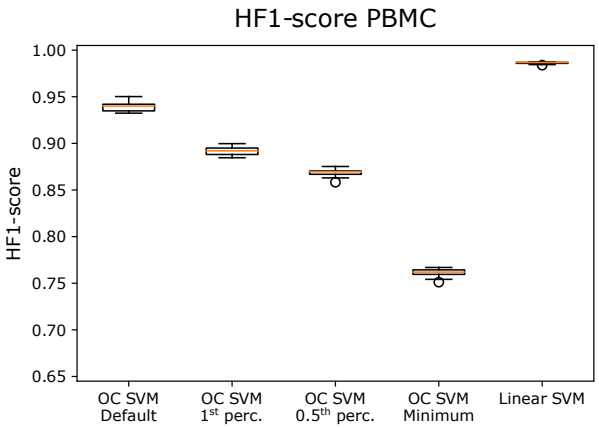


Figure 9: Boxplots showing the HF1-score of the one-class (OC) SVM with different offsets and the linear SVM on the PBMC data.

3.1.3 The one-class SVM suffers when predicting small cell populations

Next, we tested the classification performance on the AMB dataset to investigate the effect of small populations on the classification performance. This dataset is annotated at three levels. The first annotation level indicates whether a cell is a glutamatergic neuron, GABAergic neuron, or non-neuronal cell. The second annotation level includes fifteen different cell populations and the third annotation level distinguishes 92 different populations. Based on these three levels, we created a classification tree to train the classifiers (Figure S4). During the experiment, we used the same cross-validation folds as the benchmarking study in [16] which enables us to compare the results.

First, we wanted to verify our results concerning the offset of the one-class SVM on real data. Comparing the results for the different offsets of the one-class SVM, we notice that, as expected, decreasing the offset leads to less rejected cells and less internal predictions (Figure 10a). Similar to the PBMC data, the percentage of wrongly labeled cells increases. Comparing the scores of the positive and negative samples for each cell population, we see that at the first level in the tree, there is almost no overlap between the two distributions (Figures S7a, S7b and S7c). We can easily predict whether a cell is a GABAergic neuron, glutamatergic neuron or non-neuronal cell. Decreasing the offset will thus lead to more internal nodes that are predicted correctly. At the second level, the scores also show little overlap (Figures S7d, S7f and S7h). At the highest resolution, however, the scores start to overlap completely and no distinction can be made between the two distributions (Figures S7e, S7g and S7i). Here, more mistakes will be made if the offset is decreased. This is also reflected with the HF1-score (Figure 10b). In the beginning, the HF1-score increases, but when the number of wrong predictions increases too much, it starts to decrease. For this dataset, the 1$^{\text{st}}$ perc. offset would be optimal.

Next, we were interested whether the classification performance of the linear and one-class SVM is affected by the small cell populations of the AMB dataset. The performance of both classifiers decreases when the cell populations become smaller, but the performance of the one-class SVM is affected more (Figure S8). Regardless of the offset used, the one-class SVM never predicts the 'Astro Aqp4' cells correctly, while this population is relatively different from the rest as it is the only non-neuronal population. This population, however, only consists of eleven cells. Since the linear SVM can handle the small cell populations better, the overall performance is also higher (Figure 10).

3.1.4 The linear SVM outperforms $SVM_{rejection}$ and CHETAH on the AMB dataset

For the AMB dataset, we also compared our results to existing classifiers. Here, we chose to focus on $SVM_{rejection}$, as benchmarking showed that this was the best classifier at the moment, and CHETAH, as this is also a hierarchical classifier [16]. CHETAH, however, uses a different classification tree. It is not fair to compare these HF1-scores as this can be influenced by, for instance, the depth of the tree. The median F1-score for CHETAH was 0.81, which is lower than the one-class SVM with optimal offset (0.83) and the linear SVM (0.97). Based on this metric, the one-class SVM with optimal offset and the linear SVM both outperform CHETAH.

The linear SVM also shows a better performance than $SVM_{rejection}$. $SVM_{rejection}$, however, has a higher median F1-score. This is mainly because it makes almost no mistakes, only 1.7% of the cells is wrongly labeled (false positives). The number of rejected cells (false negatives), on the other hand, is not considered when calculating the median F1-score. This number is relatively high for $SVM_{rejection}$ (Figure 10a). The linear SVM, on the contrary, has no rejected cells, which is also reflected in a higher HF1-score (Figure 10).

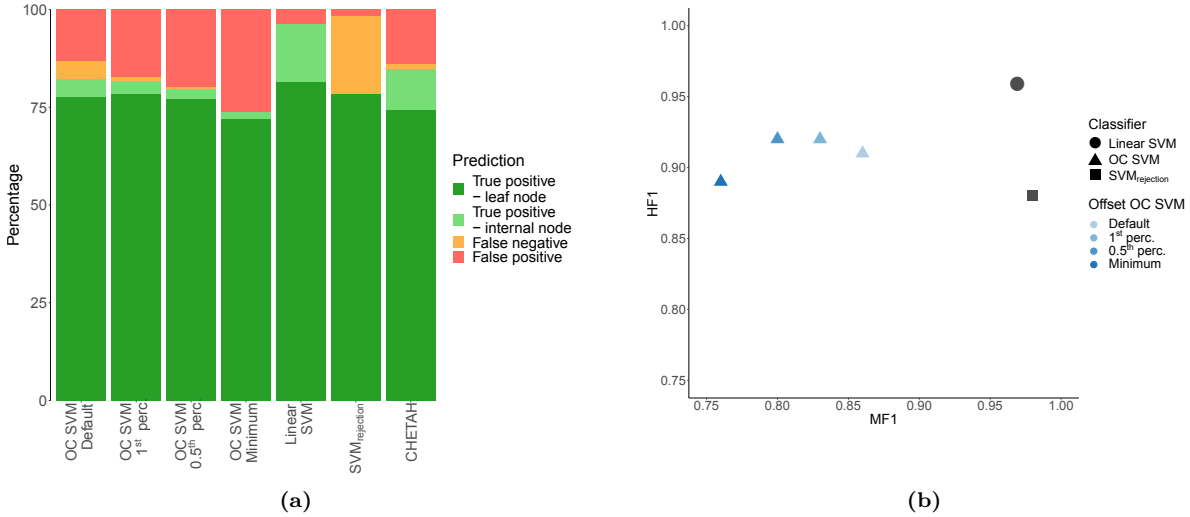


Figure 10: (a) Barplot showing the percentage of true positives, false negatives and false positives per classifier. For the true positives, a distinction is made between the correctly predicted leaf nodes and internal nodes. (b) HF1-score (HF1) and median F1-score (MF1) per classifier.

3.2 Classification tree construction

3.2.1 The classification tree for the simulated data can be reconstructed using the linear and one-class SVM

Next, we tested whether it is possible to construct the classification trees using our hierarchical progressive learning pipeline. Similarly to evaluating the classification performance, we started with an easy dataset, the simulated data. During this experiment, we did a 10 fold cross-validation. The training set was also split into three different batches, Batch 1, Batch 2, and Batch 3. To get different resolutions, some cells were relabeled in these batches (Table 1). Batch 1 contains the lowest resolution and Batch 3 has the highest resolution. During the experiment we tried all orders of the batches, e.g. starting with Batch 1 and adding Batch 2 and afterwards adding Batch 3. This results in six different combinations. Combining this with the 10 fold cross-validation, 60 trees were constructed in total by each classifier.

For the one-class SVM we tried two different offsets, the minimum and default, to test whether this would affect the tree construction. On the simulated data, the offset does not influence this. Using both offsets, there sometimes are some redundant nodes in the tree, but the results are exactly the same (Figure 11b). In fourteen out of sixty experiments, 'Group1'

is for example added to the tree twice. The same cell population is represented by two nodes in the tree. This happens when 'Group1' from a new batch does not match 'Group1' from a previous batch precisely and thus some cells are rejected. The 'Group1' from the new batch seems to be a subpopulation of 'Group1' of the old batch or vice versa. As a result, an extra leaf node is added to the tree. The one-class SVM thus does not always construct the perfect tree, but the offset does not matter.

Next, it would be interesting to test if the order of the batches would make a difference for the classifiers. We wanted to investigate whether starting with a low resolution would for instance be beneficial to starting with a high resolution. Using a linear SVM, the final tree always looks as we expected (Figure 11a). Since all sixty trees are constructed similarly, we can conclude that the order of the batches does not matter and that the linear SVM is stable. The one-class SVM, on the other hand, makes some mistakes. In respectively 14, 14, and 12 experiments, 'Group1', 'Group2', and 'Group4' are added to the tree twice (Figure 11b). This only happens if we have the following order: Batch 1 - Batch 3 - Batch 2 or Batch 3 - Batch 1 - Batch 2. Adding batches in increasing or decreasing order consequently gives perfect results.

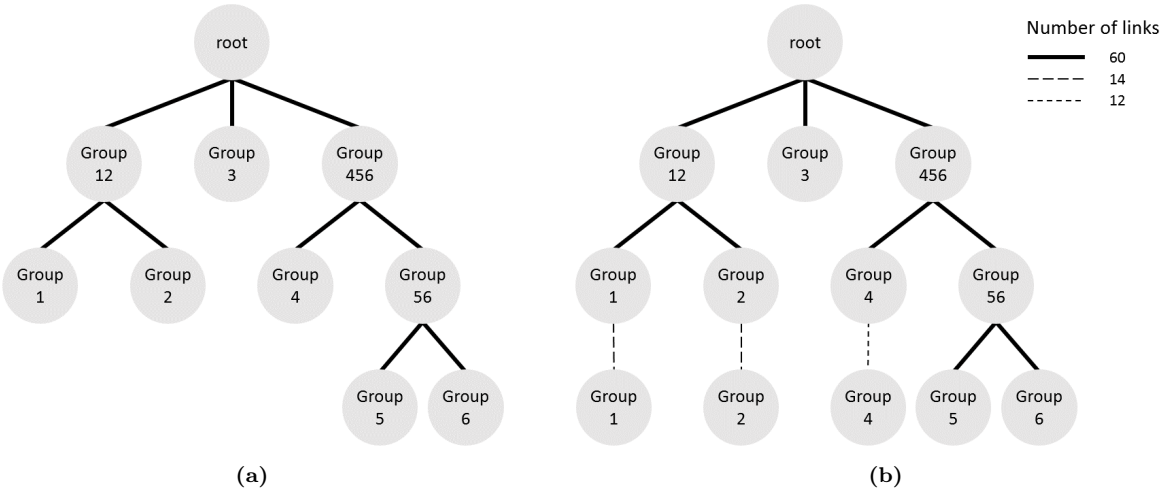


Figure 11: Constructed trees for the simulated data using a (a) linear SVM and (b) one-class SVM with minimum and default offset.

3.2.2 Small cell populations influence tree construction for the one-class SVM with default offset

When testing the classification performance of the one-class SVM, we saw that this was lower on small cell populations. To test whether small populations would also influence tree construction, we downsampled the simulated dataset and tested the construction of the tree. In the original simulated dataset, the size of all the leaf nodes was approximately 1500 cells. Now, the size of the leaf nodes 'Group1', 'Group2', and 'Group4' is halved, and the size of 'Group5' and 'Group6' has become four times smaller. The size of 'Group3' remained the same. This resulted in a dataset of 4251 instead of 8839 cells.

The one-class SVM with minimum offset makes two additional mistakes compared to the original dataset. 'Group 1' is added to the tree twice sixteen times instead of fourteen (Figure S9a). The one-class SVM with default offset, on the contrary, is making many more mistakes (Figure S9b). Only half of the trees is constructed perfectly. Interestingly, there are no problems with 'Group3', which still has the same size, but more problems occur in the other groups that indeed became smaller. From this we can conclude that small cell populations do not only influence the classification performance, but also tree construction for the default offset.

3.2.3 One-class SVM shows more stable results for the PBMC dataset compared to linear SVM

When testing the classification performance, the results on the simulated data were better than the results on the real data, e.g. the PBMC data, as this contains overlapping cell populations. Therefore, we tested the tree construction also on the PBMC dataset using the same experiment as for the simulated data: a 10 fold cross-validation and splitting the training set in three batches. In these batches, we renamed or removed certain populations to get different resolutions (Table 2). Batch 1 again has the lowest resolution and Batch 3 has the highest. All different orders of the batches were tried, which resulted in 60 constructed trees per classifier. During these experiments only the one-class SVM with default offset was used. This had the highest classification performance on the PBMC dataset and, as long as there are no small cell populations, the tree construction is not affected by the offset.

On the PBMC dataset, the one-class SVM shows slightly more stable results than the linear SVM (Figure 12). For the one-class SVM, all 60 constructed trees are the same, while the linear SVM shows little variation. Four trees are slightly different. The CD8+ naive T-cells and CD4+ memory T-cells are connected directly to the T-cells, so the internal node containing the CD8+ T-cells is missing.

The trees constructed using the linear and one-class SVM, however, are slightly different from the hematopoietic tree used when testing the classification performance (Figures 12 and S3). In the constructed trees, CD4+ memory T-cells are a subpopulation of CD8+ instead of CD4+ T-cells. A t-SNE plot of the PBMC dataset also confirms that CD4+ memory T-cells are more similar to CD8+ than CD4+ T-cells (Figure 13). Rewiring this node also improves the classification performance of the linear SVM slightly (HF1-score = 0.992 vs 0.987). Using the hematopoietic tree, the CD4+ memory T-cells, CD8+ T-cells, and CD8+ naive T-cells were also confused mostly, while now less mistakes are made (Tables S5 and S6).

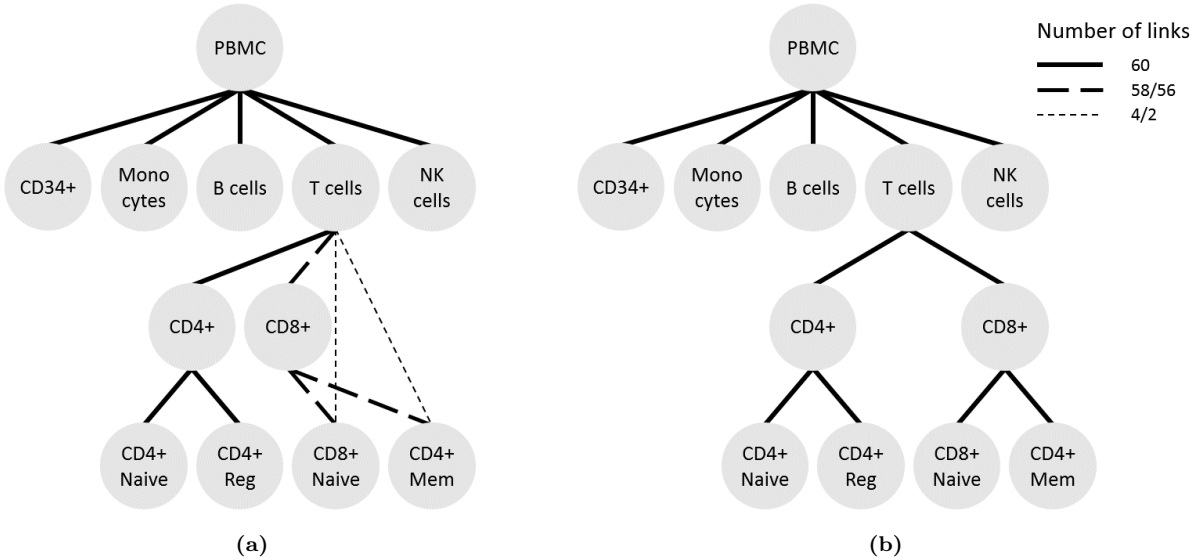


Figure 12: Constructed trees for the PBMC dataset using a (a) linear SVM and (b) one-class SVM with default offset.

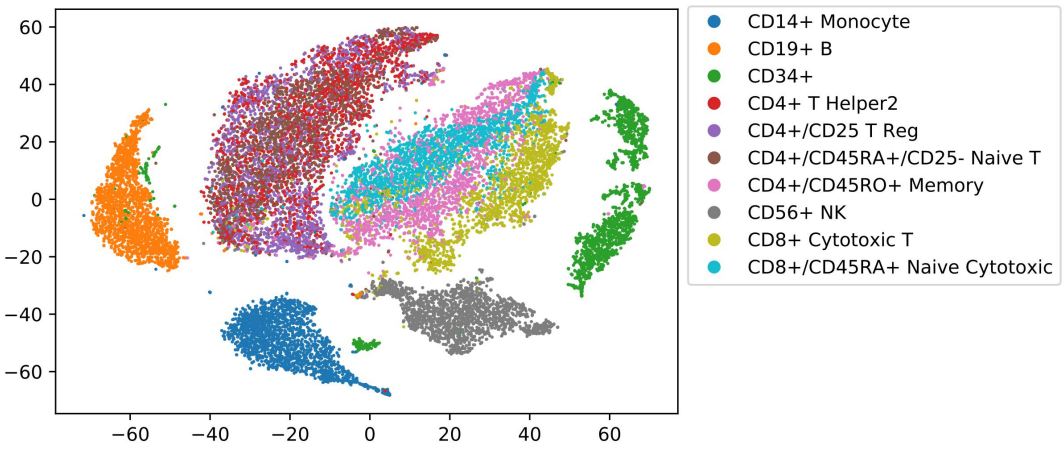


Figure 13: t-SNE plot of PBMC data

3.3 Rejection option

Finally, we compared the rejection option of the one-class and linear SVM. Here, we make a distinction between two scenarios. First, we will investigate whether the classifiers can detect new unseen cell populations in a test set. Furthermore, the rejection option could be important during the tree construction. This could happen, for example, when a certain cell population is missing in a batch of training data. Since this population is in other batches, it is important that the classifier trained on the batch with the missing population rejects these cells instead of mislabeling. If the rejection option is not working correctly, this could affect the construction of the classification tree.

3.3.1 The one-class SVM can detect new cell populations in simulated data

First, we evaluated the rejection option on the simulated data. In this dataset, the cell populations are distinct, so we expect that this is a relatively easy task for the classifiers. We removed one cell population from the training set and used this population as a test set. 'Group3' was removed from the training set as this group is most different from the other populations and should thus be easiest to reject (Figure 7). The one-class SVM with default offset correctly rejects all cells in the test set, while the linear SVM rejects nothing. It labels 20.6 % as 'Group1', 4.6% as 'Group2', 2.9% as 'Group4', 34.8% as 'Group5', and 37.1% as 'Group6'. Here, the one-class SVM clearly outperforms the linear SVM.

3.3.2 Feature selection influences the rejection option of the one-class SVM

Next, we tested the rejection option on real data since overlapping cell populations could make it more difficult to detect new populations. Here, the AMB dataset is most interesting as we could test the rejection option at three different resolutions. During each experiment, we removed a node from a different layer in the tree (and thus also removed the corresponding subpopulations). First, we removed all glutamateric cell populations, 32 in total, from the training set and put them in the test set. In the second experiment, we removed all 'L6 IT' cell populations, five in total, from the training set. Finally, we only removed the 'L6 IT VISp Penk Col27a1' population.

We would expect that the complexity of the task increases during the three experiments. Even though the glutamatergic neurons are completely separated from the other two cell types, both the linear and one-class SVM show the worst performance during the first experiment (Figures 14 and S7b). The linear SVM even rejects none of the cells. After removing the

glutamatergic neurons from the tree, the root node only has two children. In a linear one-vs-all setting with only two classes, the whole decision space is divided and there is no space for a rejection option. The failing rejection option of the one-class SVM could be caused by the feature selection method. We chose interesting PCs based on the results of the t-test (Section 2.1.1). In the first experiment, when the glutamatergic neurons are removed, the t-test is performed between the GABAergic neurons versus non-neuronal cells. Without removing the glutamatergic neurons, this t-test is performed between the GABAergic versus non-neuronal cells and glutamatergic neurons. This will result in different PCs. In the first scenario, 22 PCs are selected, while in the second a subset of those, eight PCs, plus two extra are chosen. The 22 PCs will probably only differentiate between neuronal and non-neuronal cells, but not between glutamatergic and GABAergic neurons. As a consequence, the glutamatergic cells of the test set get a high score for the GABAergic neurons and are not rejected. To verify these thoughts, we first selected the PCs and then removed the glutamatergic cells instead of vice versa. In this biased scenario, all glutamatergic cells are rejected by the one-class SVM using the default, 1^{st}, and 0.5^{th} perc. offset. Using the minimum offset 94.4% of the cells are rejected. This indicates that the feature selection method has a strong influence on the performance of the rejection option of the one-class SVM.

In general, we can conclude that using a one-class SVM with default offset improves the rejection option compared to the other offsets. It depends, however, per removed cell population whether the linear or one-class SVM with default offset is better.

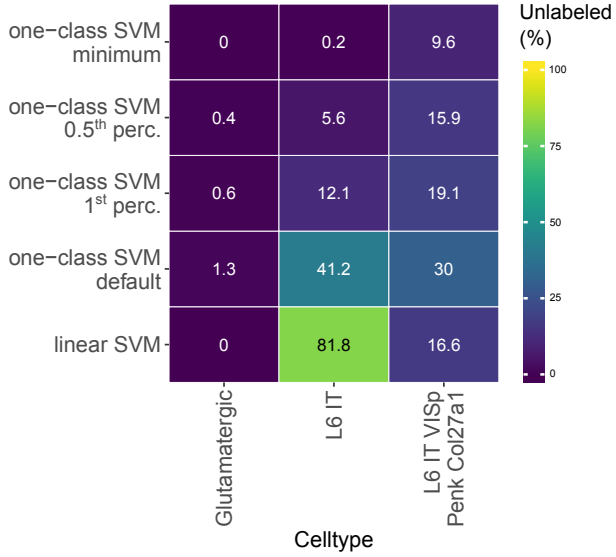


Figure 14: Percentage of cells rejected or correctly labeled at a higher resolution (e.g. when L6 IT VISp Penk Col27a1 cells are removed from the tree, it is correct if cells are rejected or if they are labeled glutamatergic neurons or L6 IT cells.)

3.3.3 Missing cell populations in the training data influence tree construction for the linear SVM

Next, we tested whether new or missing cell populations in the training set could influence tree construction. We mimicked this scenario using the simulated dataset and the same batches as in the previous tree construction experiment. In this case, we removed 'Group5' from 'Group56' in Batch 2, so Batch 2 only contains 'Group6' (Table S7). As in the previous experiments, we did a 10 fold cross-validation and tried all different orders of the batches, so 60 trees were constructed in total. Looking at the results of the linear SVM, we notice that it has difficulties with rejecting cells (Figure 15a). If 'Group5' is present in one batch and absent in another, it is not rejected, but all cells are labeled as 'Group6'. Therefore, 'Group6' is added as a parent node to 'Group5' and 'Group6'. The one-class SVM with default offset, on the other hand,

has no problems with the missing cell population (Figure 15b). Some trees (14 out of 60) are not constructed correctly, but similar to the normal experiment, increasing or decreasing the resolution of the batches leads to perfect results.

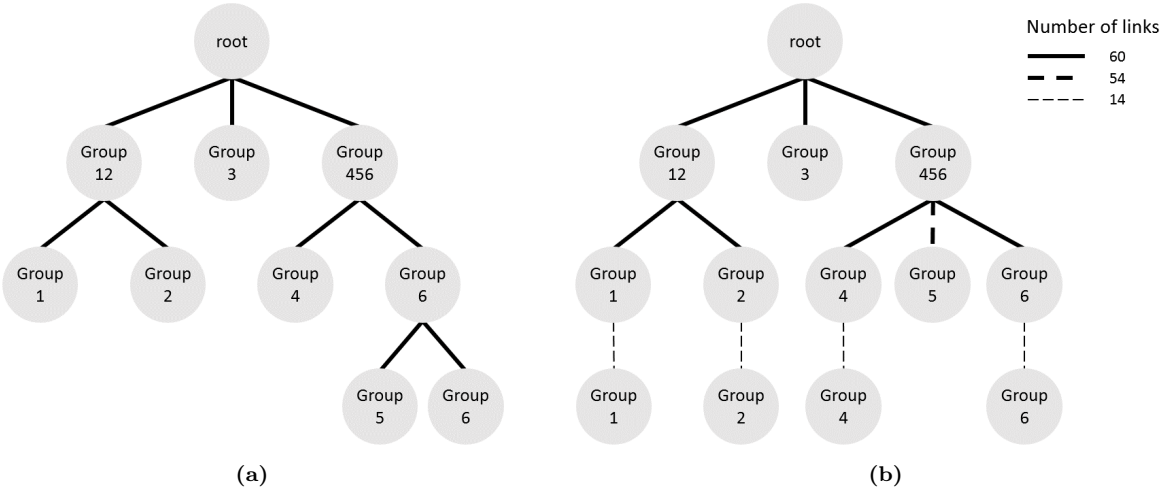


Figure 15: Constructed trees for the simulated data using a (a) linear SVM and (b) one-class SVM with default offset when testing the rejection option.

4 Discussion

In this work, we present a hierarchical progressive learning pipeline to construct a classification tree using multiple datasets. We evaluated the pipeline on three different aspects: classification performance, tree construction, and the rejection option.

4.1 Classification performance

On all datasets, the performance of the linear SVM is equal to or higher than the one-class SVM. For both classifiers, we saw a decrease in performance on smaller populations, but for the one-class SVM this effect was more apparent. On the AMB dataset, the linear SVM even outperformed $\text{SVM}_{\text{rejection}}$, which was the best performing classifier when benchmarking several methods [16]. The main advantage of the linear SVM compared to $\text{SVM}_{\text{rejection}}$ is the hierarchy used. On the AMB dataset, the linear SVM does not reject any of the cells, but labels them as an internal node. During this experiment, there were no unknown populations. Correct internal node prediction instead of rejection is therefore beneficial since it provides the user with extra information.

Both the linear and one-class SVM outperform CHETAH, currently the only other classifier that exploits a hierarchy without using prior knowledge. During the comparison, however, we had to use a flat classification measure, the median F1-score. We could not use the HF1-score as CHETAH used a different classification tree than we did.

On each dataset, we tested different offsets for the one-class SVM. Every dataset, however, gave a different optimum. The minimum offset was most optimal for the simulated dataset, since there are no overlapping cell populations. On the PBMC and AMB dataset the default and 1^{st} perc. offset yield the best results respectively. Ideally, this offset might even be different for each layer in the tree. In the AMB dataset, for instance, it was relatively easy to distinguish GABAergic neurons, glutamatergic neurons, and non-neuronal cells, so here a minimum offset could be used. At a higher resolution, however, the cell populations overlap, so the default offset might be better. The optimal offset could, for instance, be determined using a nested

cross-validation loop. Another option might be to determine the offset based on the overlap of cell populations. If, for example, there is no overlap, a minimum offset could be chosen, but if the overlap increases, the offset should also increase.

4.2 Tree construction

Considering the tree construction, we showed that the linear and one-class SVM could both reconstruct the classification tree for the simulated data. For the one-class SVM it is important though that the batches are added in increasing or decreasing resolution consequently. When applying the pipeline to new datasets, however, it may be difficult to know which dataset has the highest or lowest resolution. In these situations, looking at the number of cell populations in each dataset could give a good indication of the best order.

Interestingly, both the linear and one-class SVM reconstruct a different classification tree for the PBMC data than expected. In this tree, the CD4+ memory T-cells were a child of the CD8+ T-cells instead of the CD4+ T-cells. In the PBMC dataset, the cells were labeled using FACS, which is based on surface protein expression. It is known that gene and protein expression are poorly correlated [27, 28]. It could be that based on protein expression, the CD4+ memory T-cells are more similar to CD4+ T-cells, which is reflected in the hematopoietic tree, but that based on the gene expression the CD4+ memory T-cells are more similar to CD8+ T-cells. CITE-seq data of these cell populations could be used to verify these thoughts.

4.3 Rejection option

When testing the rejection option, the one-class SVM clearly outperforms the linear SVM by showing a perfect performance on the simulated dataset. On the AMB dataset, however, we noticed that the feature selection method has a huge influence on the rejection option of the one-class SVM. We chose features that only discriminate between the cell populations that are in the training set. This makes it more difficult to identify new populations in the test set.

4.4 Computation time

The computation time of the one-class SVM is considerably longer than the linear SVM. On the PBMC data, for instance, the training time of the one-class SVM was approximately one hour compared to two minutes for the linear SVM. For the one-class SVM, 98% of the time was taken by the PCA step. Considering the computation time the current feature selection method takes, it is also important to improve it. The datasets used in this work are still relatively small compared to, for instance, the mouse organogenesis cell atlas (2 million cells) [5]. Applying a normal PCA will not be feasible here. The feature selection could for instance be optimized by applying L2-regularization instead of PCA for the one-class SVM.

Online learning could be another option to reduce the computation time. Currently, the classifiers are retrained from scratch when we have added a new dataset and updated the tree. Especially when the datasets become bigger, this is time-consuming. It would be more efficient to only update the decision boundary using this new dataset. This could, for instance, be done using stochastic gradient descent (SGD). If SGD is used to find the optimal decision boundary, only one training example is considered at a time. The decision boundary could thus first be optimized using the first batch of training samples one by one. Afterwards, if a new batch comes in, those training samples can be considered one by one to update the decision boundary.

For the linear SVM, however, updating the decision boundary might be difficult if a new class is discovered. This new class will influence the decision boundary of all other populations directly. For the one-class SVM, on the contrary, the decision boundary is only affected by the class itself, so updating would be possible.

4.5 Limitations

We used the HF1-score to evaluate the classification performance of the classifiers. This, however, is not an optimal evaluation measure as the structure of the tree can influence the score. Nodes close to the root are weighted equally to leaf nodes. If there are for example more internal nodes that are easily predicted, the HF1-score can become high even if there are no correctly predicted leaf nodes. Ideally, the HF1-score would be a weighted HF1-score. Labels closer to the root should be weighted less than leaf nodes. This would also make it easier to compare different trees.

Both the linear and the one-class SVM have a higher classification performance on the simulated data (HF1-score = 1) compared to the PBMC (HF1-score = 0.987 and 0.940) and AMB dataset (HF1-score = 0.959 and 0.924). This might indicate that the simulated data is too easy and does not resemble all characteristics of scRNA-seq data completely. The cell populations could have been simulated too distinct in the simulated data. Furthermore, it might be that technical noise is missing. In the simulated data, the percentage of dropouts is only 24.8%, while this is 75.6% and 97.1% in the AMB and PBMC dataset respectively. Even though simulated data might be helpful when developing a new method, it is important to not solely rely on it and remain testing with real data.

During all tree construction experiments, we split an existing dataset into different batches to test the performance. Ideally, the different batches would be different datasets. An across dataset comparison is needed to test the tree construction performance in a more realistic scenario. The performance of classifiers during inter-dataset experiments is usually considerably lower than during intra-dataset experiments [16].

4.6 Conclusion and outlook

At the moment, choosing between a one-class and a linear classifier is a trade-off between the ability of discovering new cell populations and a higher accuracy. There are, however, some aspects, such as the offset or feature selection method, that could be improved to increase the performance of the one-class SVM. In general, the concept of hierarchical progressive learning has already shown its potential during this project. In the future, this method could be useful during cohort studies. This pipeline could for instance be used in the single-cell eQTLGEN (sc-eQTLGEN) consortium where they try to discover eQTLs that are cell type-specific [29]. Here, it is important that the datasets of all cohorts are annotated consistently. To achieve this, the first available datasets could be used to build the hierarchical tree, which could subsequently be used to annotate cells in the other datasets. Especially during these types of experiments hierarchical progressive learning could be valuable.

References

[1] F. Tang, C. Barbacioru, Y. Wang, E. Nordman, C. Lee, N. Xu *et al.*, "mRNA-Seq whole-transcriptome analysis of a single cell," *Nature Methods*, vol. 6, no. 5, pp. 377–382, 5 2009.

[2] B. Tasic, V. Menon, T. N. Nguyen, T. K. Kim, T. Jarsky, Z. Yao *et al.*, "Adult mouse cortical cell taxonomy revealed by single cell transcriptomics." *Nature neuroscience*, vol. 19, no. 2, pp. 335–46, 2 2016.

[3] B. Tasic, Z. Yao, L. T. Graybuck, K. A. Smith, T. N. Nguyen, D. Bertagnolli *et al.*, "Shared and distinct transcriptomic cell types across neocortical areas," *Nature*, vol. 563, no. 7729, pp. 72–78, 11 2018.

[4] V. Svensson and E. d. V. Beltrame, "A curated database reveals trends in single cell transcriptomics," *bioRxiv*, p. 742304, 8 2019.

[5] J. Cao, M. Spielmann, X. Qiu, X. Huang, D. M. Ibrahim, A. J. Hill *et al.*, "The single-cell transcriptional landscape of mammalian organogenesis," *Nature*, vol. 566, no. 7745, pp. 496–502, 2 2019.

[6] A. Regev, S. A. Teichmann, E. S. Lander, I. Amit, C. Benoist, E. Birney *et al.*, "The human cell atlas," *eLife*, vol. 6, 12 2017.

[7] A. Zeisel, H. Hochgerner, P. Lönnerberg, A. Johnsson, F. Memic, J. van der Zwan *et al.*, "Molecular Architecture of the Mouse Nervous System." *Cell*, vol. 174, no. 4, pp. 999–1014, 2018.

[8] M. G. Van Der Wijst, H. Brugge, D. H. De Vries, P. Deelen, M. A. Swertz, and L. Franke, "Single-cell RNA sequencing identifies celltype-specific cis-eQTLs and co-expression QTLs," *Nature Genetics*, vol. 50, no. 4, pp. 493–497, 4 2018.

[9] L. Tian, X. Dong, S. Freytag, K.-A. Lê Cao, S. Su, A. JalalAbadi *et al.*, "Benchmarking single cell RNA-sequencing analysis pipelines using mixture control experiments," *Nature Methods*, vol. 16, no. 6, pp. 479–487, 6 2019.

[10] G. X. Y. Zheng, J. M. Terry, P. Belgrader, P. Ryvkin, Z. W. Bent, R. Wilson *et al.*, "Massively parallel digital transcriptional profiling of single cells," *Nature Communications*, vol. 8, p. 14049, 1 2017.

[11] H. A. Pliner, J. Shendure, and C. Trapnell, "Supervised classification enables rapid annotation of cell atlases," *Nature Methods*, pp. 1–4, 9 2019.

[12] J. K. de Kanter, P. Lijnzaad, T. Candelli, T. Margaritis, and F. C. P. Holstege, "CHETAH: a selective, hierarchical cell type identification method for single-cell RNA sequencing," *Nucleic Acids Research*, vol. 47, no. 16, pp. e95–e95, 6 2019.

[13] F. Wagner and I. Yanai, "Moana: A robust and scalable cell type classification framework for single-cell RNA-Seq data," *bioRxiv*, p. 456129, 10 2018.

[14] T. E. Bakken, R. D. Hodge, J. A. Miller, Z. Yao, T. N. Nguyen, B. Aevermann *et al.*, "Single-nucleus and single-cell transcriptomes compared in matched cortical cell types," *PLOS ONE*, vol. 13, no. 12, p. e0209648, 12 2018.

[15] B. D. Aevermann, M. Novotny, T. Bakken, J. A. Miller, A. D. Diehl, D. Osumi-Sutherland *et al.*, "Cell type discovery using single-cell transcriptomics: implications for ontological representation," *Human Molecular Genetics*, vol. 27, no. R1, pp. R40–R47, 5 2018.

[16] T. Abdelaal, L. Michielsen, D. Cats, D. Hoogduin, H. Mei, M. J. T. Reinders, and A. Mahfouz, "A comparison of automatic cell identification methods for single-cell RNA sequencing data," *Genome Biology*, vol. 20, no. 1, p. 194, 12 2019.

[17] P. Jarvis, *Towards a Comprehensive Theory of Human Learning*. Taylor & Francis Ltd, 2006.

[18] B. H. Yang and H. Asada, "Progressive learning and its application to robot impedance learning," *IEEE Transactions on Neural Networks*, vol. 7, no. 4, pp. 941–952, 1996.

[19] H. M. Fayek, "Continual Deep Learning via Progressive Learning," Ph.D. dissertation, RMIT University, 2019.

[20] J. Alquicira-Hernandez, Q. Nguyen, and J. E. Powell, "scPred: Cell type prediction at single-cell resolution," *bioRxiv*, p. 369538, 12 2018.

[21] K. Boufea, S. Seth, and N. N. Batada, "scID: Identification of transcriptionally equivalent cell populations across single cell RNA-seq data using discriminant analysis," *bioRxiv*, p. 470203, 2019.

[22] D. Tax, "One-class classification Concept-learning in the absence of counter-examples," Ph.D. dissertation, TU Delft, 2001.

[23] T. Fagni and F. Sebastiani, "On the Selection of Negative Examples for Hierarchical Text Categorization," *Proceedings of the 3rd language technology conference*, pp. 24–28, 2007.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel *et al.*, "Scikit-learn: Machine Learning in Python," Tech. Rep., 2011.

[25] S. Kiritchenko and F. Famili, "Functional Annotation of Genes Using Hierarchical Text Categorization," *Proceedings of BioLink SIG, ISMB*, 1 2005.

[26] L. Zappia, B. Phipson, and A. Oshlack, "Splatter: simulation of single-cell RNA sequencing data," *Genome Biology*, vol. 18, no. 1, p. 174, 12 2017.

[27] R. de Sousa Abreu, L. O. Penalva, E. M. Marcotte, and C. Vogel, "Global signatures of protein and mRNA expression levels," *Molecular BioSystems*, vol. 5, no. 12, pp. 1512–1526, 11 2009.

[28] C. Vogel and E. M. Marcotte, "Insights into the regulation of protein abundance from proteomic and transcriptomic analyses," *Nature Reviews Genetics*, vol. 13, no. 4, pp. 227–232, 4 2012.

[29] M. G. P. van der Wijst, D. H. de Vries, H. E. Groot, G. Trynka, C.-C. Hon, M. C. Nawijn *et al.*, "Single-cell eQTLGen Consortium: a personalized understanding of disease," *arXiv*, 9 2019.

Supplementary

Supplementary Note 1

During the matching we distinguish five options: simple, multiple columns, multiple rows, complex, and impossible.

Simple

In this scenario, we find a unique match between a node from dataset 1 and a node from dataset 2. In the corresponding row and column, no other matches are thus found. Within this scenario, there are three different options:

1. Both nodes are leaf nodes. This indicates a perfect match. The tree is not updated, but the labels of dataset 2 are (Figure S10).

2. The node from dataset 1 is a leaf node, but the node from dataset 2 is the root node. This indicates that this node from dataset 1 is missing in dataset 2. The node, however, is already in the tree, so tree 1 can remain the same (Figure S11).

3. The node from dataset 1 is the root, but the node from dataset 1 is a leaf node. This indicates that this node from dataset 2 is missing in dataset 1. The node is thus also not in the tree yet, so we will add it as a child to the root (Figure S12).

Multiple rows

In this scenario, a node from dataset 1 matches multiple nodes from dataset 2. As tree 1 is the tree we are updating each time, we assume that tree 2 only consists of a root node and leaf nodes. Here, we distinguish two different scenarios:

1. A node from dataset 1 matches only leaf nodes from dataset 2. We consider the nodes from dataset 2 subpopulations of the node from dataset 1, so we add these nodes as descendants (Figure S13).

2. The root node of dataset 2 is also involved. We simple ignore this node and for the rest do the same as above (Figures S14 and S15).

Multiple columns

This scenario is quite similar to multiple rows scenario. Here, however, multiples nodes from dataset 1 match one node of dataset 2. This scenario is a little more complex as the nodes from dataset 1 does not have to be leaf nodes or the root node, but there can also be internal nodes in this tree. Here, we distinquish three different scenarios:

1. The root node of tree 1 and tree 2 are not involved, so multiple leaf or internal nodes from tree 1 match a leaf node from tree 2. We consider the nodes from tree 1 subpopulations of the node from tree 2, so we need to merge these nodes into the node from tree 2 (Figure S16). We need to check, however, whether this node already exists in this tree, because we do not want redundant nodes (Figure S17). If this is the case, we have a perfect match between a node from tree 1 and tree 2, so we do not have to update the tree, but we only have to update the labels.

2. Besides leaf or internal nodes, the root node of tree 1 is involved. This indicates that the population node from tree 2 is 'bigger' than the other leaf/internal nodes from tree 1 as part of it is unlabeled. Therefore, we add this node from tree 2 as a descendant to the

root of tree 1. Next, we rewire the nodes from tree 1 such that nodes from tree 1 that had a match with this node become descendants of it (Figure S18).

3. The root node of tree 2 is involved. This indicates that multiple nodes from dataset 1 are missing in dataset 2. These nodes, however, are already in the tree, so the tree can remain the same (Figure S19).

Complex

The scenarios described above were all relatively easy. A node from one dataset matches either one or multiple nodes from another. It could also happen, however, that multiple nodes from dataset 1 match multiple nodes from dataset 2 (Figure S20b). Here, we distinguish three different scenarios:

1. The root node of dataset 1 is involved. We just assume that the boundary should be adjusted and this is automatically done, so we remove this '1' from the table (Figure S20). If the situation is still complex after the one is removed, we continue to scenario 2 or 3. If not, we treat it as a multiple rows problem as explained above.

2. The root node of dataset 2 is involved. Again, we just assume that the boundary should be adjusted, so we remove this '1' from the table (Figure S21). If the situation is still complex after the one is removed, we continue to scenario 3. If not, we treat it as a multiple columns problem as explained above.

3. Multiple leaf/internal nodes of dataset 1 are involved and multiple leaf nodes of dataset 2. We can only solve this if the 'complex' node of dataset 1 is not a leaf node. Otherwise we are dealing with an impossible scenario which is described below. If the complex node is an internal node, we attach the involved row nodes as descendants to the complex node and attach the involved column nodes to the corresponding row node that was just added (Figure S22).

Impossible

Sometimes, it could be impossible to match the labels from two datasets. Something could have gone wrong during the clustering, e.g. a population 1 and 2 from dataset 1 match population A from dataset 2, but population 2 also matches population C from dataset 2 (Figure S23). Here, population A and C should be merged into population 2, but population A should also be split into population 1 and 2. Population 2, however, cannot be added to the tree twice.

It could also be that dataset 2 contains labels at a different resolution, e.g. that population B is a subpopulation of population A (Figure S24). This is not what we assumed and thus impossible to match.

Both scenarios occur when a leaf node from dataset 1 is at a crossing of multiple rows and multiple columns (i.e. a complex situation). An extra difficulty is that there are thus multiple situations that could explain this. All of these situation are not what we desired and thus we call it impossible and do nothing.

Supplementary Figures

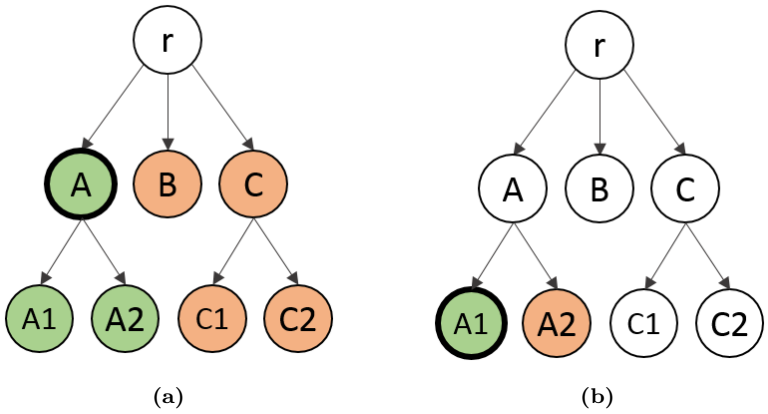


Figure S1: Two examples of the siblings policy. The cell population for which we are training the classifier is visualized in bold, the positive trainings samples in green, and the negative samples in orange.

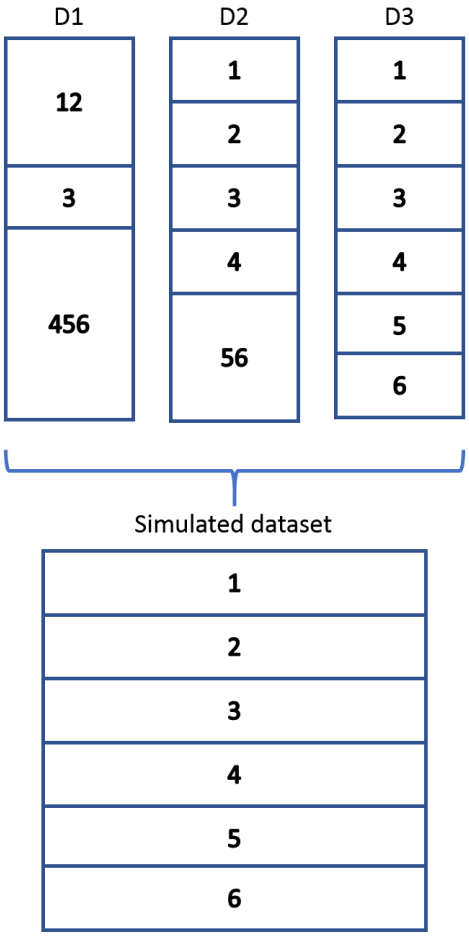


Figure S2: Schematic overview of how the simulated dataset was generated.

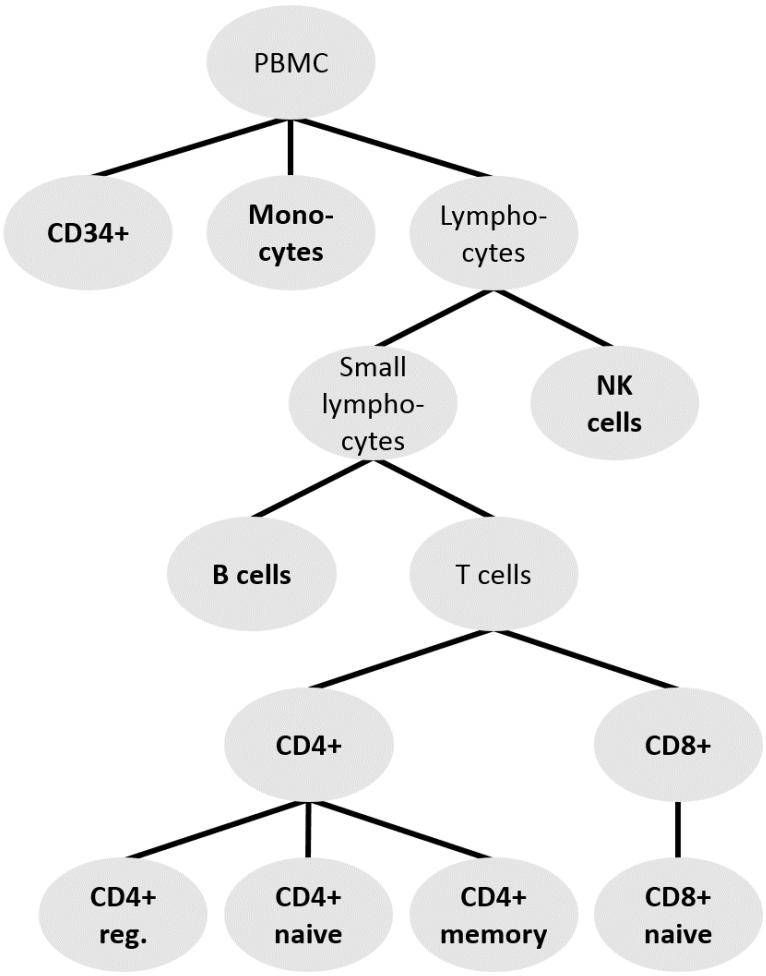


Figure S3: Classification tree of the PBMC dataset. Bold names indicate cell populations that exist in our dataset.

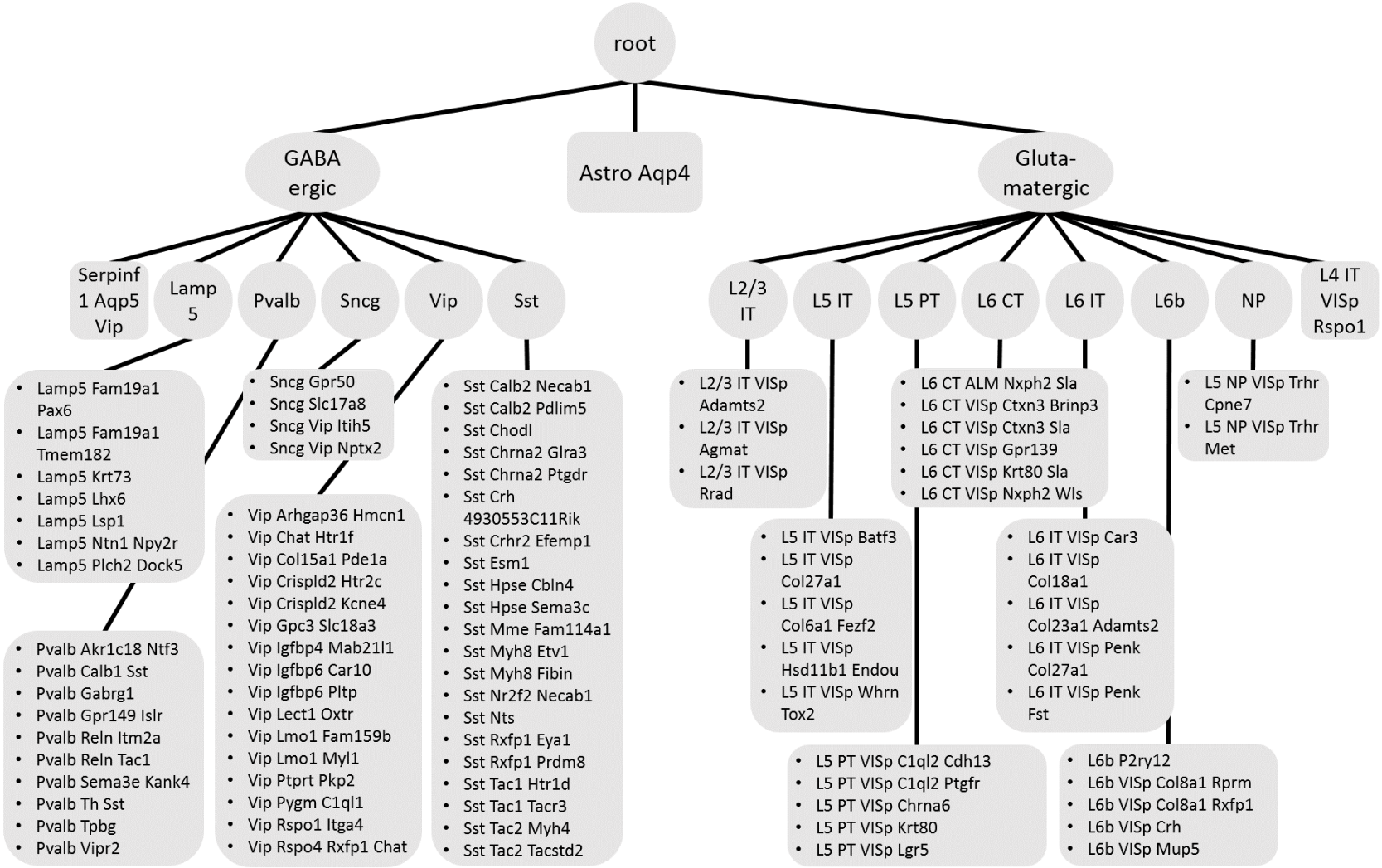


Figure S4: Classification tree of the AMB dataset. The circular and rectangle nodes indicate internal and leaf nodes respectively. If multiple leaf nodes are descendants of the same internal node, they are placed in the same rectangle.

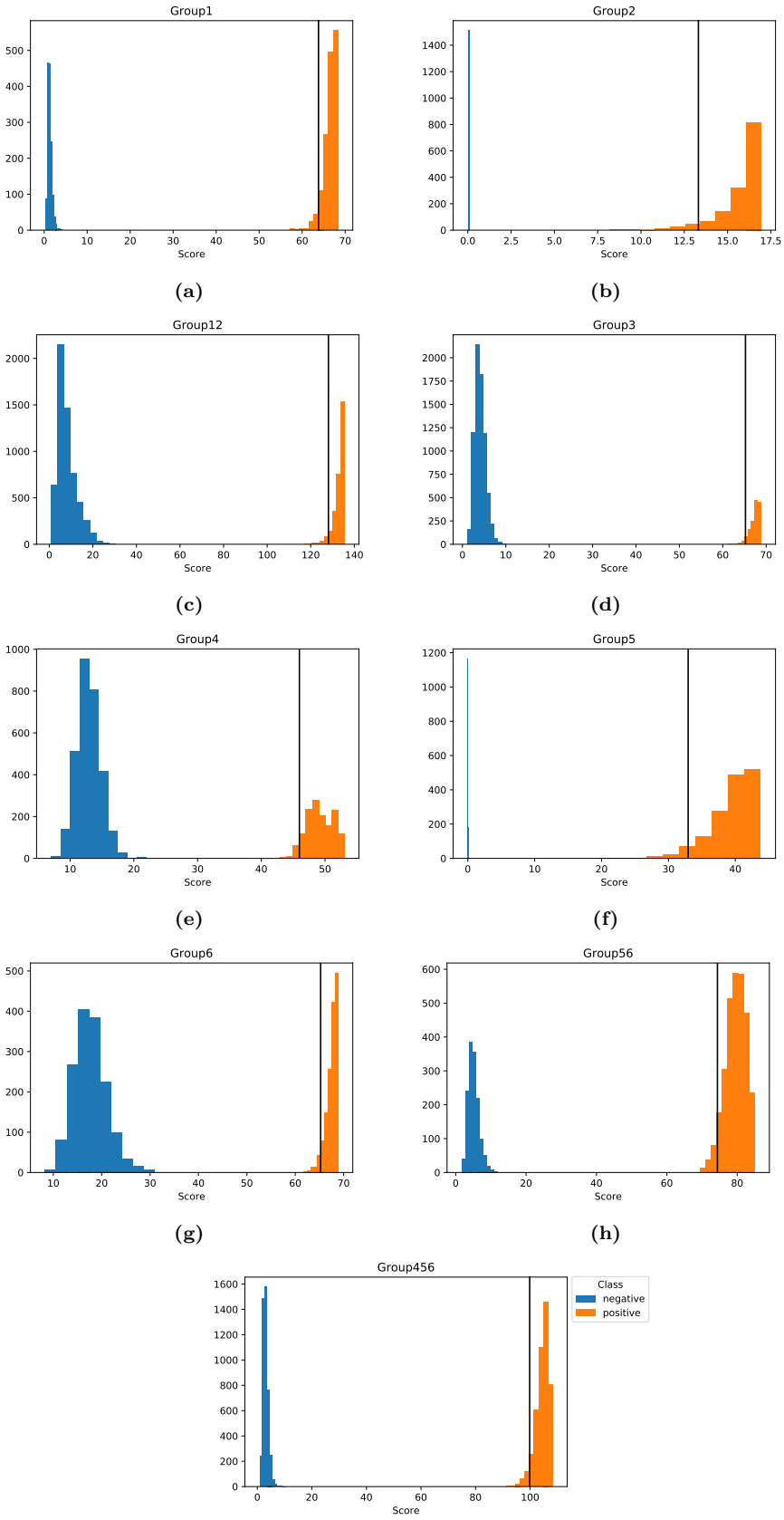


Figure S5: A comparison of the scores of the positive and negative samples of each group. The black line indicates the default offset.

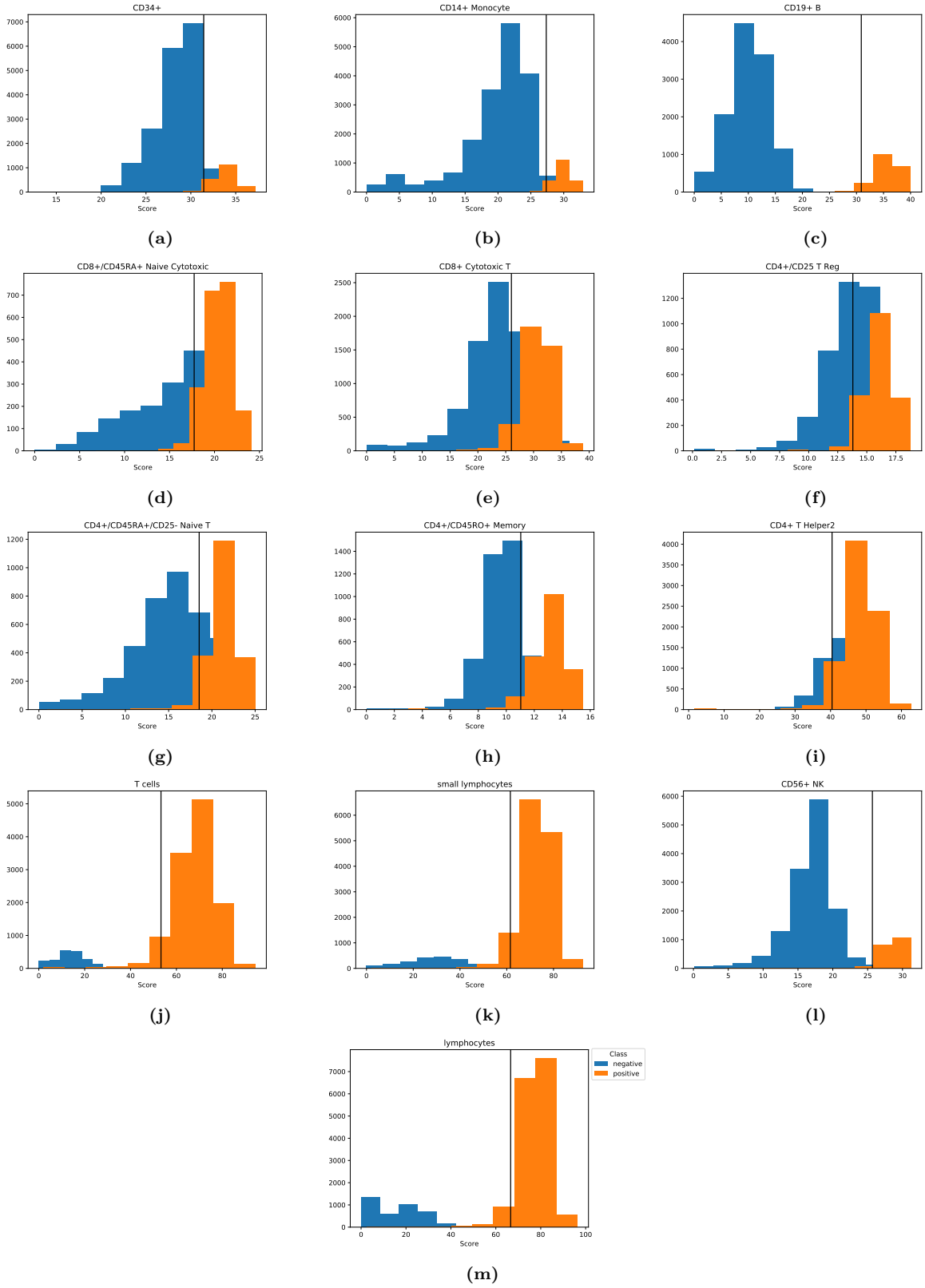


Figure S6: A comparison of the scores of the positive and negative samples of each group. The black line indicates the default offset.

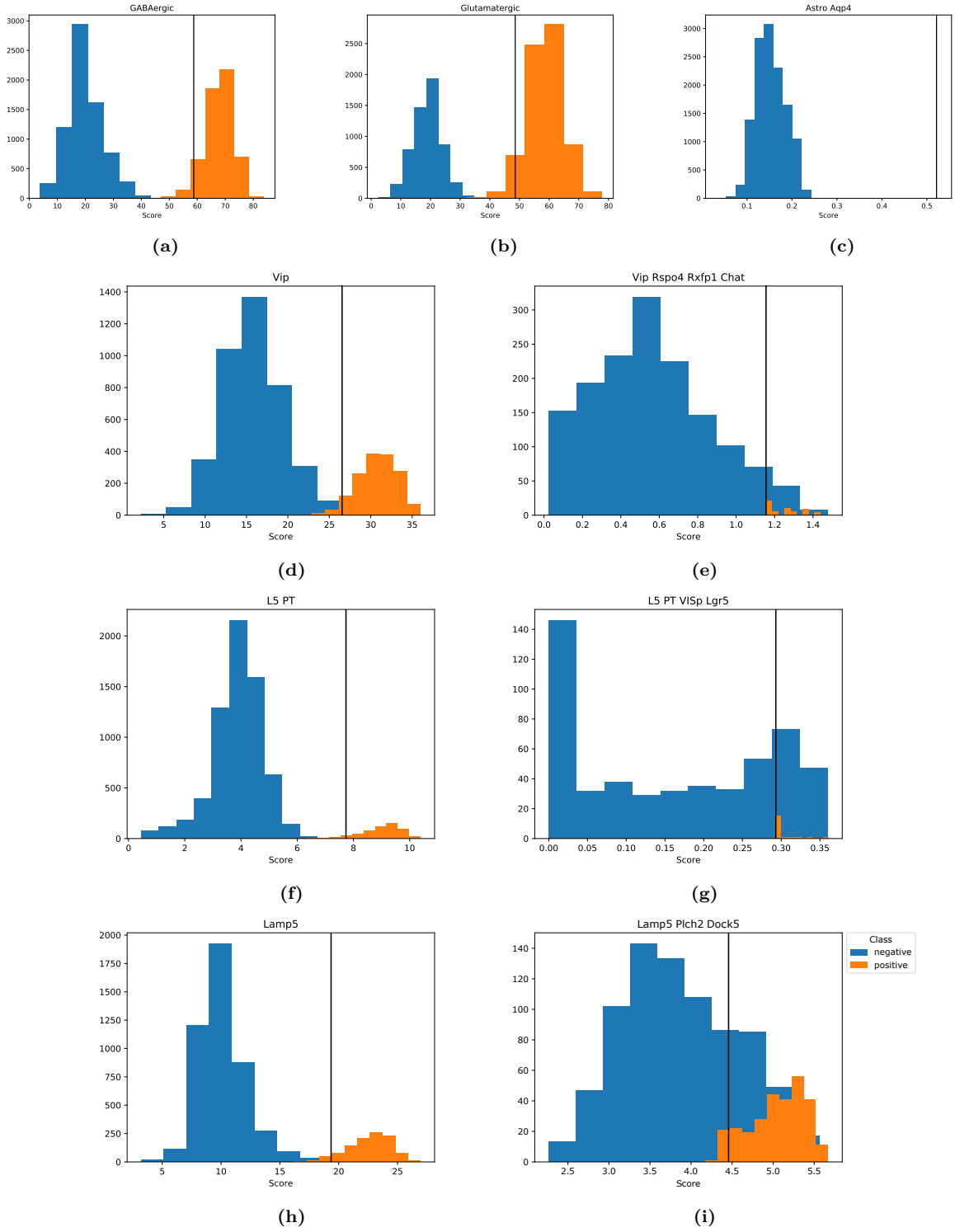


Figure S7: A comparison of the scores of the positive and negative samples of each group. The black line indicates the default offset.

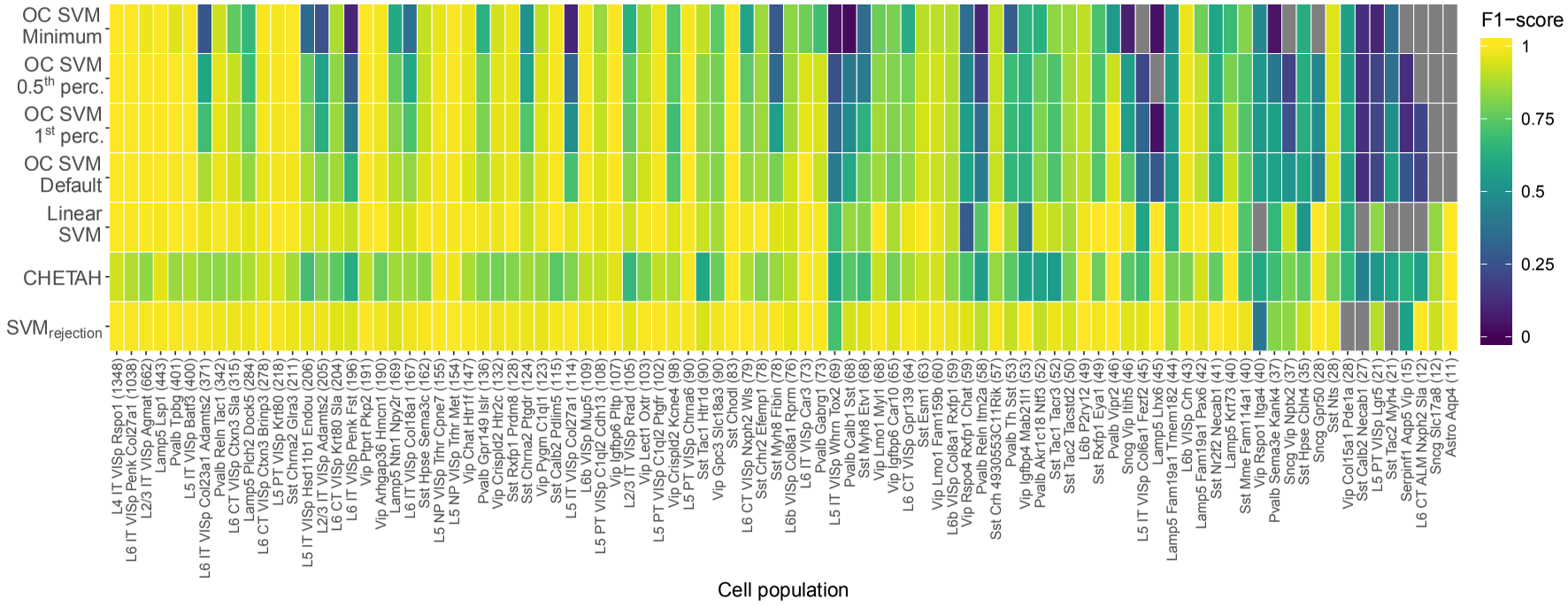


Figure S8: F1 score per cell population.

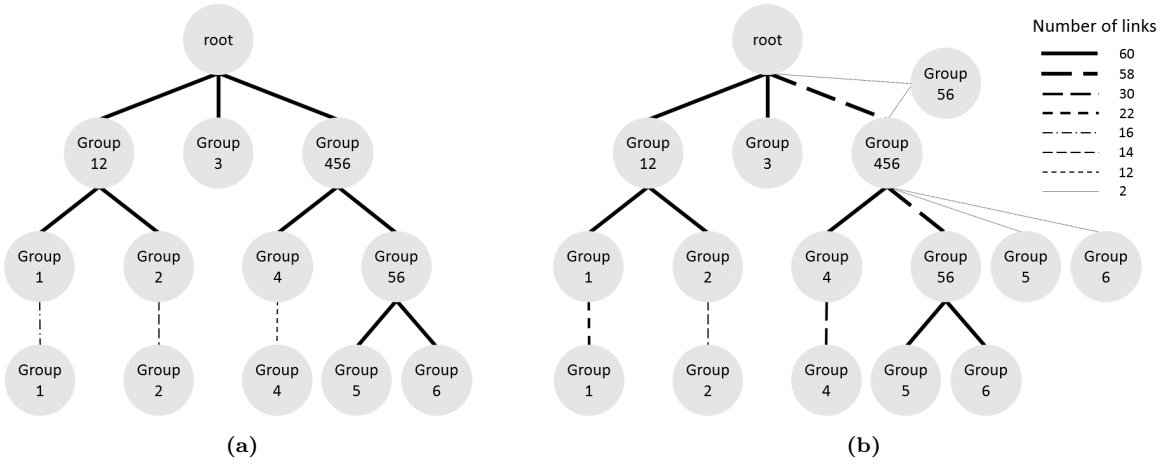
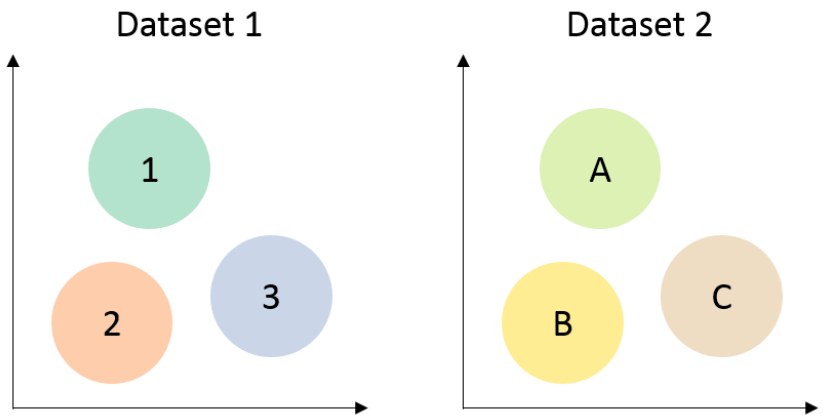


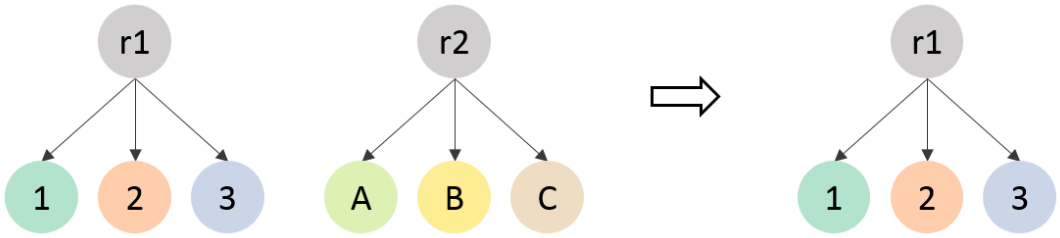
Figure S9: Constructed trees using a one-class SVM with (a) minimum and (b) default offset for the downsamples simulated dataset.



(a)

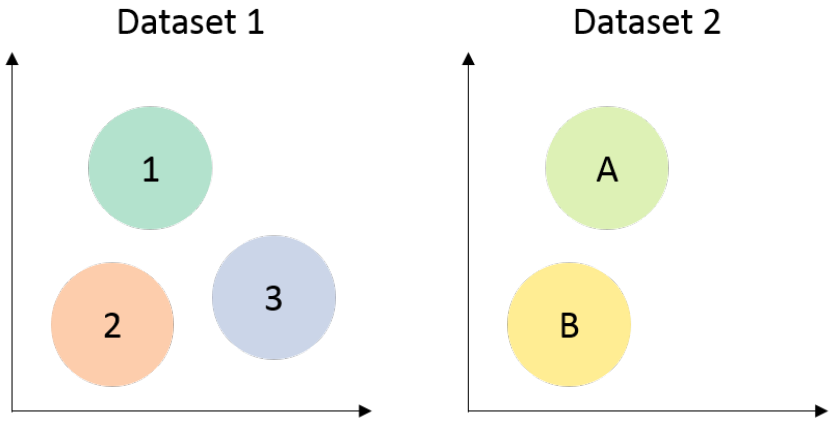
	1	2	3	root1
A	**2**	0	0	0
B	0	**2**	0	0
C	0	0	**2**	0
root2	0	0	0	0

(b)



(c)

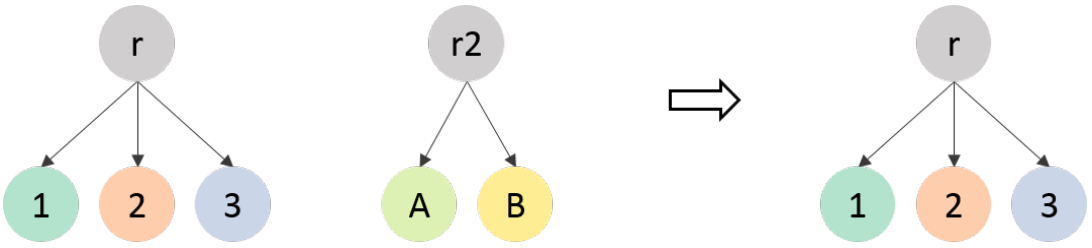
Figure S10: Simple, scenario 1



(a)

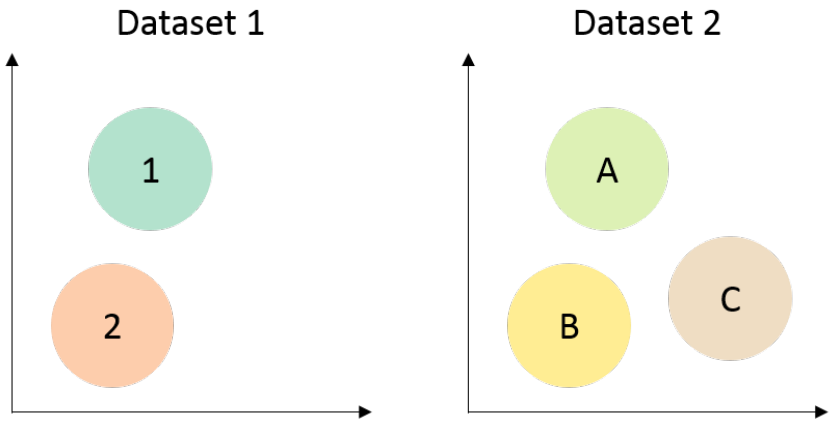
	1	2	3	root
A	**2**	0	0	0
B	0	**2**	0	0
root2	0	0	**1**	0

(b)



(c)

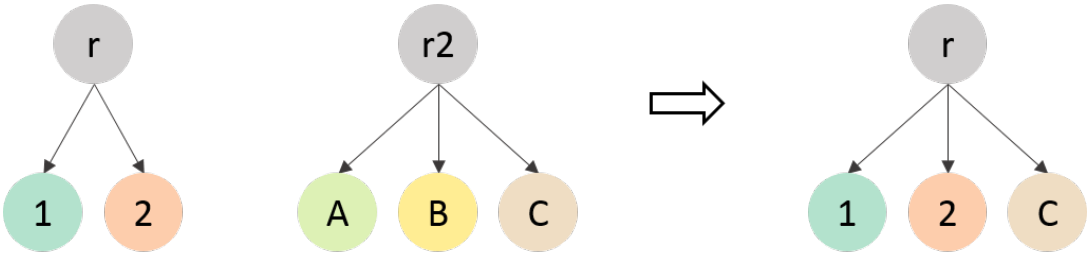
Figure S11: Simple, scenario 2



(a)

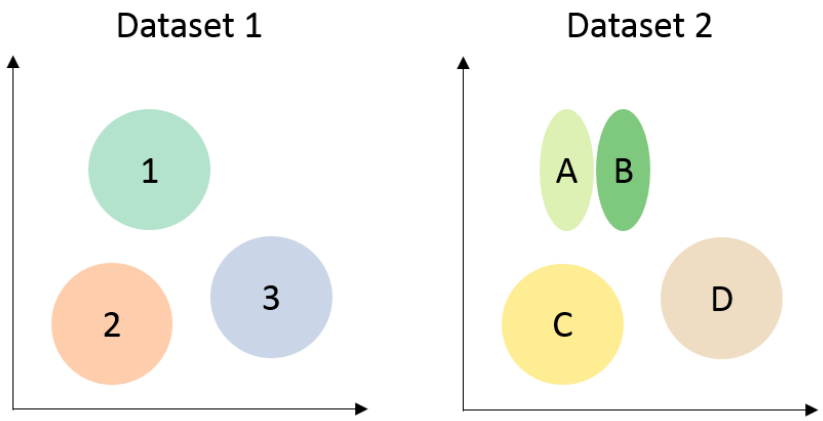
	1	2	root
A	**2**	0	0
B	0	**2**	0
C	0	0	**1**
root2	0	0	0

(b)



(c)

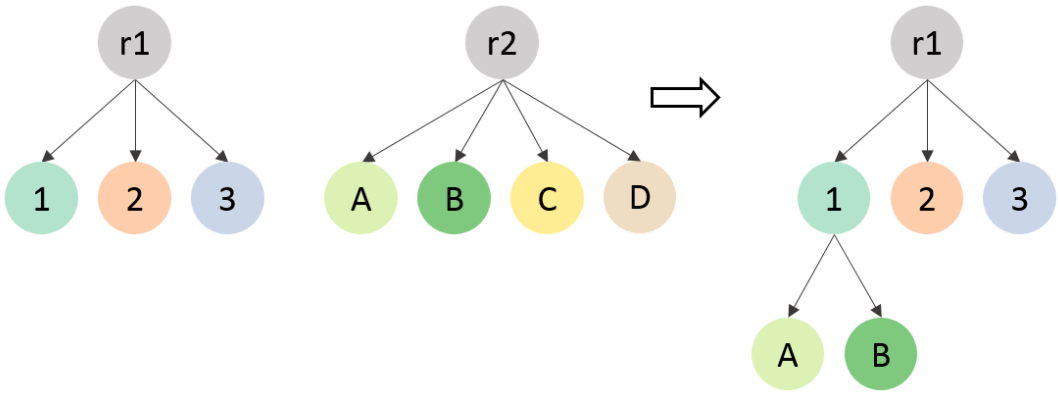
Figure S12: Simple, scenario 3



(a)

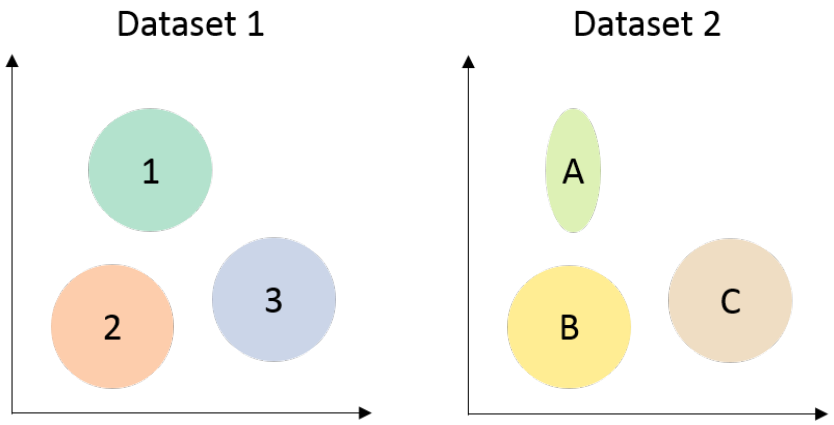
	1	2	3	root1
A	**2**	0	0	0
B	**2**	0	0	0
C	0	**2**	0	0
D	0	0	**2**	0
root2	0	0	0	0

(b)



(c)

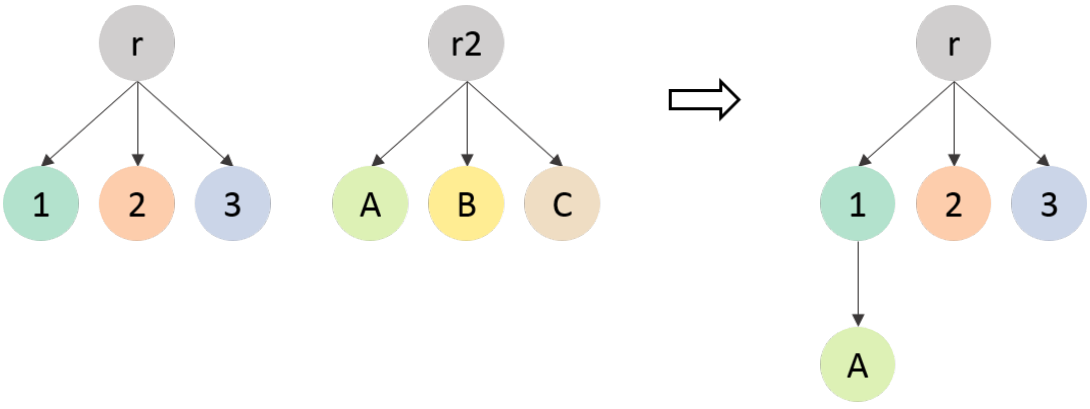
Figure S13: Multiple rows, scenario 1



(a)

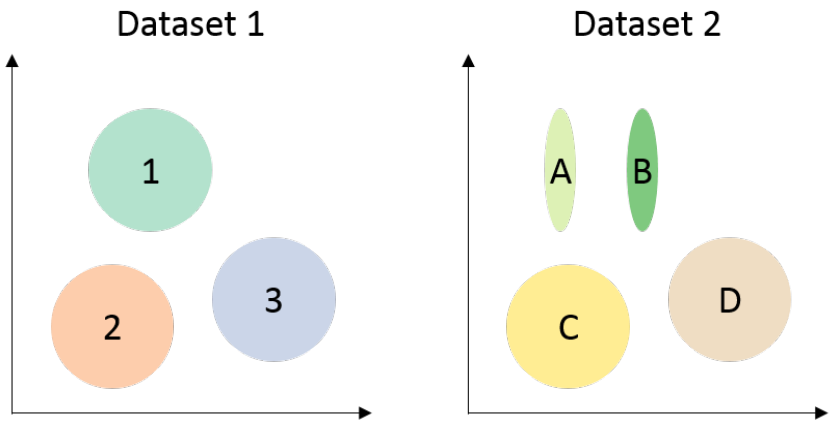
	1	2	3	root
A	**2**	0	0	0
B	0	**2**	0	0
C	0	0	**2**	0
root2	**1**	0	0	0

(b)



(c)

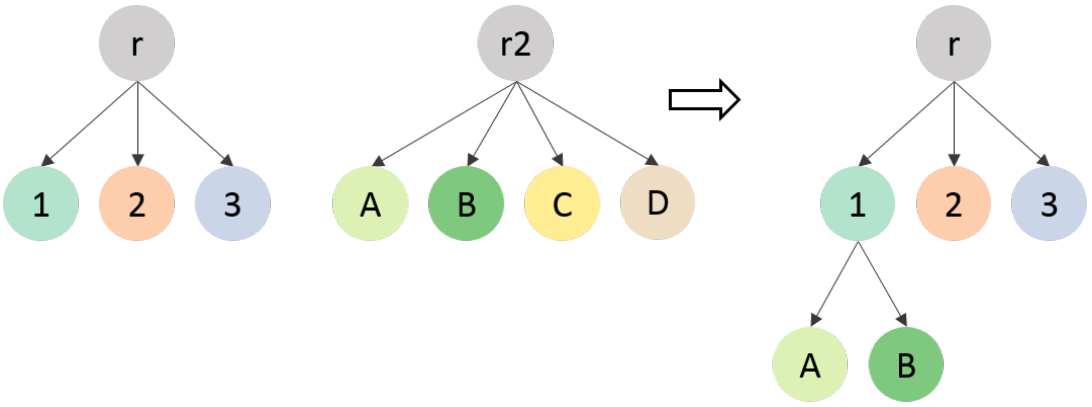
Figure S14: Multiple rows, scenario 2



(a)

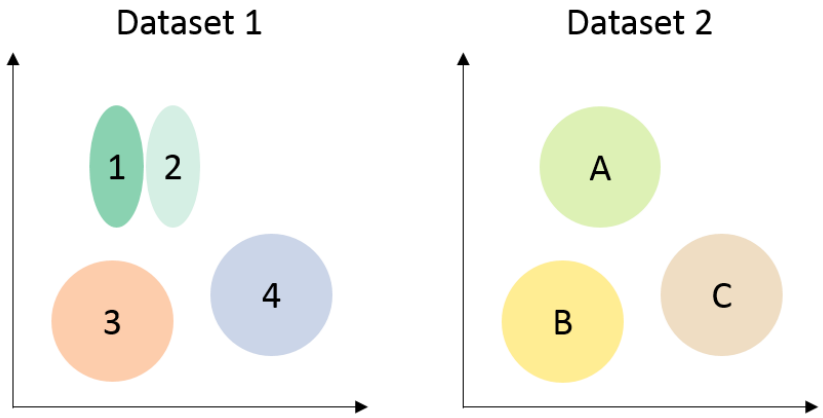
	1	2	3	root
A	**2**	0	0	0
B	**2**	0	0	0
C	0	**2**	0	0
D	0	0	**2**	0
root2	**1**	0	0	0

(b)



(c)

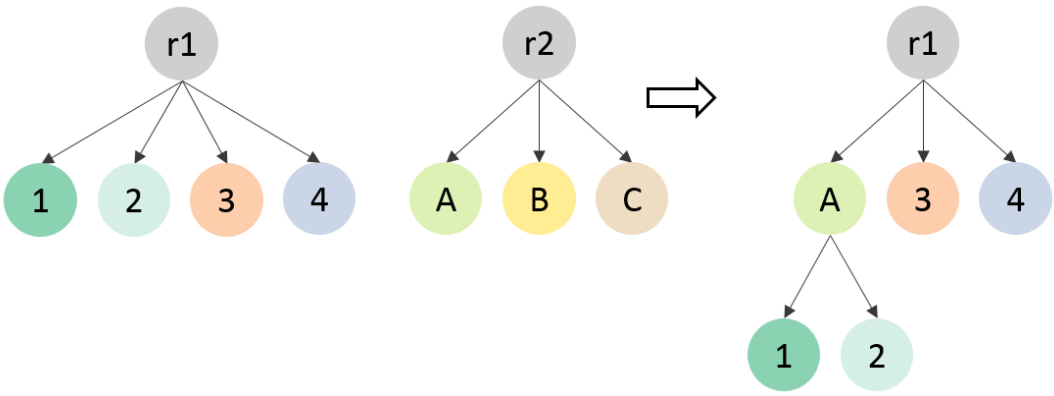
Figure S15: Multiple rows, scenario 3



(a)

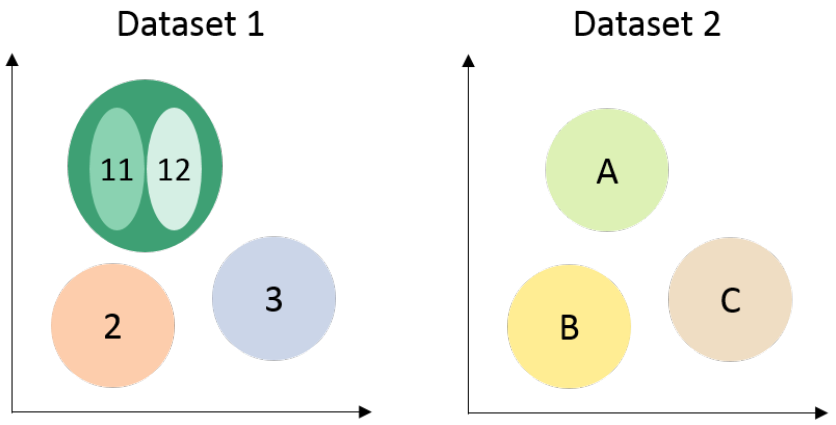
	1	2	3	4	root1
A	**2**	**2**	0	0	0
B	0	0	**2**	0	0
C	0	0	0	**2**	0
root2	0	0	0	0	0

(b)



(c)

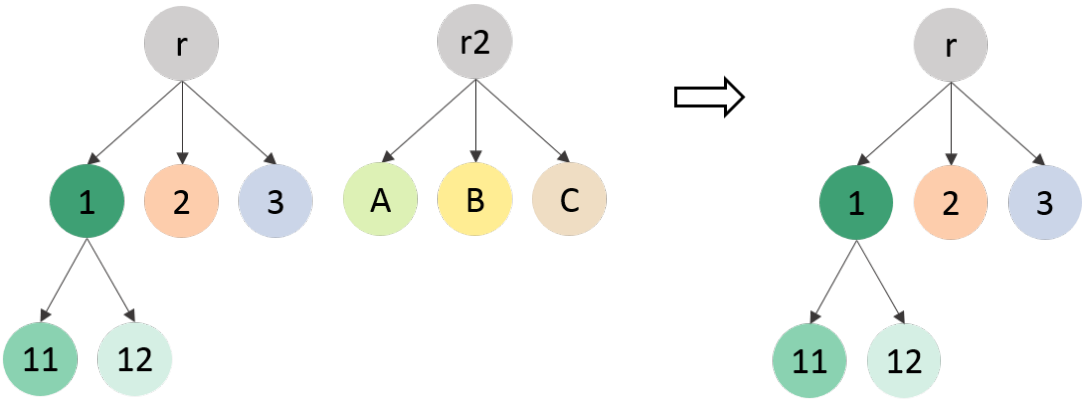
Figure S16: Multiple columns, scenario 1



(a)

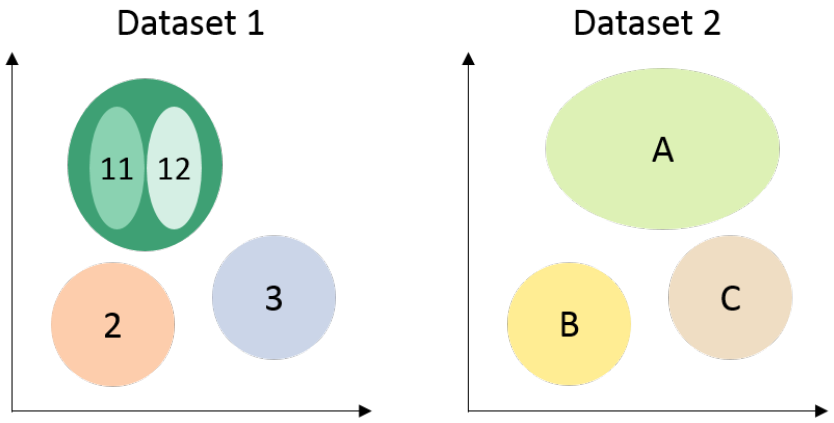
	11	12	1	2	3	root
A	**2**	**2**	**1**	0	0	0
B	0	0	0	**2**	0	0
C	0	0	0	0	**2**	0
root2	0	0	0	0	0	0

(b)



(c)

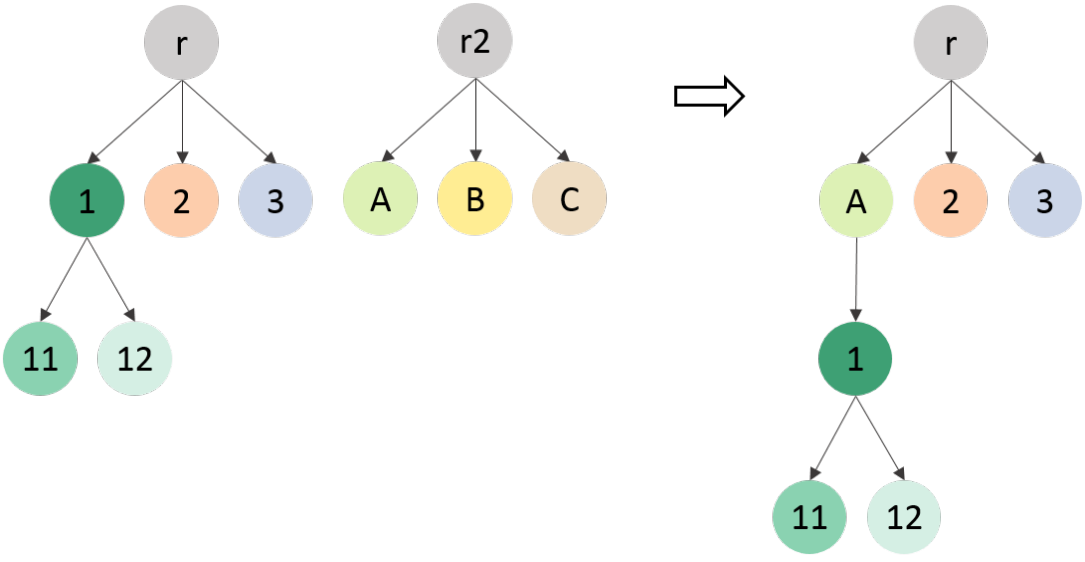
Figure S17: Multiple columns, scenario 2



(a)

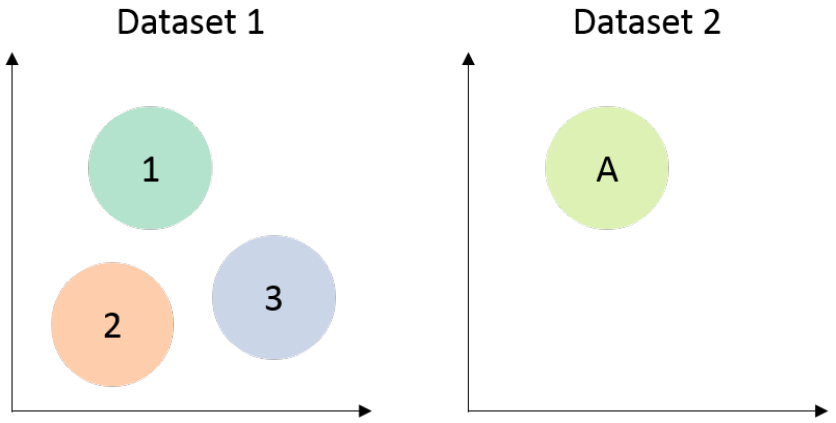
	11	12	1	2	3	root
A	**2**	**2**	**1**	0	0	**1**
B	0	0	0	**2**	0	0
C	0	0	0	0	**2**	0
root2	0	0	0	0	0	0

(b)



(c)

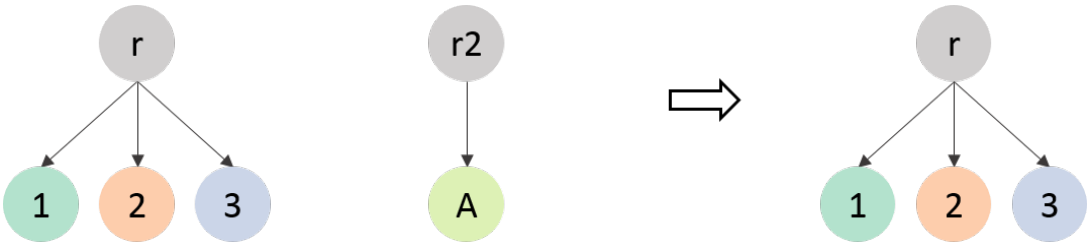
Figure S18: Multiple columns, scenario 3



(a)

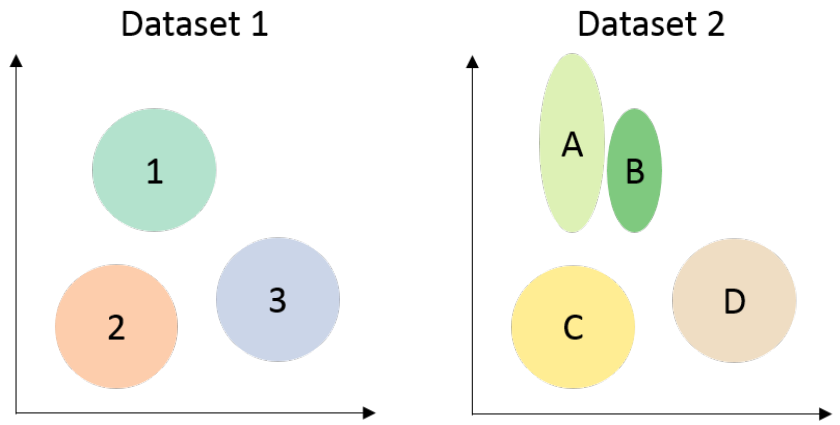
	1	2	3	root
A	**2**	0	0	0
B	0	0	0	0
root2	0	**1**	**1**	0

(b)



(c)

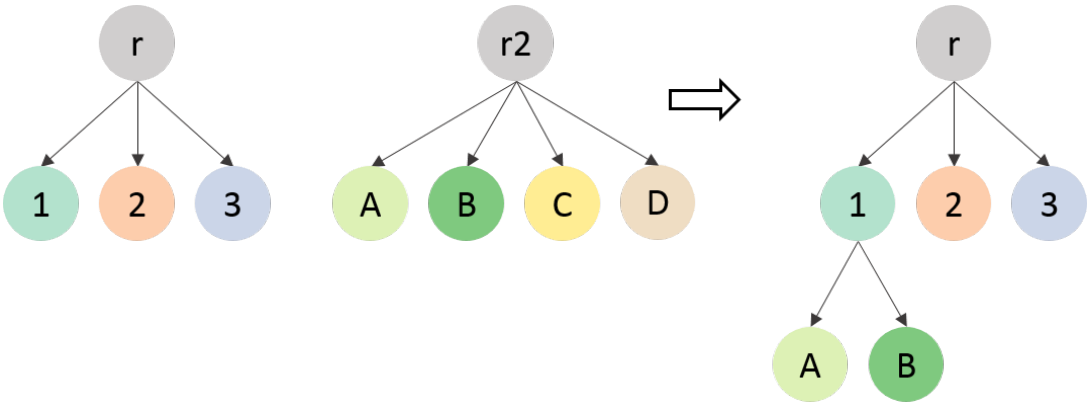
Figure S19: Multiple columns, scenario 4



(a)

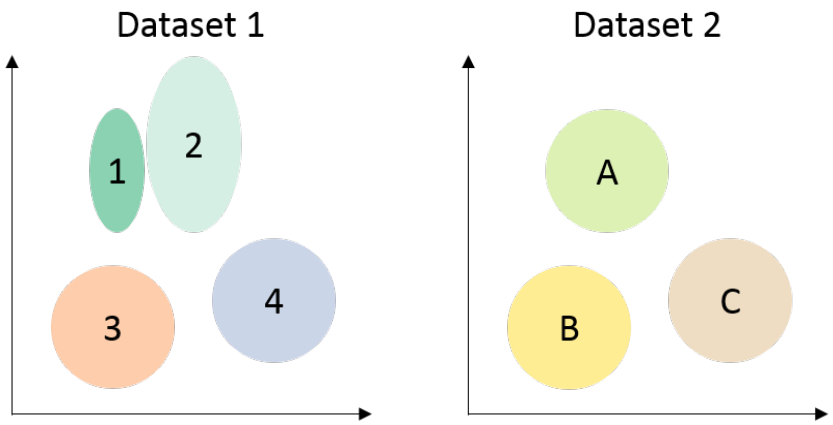
	1	2	3	root
A	**2**	0	0	**1**
B	**2**	0	0	0
C	0	**2**	0	0
D	0	0	**2**	0
root2	0	0	0	0

(b)



(c)

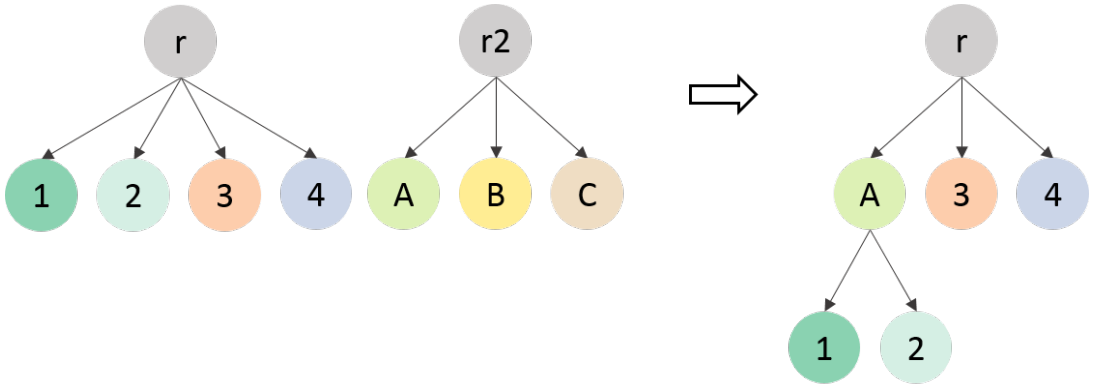
Figure S20: Complex, scenario 1



(a)

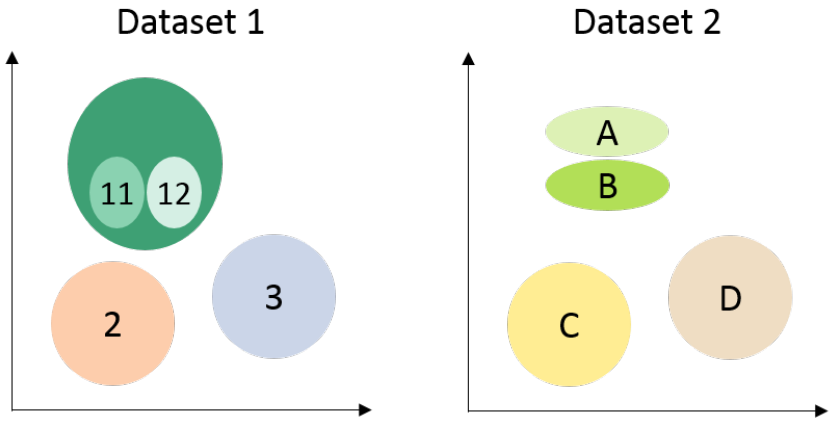
	1	2	3	4	root
A	**2**	**2**	0	0	0
B	0	0	**2**	0	0
C	0	0	0	**2**	0
root2	0	**1**	0	0	0

(b)



(c)

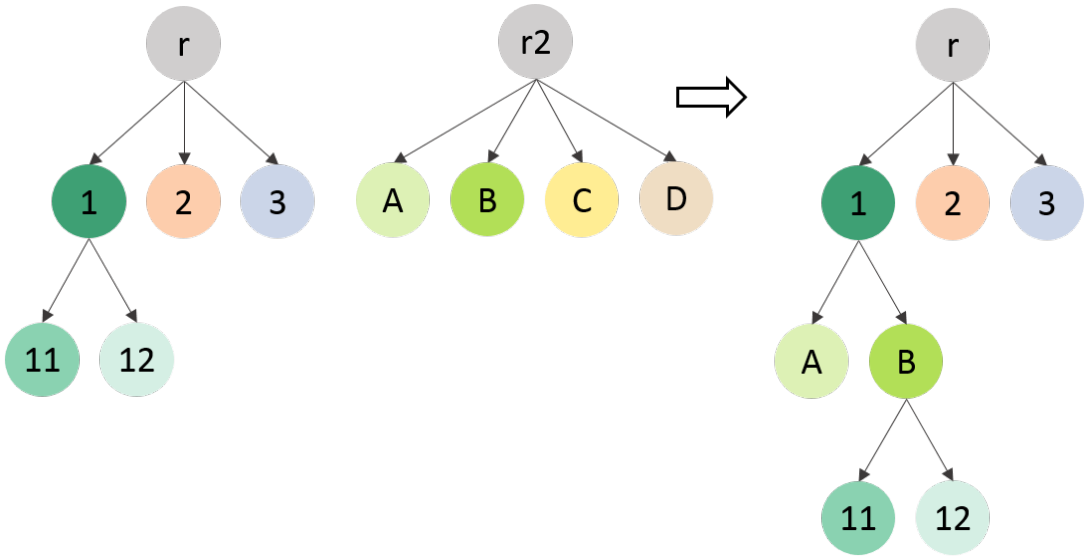
Figure S21: Complex, scenario 2



(a)

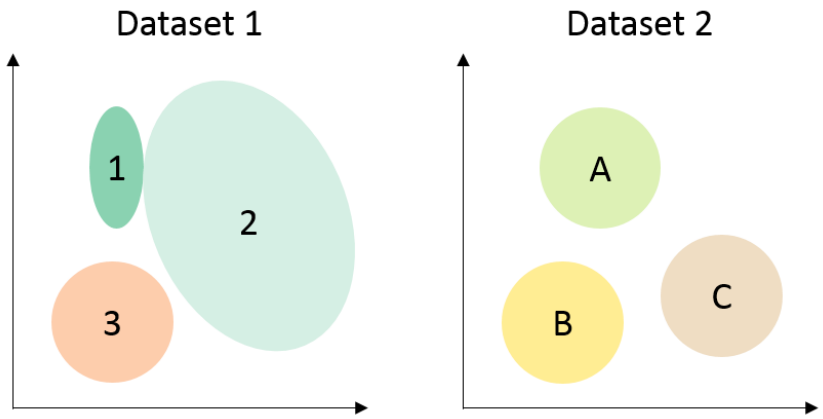
	11	12	1	2	3	root
A	0	0	**1**	0	0	0
B	**2**	**2**	**1**	0	0	0
C	0	0	0	**2**	0	0
D	0	0	0	0	**2**	0
root2	0	0	0	0	0	0

(b)



(c)

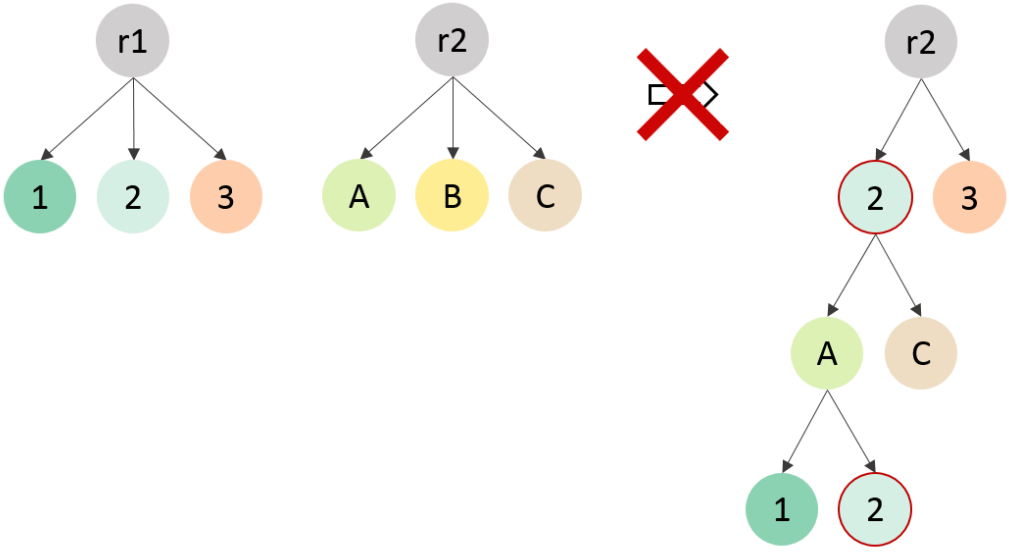
Figure S22: Complex, scenario 3



(a)

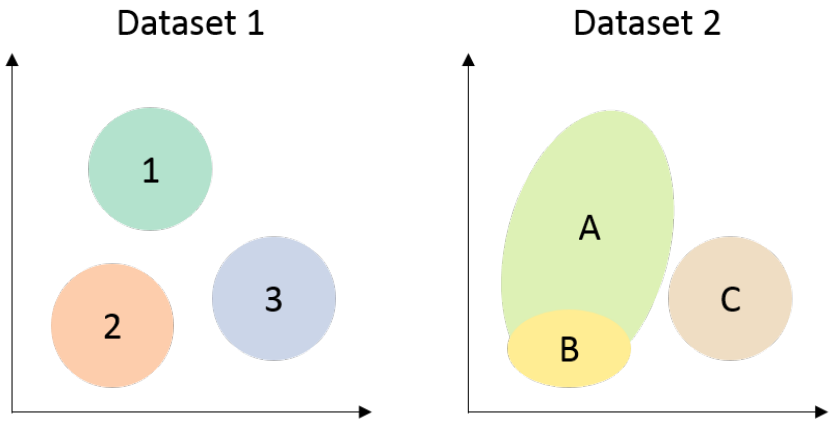
	1	2	3	root1
A	**2**	**2**	0	0
B	0	0	**2**	0
C	0	**2**	0	0
root2	0	0	0	0

(b)



(c)

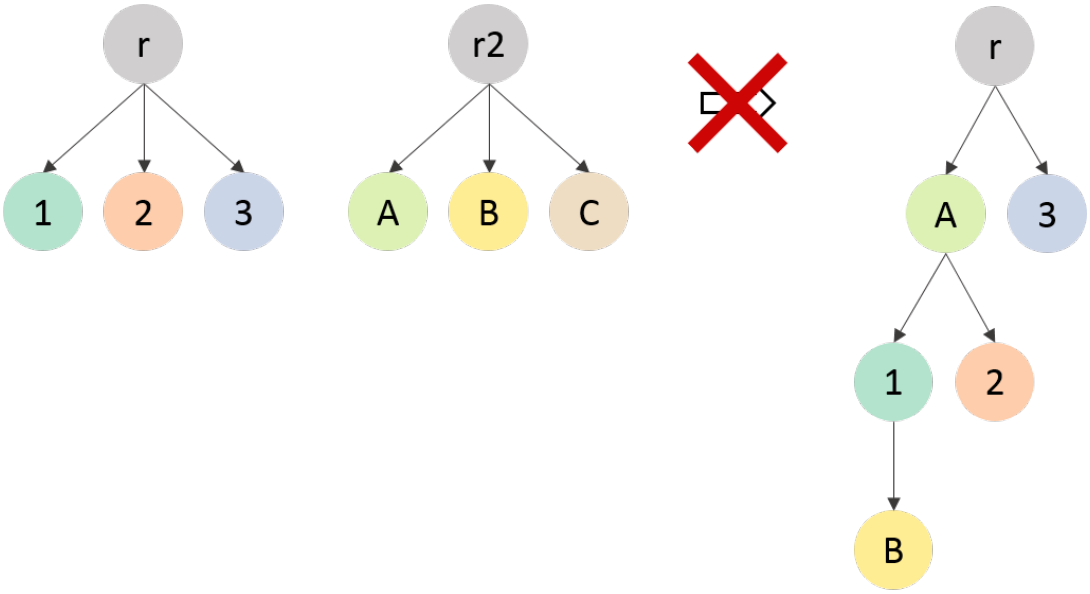
Figure S23: Impossible, scenario 1



(a)

	1	2	3	root
A	**2**	**2**	0	0
B	0	**2**	0	0
C	0	0	**2**	0
root2	0	0	0	0

(b)



(c)

Figure S24: Impossible, scenario 2

Supplementary Tables

Table S1: Confusion matrix of the one-class SVM with default offset on the simulated data.

	GROUP 1	GROUP 2	GROUP 12	GROUP 3	GROUP 4	GROUP 5	GROUP 6	GROUP 56	GROUP 456	ROOT
GROUP 1	1409	0	21	0	0	0	0	0	0	80
GROUP 2	0	1323	43	0	0	0	0	0	0	57
GROUP 3	0	0	0	1401	0	0	0	0	0	71
GROUP 4	0	0	0	0	1340	0	0	0	10	75
GROUP 5	0	0	0	0	0	1430	0	55	10	33
GROUP 6	0	0	0	0	0	0	1370	60	4	47

Table S2: Confusion matrix of the one-class SVM with minimum offset on the simulated data.

	GROUP 1	GROUP 2	GROUP 12	GROUP 3	GROUP 4	GROUP 5	GROUP 6	GROUP 56	GROUP 456	ROOT
GROUP 1	1510	0	0	0	0	0	0	0	0	0
GROUP 2	0	1423	0	0	0	0	0	0	0	0
GROUP 3	0	0	0	1471	0	0	0	0	0	1
GROUP 4	0	0	0	0	1425	0	0	0	0	0
GROUP 5	0	0	0	0	0	1526	0	1	1	0
GROUP 6	0	0	0	0	0	0	1480	1	0	0

Table S3: Confusion matrix of the one-class SVM with default offset on the PBMC data.

	CD34+	CD14+ MONOCYTES	CD56+ NK CELLS	CD19+ B CELLS	CD4+ T CELLS	CD4+/CD25 T REG	CD4+/CD45RA+/ CD25- NAIVE T	CD4+/CD45RO+ MEMORY	CD8+ CYTOTOXIC T	CD8+/CD45RA+ NAIVE CYTOTOXIC	T CELLS	SMALL LYMPHO-CYTES	LYMPHO-CYTES	ROOT
CD34+	1867	0	3	26	0	0	0	0	3	0	0	9	10	82
CD14+ MONOCYTES	5	1904	0	0	1	5	0	0	0	0	1	3	3	78
CD56+ NK CELLS	3	1	1760	0	0	0	0	0	4	0	1	1	14	216
CD19+ B CELLS	11	0	0	1773	0	0	0	0	0	0	0	15	27	174
CD4+ T CELLS	1	1	0	0	29	815	1039	18	0	0	7	27	13	50
CD4+/CD25 T REG	0	2	0	0	11	1388	430	12	0	0	2	24	26	105
CD4+/CD45RA+/ CD25- NAIVE T	5	1	0	0	27	60	1824	1	0	0	10	27	20	25
CD4+/ CD45RO+ MEMORY	7	0	0	0	8	36	4	1667	68	73	23	35	16	63
CD8+ CYTOTOXIC T	0	0	11	0	17	20	5	138	718	914	19	33	66	59
CD8+/CD45RA+ NAIVE CYTOTOXIC	0	0	0	1	22	4	125	299	34	1459	6	9	11	30

Table S4: Confusion matrix of the one-class SVM with minimum offset on the PBMC data.

	CD34+	CD14+ MONOCYTES	CD56+ NK CELLS	CD19+ B CELLS	CD4+ T CELLS	CD4+/CD25 T REG	CD4+/CD45RA+/ CD25- NAIVE T	CD4+/CD45RO+ MEMORY	CD8+ CYTOTOXIC T	CD8+/CD45RA+ NAIVE CYTOTOXIC	T CELLS	SMALL LYMPHO-CYTES	LYMPHO-CYTES	ROOT
CD34+	1515	2	1	133	1	29	82	236	0	0	0	0	0	1
CD14+ MONOCYTES	9	685	0	113	1	39	8	1137	0	0	0	8	0	0
CD56+ NK CELLS	2	0	737	0	0	710	455	96	0	0	0	0	0	0
CD19+ B CELLS	1	0	0	1951	0	1	47	0	0	0	0	0	0	0
CD4+ T CELLS	0	2	0	0	0	271	1725	0	0	0	0	2	0	0
CD4+/CD25 T REG	0	1	0	0	0	692	1306	0	0	0	0	0	0	1
CD4+/CD45RA+/ CD25- NAIVE T	1	1	0	0	0	12	1985	1	0	0	0	0	0	0
CD4+/ CD45RO+ MEMORY	9	0	0	0	0	206	1296	489	0	0	0	0	0	0
CD8+ CYTOTOXIC T	1	0	0	0	0	259	1593	137	1	9	0	0	0	0
CD8+/CD45RA+ NAIVE CYTOTOXIC	0	0	0	1	0	1	1988	9	0	1	0	0	0	0

Table S5: Confusion matrix of the linear SVM on the PBMC data. Here, the linear SVM was trained using the hematopoietic tree.

	CD34+	CD14+ MONOCYTES	CD56+ NK CELLS	CD19+ B CELLS	CD4+ T CELLS	CD4+/CD25 T REG	CD4+/CD45RA+/ CD25- NAIVE T	CD4+/CD45RO+ MEMORY	CD8+ CYTOTOXIC T	CD8+/CD45RA+ NAIVE CYTOTOXIC	T CELLS	SMALL LYMPHO-CYTES	LYMPHO-CYTES	ROOT
CD34+	1964	1	1	26	0	0	0	0	6	0	0	0	0	2
CD14+ MONOCYTES	0	1979	1	1	2	15	0	1	0	0	0	0	0	1
CD56+ NK CELLS	2	0	1990	0	0	0	0	1	6	0	0	0	0	1
CD19+ B CELLS	0	0	0	1999	0	0	0	1	0	0	0	0	0	0
CD4+ T CELLS	0	1	0	0	191	1012	790	3	0	2	0	0	0	1
CD4+/CD25 T REG	0	0	0	0	113	1707	175	4	0	0	0	0	0	1
CD4+/CD45RA+/ CD25- NAIVE T	0	1	0	1	94	140	1751	1	1	9	0	0	0	2
CD4+/ CD45RO+ MEMORY	3	0	0	0	1	33	14	1890	32	26	0	0	0	1
CD8+ CYTOTOXIC T	0	0	0	0	1	5	0	44	1885	65	0	0	0	0
CD8+/CD45RA+ NAIVE CYTOTOXIC	0	0	0	1	3	0	13	36	31	1916	0	0	0	0

Table S6: Confusion matrix of the linear SVM on the PBMC data. Here, the linear SVM was trained on the altered tree. The CD4+ memory T-cells are a subpopulation of CD8+ T-cells now.

	CD34+	CD14+ MONOCYTES	CD56+ NK CELLS	CD19+ B CELLS	CD4+ T CELLS	CD4+/CD25 T REG	CD4+/CD45RA+/ CD25- NAIVE T	CD4+/CD45RO+ MEMORY	CD8+ CYTOTOXIC T	CD8+/CD45RA+ NAIVE CYTOTOXIC	T CELLS	SMALL LYMPHO-CYTES	LYMPHO-CYTES	ROOT
CD34+	1964	1	1	26	0	0	0	2	0	4	0	0	0	2
CD14+ MONOCYTES	0	1979	1	1	0	17	0	1	0	0	0	0	0	1
CD56+ NK CELLS	2	0	1990	0	0	0	0	7	0	0	0	0	0	1
CD19+ B CELLS	0	0	0	1999	0	0	0	0	0	1	0	0	0	0
CD4+ T CELLS	0	1	0	0	0	1118	875	5	0	0	0	0	0	1
CD4+/CD25 T REG	0	0	0	0	0	1797	200	2	0	0	0	0	0	1
CD4+/CD45RA+/ CD25- NAIVE T	0	1	0	1	0	133	1860	2	0	1	0	0	0	2
CD4+/ CD45RO+ MEMORY	3	0	0	0	0	4	0	1974	0	18	0	0	0	1
CD8+ CYTOTOXIC T	0	0	0	0	0	2	0	775	0	1223	0	0	0	0
CD8+/CD45RA+ NAIVE CYTOTOXIC	0	0	0	1	0	0	3	24	0	1972	0	0	0	0

Table S7: Labels of the simulated dataset when testing tree construction with missing cell populations.

Original cell population	Label Batch 1	Label Batch 2	Label Batch 3
Group1	Group12	Group1	Group1
Group2	Group12	Group2	Group2
Group3	Group3	Group3	Group3
Group4	Group456	Group4	Group4
Group5	Group456	-	Group5
Group6	Group456	Group6	Group6