chapter 3

Hierarchical progressive learning of cell identities in single-cell data

Lieke Michielsen, Marcel J.T. Reinders, Ahmed Mahfouz

This chapter is published in: *Nature Communications* (2021) 12: 2799, doi: 10.1038/s41467-021-23196-8.
Supplementary material is available online at:
https://www.nature.com/articles/s41467-021-23196-8#Sec27

Supervised methods are increasingly used to identify cell populations in single-cell data. Yet, current methods are limited in their ability to learn from multiple datasets simultaneously, are hampered by the annotation of datasets at different resolutions, and do not preserve annotations when retrained on new datasets. The latter point is especially important as researchers cannot rely on downstream analysis performed using earlier versions of the dataset. Here, we present scHPL, a hierarchical progressive learning method which allows continuous learning from single-cell data by leveraging the different resolutions of annotations across multiple datasets to learn and continuously update a classification tree. We evaluate the classification and tree learning performance using simulated as well as real datasets and show that scHPL can successfully learn known cellular hierarchies from multiple datasets while preserving the original annotations. scHPL is available at https://github.com/lcmmichielsen/scHPL.

3.1 Introduction

Cell identification is an essential step in single-cell studies with profound effects on downstream analysis. For example, in order to compare cell-population-specific eQTL findings across studies, cell identities should be consistent [1]. Currently, cells in single-cell RNA-sequencing (scRNA-seq) datasets are primarily annotated using clustering and visual exploration techniques, i.e. cells are first clustered into populations which are subsequently named based on the expression of marker genes. This is not only time-consuming, but also subjective [2]. The number of cell populations identified in a dataset, for example, is strongly correlated with the number of cells analyzed, which results in inconsistency across datasets [3–5].

Recently, many supervised methods have been developed to replace unsupervised techniques. The underlying principles of these methods vary greatly. Some methods, for instance, rely on prior knowledge and assume that for each cell population marker genes can be defined (e.g. SCINA [6] and Garnett [7]), while others search for similar cells in a reference database (e.g. scmap [8] and Cell-BLAST [9]), or train a classifier using a reference atlas or a labeled dataset (e.g. scPred [10] and CHETAH [11]).

Supervised methods rely either on a reference atlas or labeled dataset. Ideally, we would use a reference atlas containing all possible cell populations to train a classifier. Such an atlas, however, does not exist yet and might never be fully complete. In particular, aberrant cell populations might be missing as a huge number of diseases exist and mutations could result in new cell populations. To overcome these limitations, some methods (e.g. OnClass) rely on the Cell Ontology to identify cell populations that are missing from the training data but do exist in the Cell Ontology database [12]. These Cell Ontologies, however, were not developed for scRNA-seq data specifically. As a consequence, many newly identified (sub) populations are missing and relationships between cell populations might be inaccurate. A striking example of this inadequacy are neuronal cell populations. Recent single-cell studies have identified hundreds of populations [4,13,14], including seven subtypes and 92 cell populations in one study only [5]. In contrast, the Cell Ontology currently includes only one glutamatergic neuronal cell population without any subtypes.

Since no complete reference atlas is available, a classifier should ideally be able to combine the information of multiple annotated 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. We advocate that this can be realized with progressive learning, a learning strategy inspired by humans. Human learning is a continuous process that never ends [15]. Using progressive learning, the task complexity is gradually increased, for instance, by adding more classes, but it is essential that the knowledge of the previous classes is preserved [16,17]. This strategy allows combining information of multiple existing datasets and retaining the possibility to add more datasets afterwards. However, it cannot be simply applied to scRNA-seq datasets as a constant terminology to describe cell populations is missing, which eliminates straightforward identification of new cell populations based on their names. For example, the recently discovered neuronal populations are typically identified using clustering and named based on the expression of marker genes. A standardized nomenclature for these clusters is missing [18], so the relationship between cell populations defined in different datasets is often unknown.

Moreover, the level of detail (resolution) at which datasets are annotated highly depends on the number of cells analyzed [19]. For instance, if a dataset is annotated at a low resolution, it might contain T-cells, while a dataset at a higher resolution can include subpopulations of T-cells, such as CD4+ and CD8+ T-cells. We need to consider this hierarchy of cell populations in our representation, which can be done with a hierarchical classifier. This has the advantage that cell population definitions of multiple datasets can be combined, ensuring consistency. A hierarchical classifier has additional advantages in comparison to a classifier that does not exploit this hierarchy between classes (here denoted as 'flat classifier'). One of these advantages is that the classification problem is divided into smaller sub-problems, while a flat classifier needs to distinguish between many classes simultaneously. Another advantage is that if we are not sure about the annotation of an unlabeled cell at the highest resolution, we can always label it as an intermediate cell population (i.e. at a lower resolution).

Currently, some classifiers, such as Garnett, CHETAH, and Moana, already exploit this hierarchy between classes [7,11,20]. Garnett and Moana both depend on prior knowledge in the form of marker genes for the different classes. Especially for deeper annotated datasets it can be difficult to define marker genes for each cell population that are robust across scRNA-seq datasets [21,22]. Moreover, we have previously shown that adding prior knowledge is not beneficial [23]. CHETAH, on the contrary, constructs a classification tree based on one dataset by hierarchically clustering the reference profiles of the cell populations and classifies new cells based on the similarity to the reference profile of that cell population. However, simple flat classifiers outperform CHETAH [23], indicating that a successful strategy to exploit this hierarchy is still missing. Furthermore, these hierarchical classifiers cannot exploit the different resolutions of multiple datasets as this requires adaptation of the hierarchical representation.

Even if multiple datasets are combined into a hierarchy, there might be unseen populations in an unlabeled dataset. Identifying these cells as a new population is a challenging problem. Although some classifiers have implemented an option to reject cells, they usually fail when being tested in a realistic scenario [23]. In most cases, the rejection option is implemented by setting a threshold on the posterior probability [7,10,23,24]. If the highest posterior probability does not exceed a threshold, the cell is rejected. By looking at the posterior, the actual similarity between a cell and the cell population is ignored.

In this work, we propose a hierarchical progressive learning approach to overcome these challenges. To summarize our contributions: (i) we exploit the hierarchical relationships between cell populations to be able to classify data sets at different resolutions, (ii) we propose a progressive learning approach that updates the hierarchical relationships dynamically and consistently, and (iii) we adopt an advanced rejection procedure including a one-class classifier to be able to discover new cell (sub)populations.

3.2 Results

3.2.1 Hierarchical progressive learning of cell identities

We developed scHPL, a hierarchical progressive learning approach to learn a classification tree using multiple labeled datasets (Figure 1A) and use this tree to predict the labels of a new, unlabeled dataset (Figure 1B). The tree is learned using multiple iterations (Methods). First, we match the labels of two datasets by training a flat classifier for each dataset and predicting the labels of the other dataset. Based on these predictions we create a matching matrix (*X*)

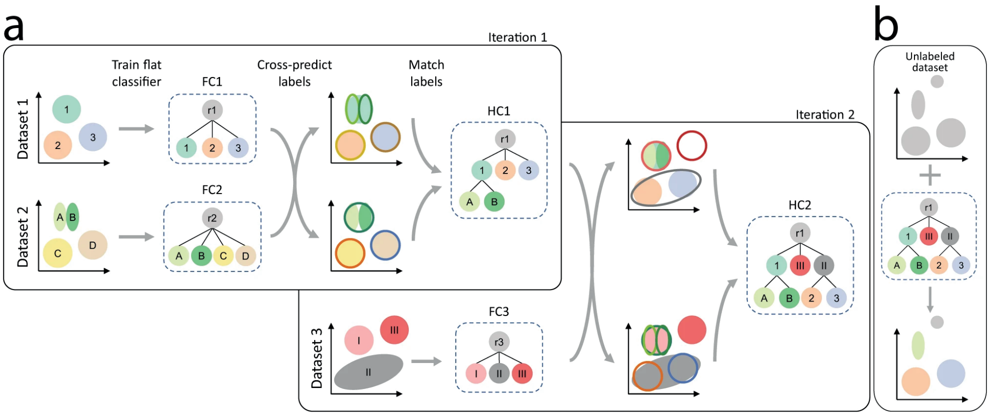


Figure 1. Schematic overview of scHPL. A) Overview of the training phase. In the first iteration, we start with two labeled datasets. The colored areas represent the different cell populations. For both datasets a flat classifier (FC1 & FC2) is constructed. Using this tree and the corresponding dataset, a classifier is trained for each node in the tree except for the root. We use the trained classification tree of one dataset to predict the labels of the other. The decision boundaries of the classifiers are indicated with the contour lines. We compare the predicted labels to the cluster labels to find matches between the labels of the two datasets. The tree belonging to the first dataset is updated according to these matches, which results in a hierarchical classifier (HC1). In dataset 2, for example, subpopulations of population '1' of dataset 1 are found. Therefore, these cell populations, 'A' and 'B', are added as children to the '1' population. In iteration 2, a new labeled dataset is added. Again a flat classifier (FC3) is trained for this dataset and HC1 is trained on dataset 1 and 2, combined. After cross-prediction and matching the labels, we update the tree which is then trained on all datasets 1-3 (HC2). **B)** The final classifier can be used to annotate a new unlabeled dataset. If this dataset contains unknown cell populations, these will be rejected.

and match the cell populations of the two datasets. In the matching process, we separate different biological scenarios, such as a perfect match, merging or splitting cell populations, as well as creating a new population (Figure 2, Table S1). In the following iterations, we add one labeled dataset at a time by training a flat classifier on this new dataset and training the previously learned tree on all pre-existing datasets. Similar to the previous iteration, the tree is updated after cross-prediction and matching of the labels. It could happen that datasets are inconsistently labeled and the labels cannot be matched (Supplementary Note 1). In this case, one of the populations might be missing from the tree.

Either during tree learning or prediction, there can be unseen populations. Therefore, an efficient rejection option is needed, which we do in two steps. First, we reject cells by thresholding the reconstruction error of a cell when applying a PCA-based dimension reduction: a new, unknown, population is not used to learn the PCA transformation, and consequently will not be properly represented by the selected PCs, leading to a high reconstruction error (Methods). Second, to accommodate rejections when the new population is within the selected PCA domain, scHPL adopts two alternatives to classify cells: a linear and a one-class support vector machine (SVM). The linear SVM has shown a high performance in a benchmark of scRNA-seq classifiers [23], but has a limited rejection option. Whereas, the one-class SVM solves this as only positive training samples are used to fit a tight decision boundary around [25].

3.2.2 Linear SVM has a higher classification accuracy than one-class SVM

We tested our hierarchical classification scheme by measuring the classification performance of the one-class SVM and linear SVM on simulated, PBMC (PBMC-FACS) and brain (Allen Mouse Brain) data using 10-, 10-, and 5-fold cross-validation respectively (Methods). The

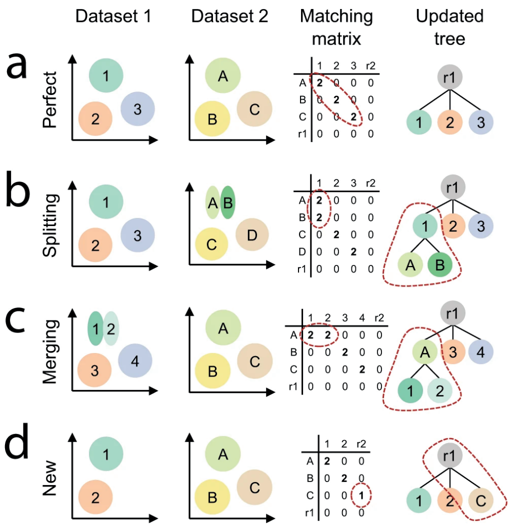


Figure 2. Schematic examples of different matching scenarios. A) Perfect match, **B)** splitting, **C)** merging, **D)** new population. The first two columns represent a schematic representation of two datasets. After cross-predictions, the matching matrix *(X)* is constructed using the confusion matrices (Methods). We update the tree based on *X*.

simulated dataset was constructed using Splatter [26] and consists of 8,839 cells, 9,000 genes and 6 different cell populations (Figure S1). PBMC-FACS is the downsampled FACS-sorted PBMC dataset from Zheng et al. [27] and consists of 20,000 cells and 10 cell populations. The Allen Mouse Brain (AMB) dataset is challenging as it has deep annotation levels [5], containing 92 different cell populations ranging in size from 11 to 1,348 cells. In these experiments, the classifiers were trained on predefined trees (Figure S1-3).

On all datasets, the linear SVM performs better than the one-class SVM (Figure 3A-D). The simulated dataset is relatively easy since the cell populations are widely separated (Figure S1C). The linear SVM shows an almost perfect performance: only 0.9% of the cells are rejected (based on the reconstruction error only), which is in line with the adopted threshold resulting in 1% false negatives. The one-class SVM labels 92.9% of the cells correctly, the rest is labeled as an internal node (2.3%) or rejected (4.8%), which results in a median Hierarchical F1-score (HF1-score) of 0.973, where HF1 is an F1-score that considers class importance across the hierarchy (Methods).

As expected, the performance of the classifiers on real data drops, but the HF1-scores remain higher than 0.9. On both the PBMC-FACS and AMB dataset, the performance of the linear

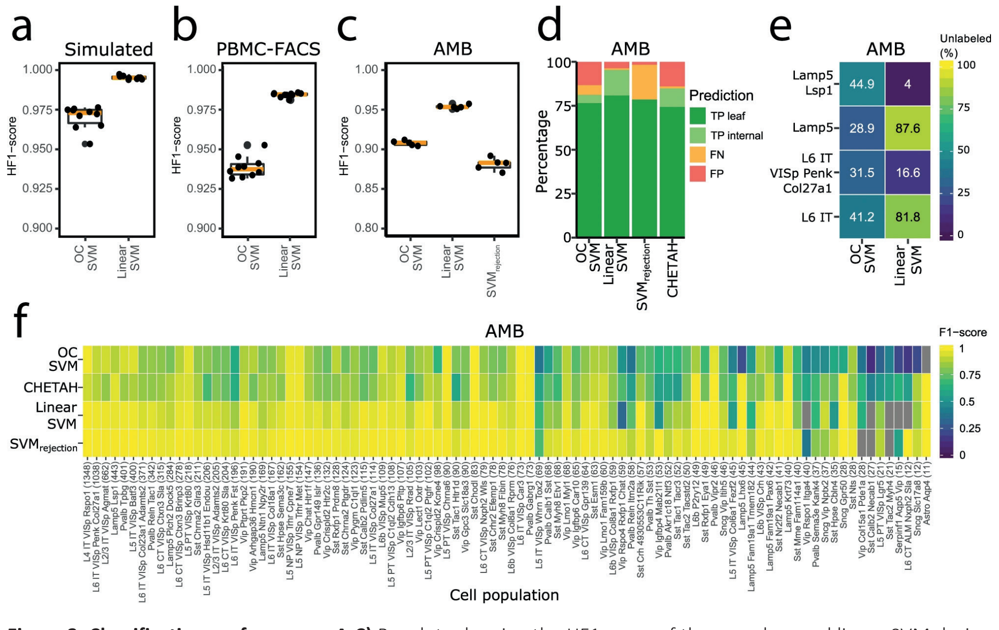


Figure 3. Classification performance. A-C) Boxplots showing the HF1-score of the one-class and linear SVM during *n*-fold cross-validation on the **A)** simulated (*n* = 10), **B)** PBMC-FACS (*n* = 10), and **C)** AMB (*n* = 5) dataset. In the boxplots, the middle (orange) line represents the median, the lower and upper hinge represent the first and third quartiles, and the lower and upper whisker represent the values no further than 1.5 inter-quartile range away from the lower and upper hinge respectively. **D)** Barplot showing the percentage of true positives (TP), false negatives (FN), and false positives (FP) per classifier on the AMB dataset. For the TPs a distinction is made between correctly predicted leaf nodes and internal nodes. **E)** Heatmap showing the percentage of unlabeled cells per classifier during the different rejection experiments. **F)** Heatmap showing the F1-score per classifier per cell population on the AMB dataset. Grey indicates that a classifier never predicted a cell to be of that population.

SVM is higher than the one-class SVM (Figure 3B-D). For the AMB dataset, we used the same cross-validation folds as in Abdelaal et al. [23], which enables us to compare our results. When comparing to CHETAH, which allows hierarchical classification, we notice that the linear SVM performs better based on the median F1-score (0.94 vs 0.83). The one-class SVM has a slightly lower median F1-score (0.80 vs 0.83), but it has more correctly predicted cells and less wrongly predicted cells (Figure 3D).

The linear (hierarchical) SVM also shows a better performance compared to $\mathrm{SVM}_{\mathrm{rejection}}$, which is a flat linear SVM with rejection option based on the posterior probability and was the best classifier for this data [23]. $\mathrm{SVM}_{\mathrm{rejection}}$, however, has a slightly higher median F1-score (0.98 vs 0.94). This is mainly because it makes almost no mistakes, only 1.7% of the cells are wrongly labeled (Figure 3D). The number of rejected cells, on the other hand, is not considered when calculating the median F1-score. This number is relatively high for $\mathrm{SVM}_{\mathrm{rejection}}$ (19.8%). The linear SVM, on the contrary, has almost no rejected cells, which is also reflected in a higher HF1-score (Figure 3C). Because of this large amount of rejections of $\mathrm{SVM}_{\mathrm{rejection}}$, the one-class SVM also has a higher HF1-score.

On the AMB dataset, we observe that the performance of all classifiers decreases when the number of cells per cell population becomes smaller. The performance of the one-class SVM is affected more than the others (Figure 3F). The one-class SVM, for instance, 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 cell population, however, only consists of eleven cells.

In the previous experiments, we used all genes to train the classifiers. Since the selection of highly variable genes (HVGs) is common in scRNA-seq analysis pipelines, we tested the effect of selecting HVGs on the classification performance of the PBMC-FACS dataset. We noted that using all genes results in the highest HF1-score for both the linear and one-class SVM (Figure S4).

3.2.3 One-class SVM detects new cells better than linear SVM

Besides a high accuracy, the classifiers should be able to reject unseen cell populations. 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. We removed one cell population, 'Group 3', from the training set and used this population as a test set. The one-class SVM outperforms the linear SVM as it correctly rejects all these cells, while the linear SVM rejects only 38.9% of them.

Next, we tested the rejection option on the AMB dataset. Here, we did four experiments and each time removed a node, including all its subpopulations, from the predefined tree (Figure S3). We removed the 'L6 IT' and 'Lamp5' cell populations from the second layer of the tree, and the 'L6 IT VISp Penk Col27a1' and 'Lamp5 Lsp1' from the third layer. When a node is removed from the second layer of the tree, the linear SVM clearly rejects these cells better than the one-class SVM (Figure 3E). On the contrary, the one-class SVM rejects leaf node cells better.

3.2.4 scHPL accurately learns cellular hierarchies

Next, we tested if we could learn the classification trees for the simulated and PBMC-FACS data using scHPL. In both experiments, we performed a 10-fold cross-validation and splitted the training set in three different batches, Batch 1, Batch 2, and Batch 3, to simulate the idea of different datasets. To obtain different annotation levels in these batches, multiple cell populations were merged into one population in some batches, or cell populations were removed from certain batches (Tables S2-3). Batch 1 contains the lowest resolution and Batch 3 the highest. When learning the trees, we try all (six) different orders of the batches to see whether this affects the tree learning. Combining this with the 10-fold cross-validation, 60 trees were learned in total by each classifier. To summarize the results, we constructed a final tree in which the thickness of an edge indicates how often it appeared in the 60 learned trees.

The linear and one-class SVM showed stable results during both experiments; all 60 trees - except for two trees learned by the one-class SVM on the PBMC data - look identical (Figure 4A-D). The final tree for the simulated data looks as expected, but the tree for the PBMC data looks slightly different from the predefined hematopoietic tree (Figure S2A). In the learned trees, CD4+ memory T-cells are a subpopulation of CD8+ instead of CD4+ T-cells. The correlation between the centroids of CD4+ memory T-cell and CD8+ T-cells (r = 0.985±0.003) is also slightly higher than the correlation to CD4+ T-cells (r = 0.975±0.002) (Figure S5). Using the learned tree instead of the predefined hematopoietic tree improves the classification performance of the linear SVM slightly (HF1-score = 0.990 vs 0.985). Moreover, when relying

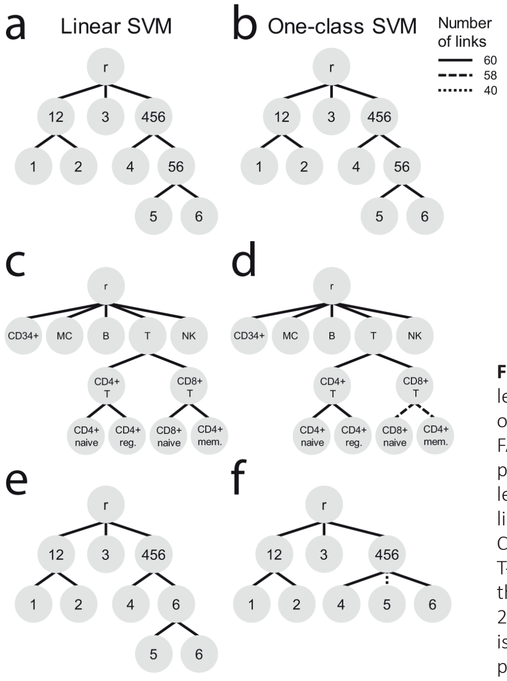


Figure 4. Tree learning evaluation. Classification trees learned when using a **A, C, E)** linear SVM or **B, D, F)** one-class SVM during the **A, B)** simulated, **C, D)** PBMC-FACS, and **E, F)** simulated rejection experiment. The line pattern of the links indicates how often that link was learned during the 60 training runs. **D)** In 2/60 trees, the link between the CD8+ T-cells and the CD8+ naive and CD4+ memory T-cells is missing. In those trees, the CD8+ T-cells and CD8+ naive T-cells have a perfect match and the CD4+ memory T-cells are missing from the tree. **F)** In 20/60 trees, the link between 'Group456' and 'Group5' is missing. In those trees, these two populations are a perfect match.

on the predefined hematopoietic tree, CD4+ memory T-cells, CD8+ T-cells, and CD8+ naive T-cells were also often confused, further highlighting the difficulty in distinguishing these populations based on their transcriptomic profiles alone (Tables S4-5).

Next, we tested the effect of the matching threshold (default = 0.25) on the tree construction by varying this to 0.1 and 0.5. For the linear SVM, changing the threshold had no effect. For the one-class SVM, we noticed a small difference when changing the threshold to 0.1. The two trees that were different using the default threshold (Figure 4D), were now constructed as the other 58 trees. In general, scHPL is robust to settings of the matching threshold due to its reliance on reciprocal classification.

3.2.5 Missing populations affect tree construction with linear SVM

We tested whether new or missing cell populations in the training set could influence tree learning. We mimicked this scenario using the simulated dataset and the same batches as in the previous tree learning experiment. In the previous experiment, ‘Group5’ and ‘Group6’ were merged into ‘Group56’ in Batch 2, but now we removed ‘Group5’ completely from this batch (Table S6). In this setup, the linear SVM misconstructs all trees (Figure 4E). If ‘Group5’ is present in one batch and absent in another, the ‘Group5’ cells are not rejected, but labeled as ‘Group6’. Consequently, ‘Group6’ is added as a parent node to ‘Group5’ and ‘Group6’. On the other hand, the one-class SVM suffers less than the linear SVM from these missing populations and correctly learns the expected tree in 2/3 of the cases (Figure 4F). In the remaining third (20 trees), ‘Group5’ matched perfectly with ‘Group456’ and was thus not added to the tree. This occurs only 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 resolution consequently resulted in the correct tree.

3.2.6 Linear SVM can learn the classification tree during an inter-dataset experiment

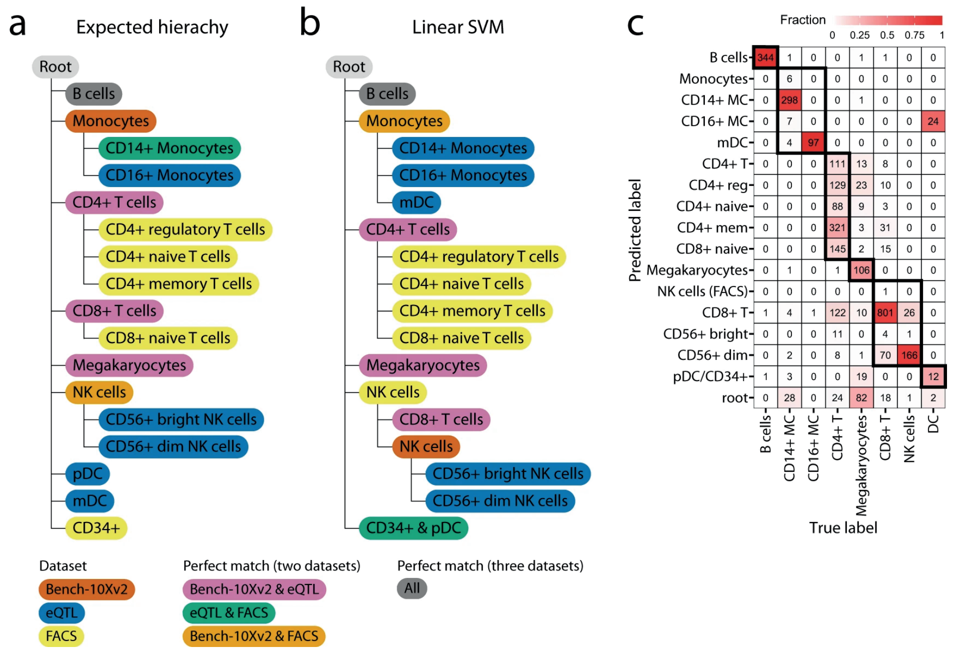
Finally, we tested scHPL in a realistic scenario by using three PBMC datasets (PBMC-eQTL, PBMC-Bench10Xv2, and PBMC-FACS) to learn a classification tree and using this tree to predict the labels of a fourth PBMC dataset (PBMC-Bench10Xv3) (Table 1). Before applying scHPL, we aligned the datasets using Seurat [28]. We constructed an expected classification tree based on the names of the cell populations in the datasets (Figure 5A). Note that matching based on names might result in an erroneous tree since every dataset was labeled using different clustering techniques, marker genes, and their own naming conventions.

When comparing the tree learned using the linear SVM to the expected tree, we notice five differences (Figure 5A-B). Some of these differences are minor, such as the matching of monocytes from the Bench10Xv2 dataset to myeloid dendritic cells (mDC), CD14+ monocytes, and the CD16+ monocytes. Monocytes can differentiate into mDC which can explain their transcriptomic similarity [29]. Other differences between the reconstructed and the expected trees are likely the result of (partly) mislabeled cell populations in the

Table 1. Number of cells per cell population in the different training datasets (batches) and test dataset. Subpopulations are indicated using an indent.

Cell population	Batch 1 eQTL	Batch 2 Bench 10Xv2	Batch 3 FACS	Test dataset Bench 10Xv3
CD19+ B	812	676	2,000	346
CD34+			2,000	
Monocytes (MC)		1,194		
CD14+	2,081		2,000	354
CD16+	274			98
CD4+ T	13,523	1,458		960
Reg.			2,000	
Naive			2,000	
Memory			2,000	
CD8+ T	4,195	2,128		962
Naive			2,000	
Megakaryocyte (MK)	142	433		270
NK cell		429	2,000	194
CD56+ bright	355			
CD56+ dim	2,415			
Dendritic				35
Plasmacytoid (pDC)	101			
Myeloid (mDC)	455			

original datasets (Figure S6-15). (*i*) According to the expression of *FCER1A* (a marker for mDC) and *FCGR3A* (CD16+ monocytes), the labels of the mDC and the CD16+ monocytes in the eQTL dataset are reversed (Figure S6-8). (*ii*) Part of the CD14+ monocytes in the FACS dataset express *FCER1A*, which is a marker for mDC (Figure S6, S8-9). The CD14+ monocytes in the FACS dataset are thus partly mDCs, which explains the match with the mDC from the eQTL dataset. (*iii*) Part of the CD4+ T-cells from the eQTL and Bench10Xv2 dataset should be relabeled as CD8+ T-cells (Figure S6, S10-13). In these datasets, the cells labeled as the CD8+ T-cells only contain cytotoxic CD8+ T-cells, while naive CD8+ T-cells are all labeled as CD4+ T-cells. This mislabeling explains why the CD8+ naive T-cells are a subpopulation of the CD4+ T-cells. (*iv*) Part of the CD34+ cells in the FACS dataset should be relabeled as pDC (Figure S6, S14-15), which explains why the pDC are a subpopulation of the CD34+ cells. In the FACS dataset, the labels were obtained using sorting, which would indicate that these labels are correct. The purity of the CD34+ cells, however, was significantly low (45%) compared to other cell populations (92-100%) [27]. There is only one difference , however, that cannot be explained by mislabeling. The NK cells from the FACS dataset do not only match the NK cells from the eQTL dataset, but also the CD8+ T-cells.



	B cells	CD14+ MC	CD16+ MC	CD4+ T	Megakaryocytes	CD8+ T	NK cells	DC
B cells	344	1	0	0	1	1	0	0
Monocytes	0	6	0	0	0	0	0	0
CD14+ MC	0	298	0	0	1	0	0	0
CD16+ MC	0	7	0	0	0	0	0	24
mDC	0	4	97	0	0	0	0	0
CD4+ T	0	0	0	111	13	8	0	0
CD4+ reg	0	0	0	129	23	10	0	0
CD4+ naive	0	0	0	88	9	3	0	0
CD4+ mem	0	0	0	321	3	31	0	0
CD8+ naive	0	0	0	145	2	15	0	0
Megakaryocytes	0	1	0	1	106	0	0	0
NK cells (FACS)	0	0	0	0	0	1	0	0
CD8+ T	1	4	1	122	10	801	26	0
CD56+ bright	0	0	0	11	0	4	1	0
CD56+ dim	0	2	0	8	1	70	166	0
pDC/CD34+	1	3	0	0	19	0	0	12
root	0	28	0	24	82	18	1	2

Figure 5. PBMC inter-dataset evaluation. A) Expected and **B) l**earned classification tree when using a linear SVM on the PBMC datasets. The color of a node represents the agreement between dataset(s) regarding that cell population. **C)** Confusion matrix when using the learned classification tree to predict the labels of PBMC-Bench10Xv3. The dark boundaries indicate the hierarchy of the constructed classification tree.

Most cells of the Bench10Xv3 dataset can be correctly annotated using the learned classification tree (Figure 5E). Interestingly, we notice that the CD16+ monocytes are predicted to be mDCs and vice versa, which could be explained by the fact that the labels of the mDCs and the CD16+ monocytes were flipped in the eQTL dataset. Furthermore, part of the CD4+ T-cells are predicted to be CD8+ naïve T-cells. In the Bench10Xv3, we noticed the same mislabeling of part of the CD4+ T-cells as in the eQTL and Bench10Xv2 datasets, which supports our predictions (Figure S6, S10-13).

The tree constructed using the one-class SVM differs slightly compared to the linear SVM (Figure S16A). Here, the monocytes from the Bench10Xv2 match the CD14+ monocytes and mDC (which are actually CD16+ monocytes) as we would expect. Next, the CD14+ monocytes from the FACS dataset merge the CD16+ monocytes (which are actually mDC) and the monocytes. Using the one-class SVM the CD8+ T-cells and NK cells from the Bench10Xv2 dataset are missing since there was a complex scenario. The NK cells are a relatively small population in this dataset which made it difficult to train a classifier for this population.

In the previous experiments, we used the default setting of Seurat to align the datasets (using 2000 genes). We tested whether changing the number of genes to 1000 and 5000 affects the performance. When using the one-class SVM, the number of genes does not affect tree construction. For the linear SVM, we notice one small difference when using 1000 genes: the CD8+ T-cells from the Bench10Xv2 dataset are a subpopulation of the CD8+ T-cells from the eQTL dataset instead of a perfect match.

The predicted labels of the Bench10Xv3 dataset change slightly when using a different number of genes (Figure S17). Whether more genes improves the prediction, differs per cell population. The labels of the megakaryocytes, for instance, are better predicted when more genes are used, but for the dendritic cells we observe the reverse pattern.

3.2.7 Mapping brain cell populations using scHPL

Next, we applied scHPL to construct a tree which maps the relationships between brain cell populations. This is a considerably more challenging task compared to PBMCs given the large number of cell populations as well as the fact that brain cell types are not consistently annotated. First, we combined two datasets from the primary visual cortex of the mouse brain, AMB2016 and AMB2018 [4,5]. AMB2018 contains more cells (12,771 vs. 1,298) and is clustered at a higher resolution (92 cell populations vs. 41) compared to AMB2016. Before applying scHPL, we aligned the datasets using Seurat [28]. Using scHPL with a linear SVM results in an almost perfect tree (Figure 6). We verified these results by comparing our constructed tree to cluster correspondences in Extended Data Fig. 6 from Tasic et al. [5]. Since AMB2018 is clustered at a higher resolution, most populations are subpopulations of AMB2016, which are all correctly identified by scHPL. Conversely, three L4 populations from AMB2016 were merged into one population (L4 IT VISp Rspo1) from AMB2018 [5], forming a continuous spectrum. This relation was also automatically identified using scHPL (Figure 6). Compared to the results from Tasic et al. [5], one cell population from AMB2018 is attached to a different parent node. scHPL assigned 'L6b VISp Col8a1 Rprm' as a subpopulation of 'L6a Sla' instead of 'L6b Rgs12'. This population, however, does not express *Rgs12*, but does express *Sla* (Figure S18), supporting the matching identified by scHPL. Three cell populations could not be added to the tree due to complex scenarios. According to Extended Data Fig. 6 from Tasic et al. [5], these AMB2018 populations are a subpopulation of multiple AMB2016 subpopulations.

The AMB2016 and AMB2018 datasets were generated and analyzed by the same group and hence the cluster matching is certainly easier than a real-life scenario. Therefore, we tested scHPL also on a complicated scenario with brain datasets that are sequenced using different protocols and by different labs (Table S7, Figure S19). We used three datasets (Zeisel, Tabula Muris, and Saunders) to construct the tree (Figure 7A-D) [2,30,31]. The Zeisel dataset is annotated at two resolutions. Before applying scHPL, we aligned the datasets using Seurat [28]. First, we constructed a tree using a linear SVM based on the low resolution of Zeisel. We started with the Saunders dataset and added Zeisel (Figure 7E). This is a clear illustration of the possible scenarios scHPL can manage. Some populations are a perfect match between the two datasets (e.g. neurons), some populations from Saunders are splitted (e.g. astrocytes), some are merged (e.g. macrophages and microglia), and some populations from Zeisel have no match (e.g. Ttr). Next, we updated the tree by adding the Tabula Muris dataset (Figure 7F). Here, we found matches that would not have been possible to identify by relying on the assigned cell type labels to map cell types. For example, mural cells from Saunders are a perfect match with the brain pericytes from the Tabula Muris. The results of scHPL with the one-class SVM were almost identical to the linear SVM (Figure S20A).

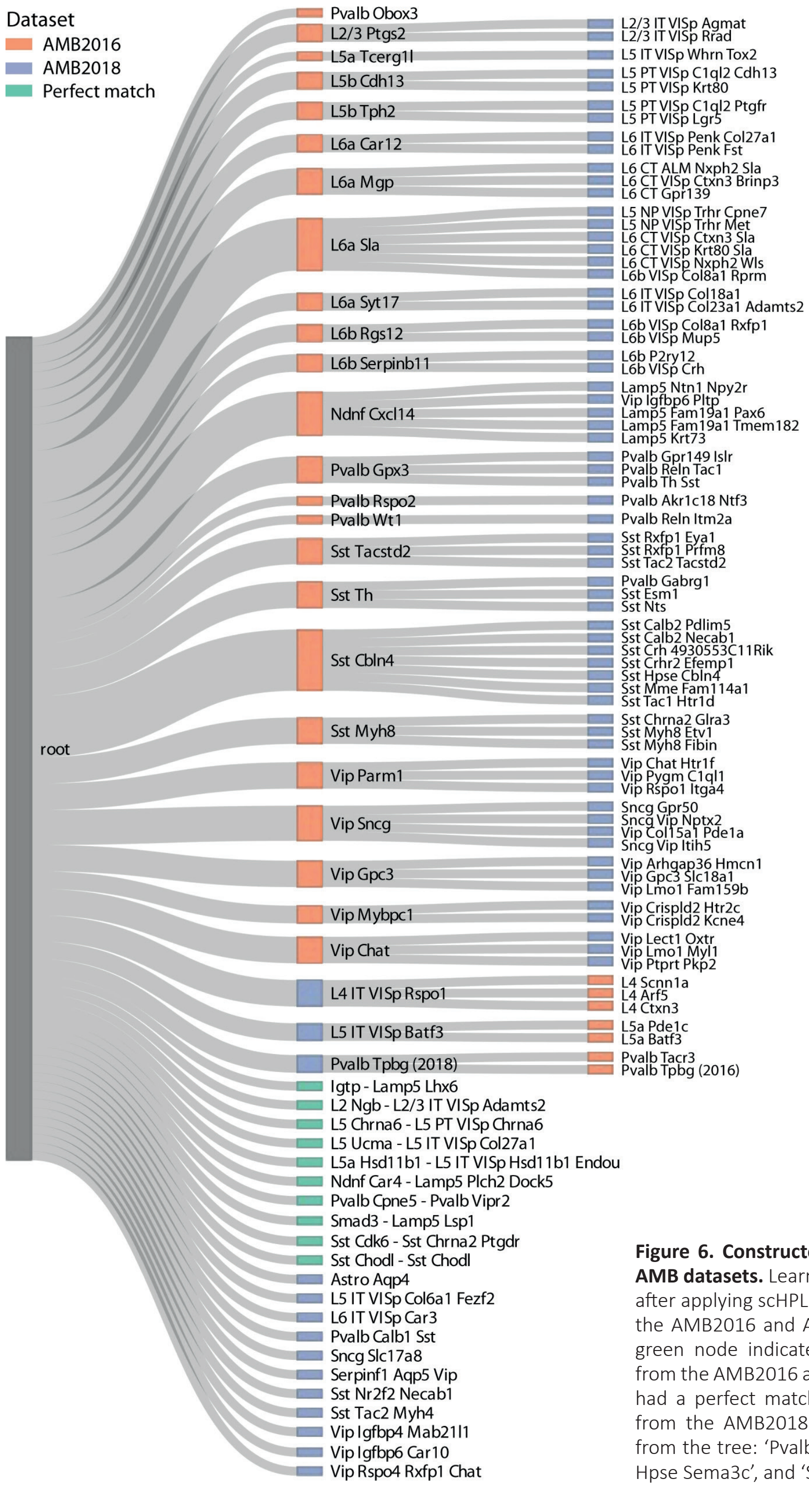


Figure 6. Constructed hierarchy for the AMB datasets. Learned classification tree after applying scHPL with a linear SVM on the AMB2016 and AMB2018 datasets. A green node indicates that a population from the AMB2016 and AMB2018 dataset had a perfect match. Three populations from the AMB2018 dataset are missing from the tree: 'Pvalb Sema3e Kank4', 'Sst Hpse Sema3c', and 'Sst Tac1 Tacr3'.

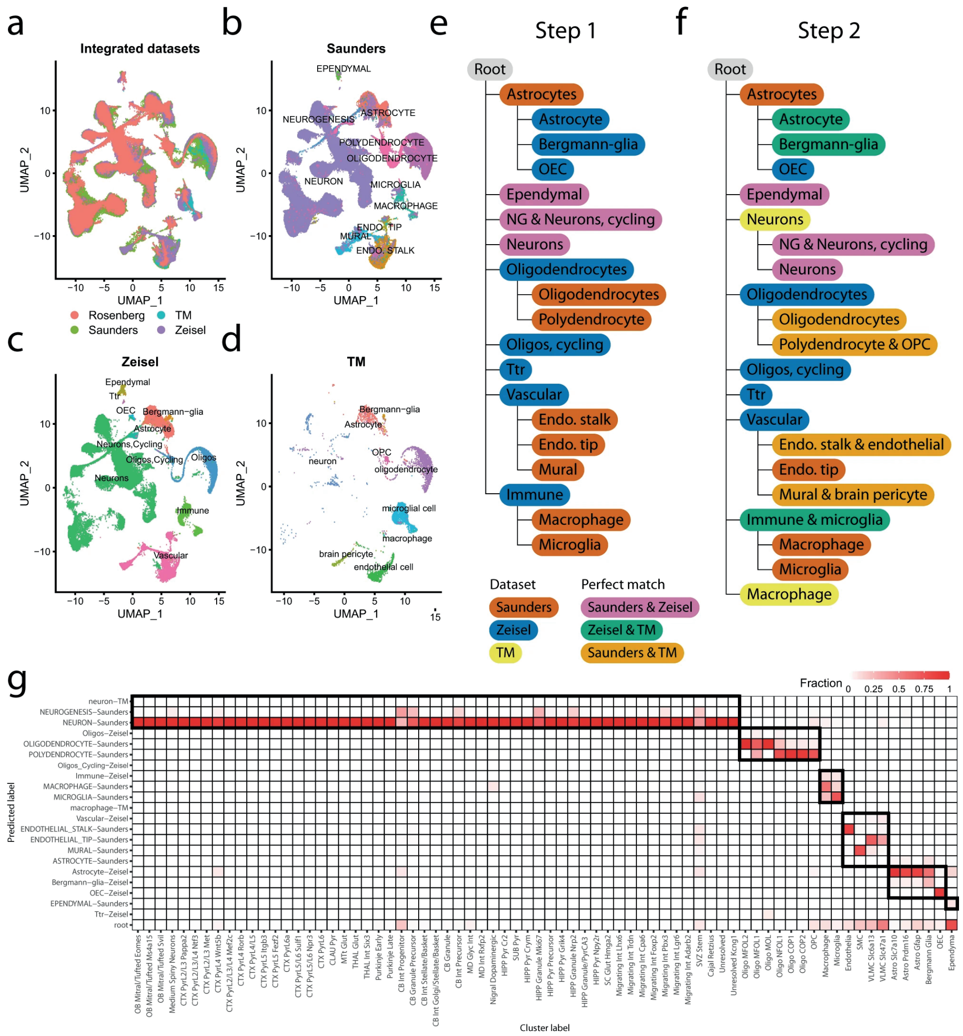


Figure 7. Brain inter-dataset evaluation. A-D) UMAP embeddings of the datasets after alignment using Seurat v3. **E)** Learned hierarchy when starting with the Saunders dataset and adding Zeisel with linear SVM. **F)** Updated tree when the Tabula Muris dataset is added. **G)** Confusion matrix when using the learned classification tree to predict the labels of Rosenberg. The dark boundaries indicate the hierarchy of the classification tree.

Next, we used the resulting tree to predict the labels of a fourth independent dataset (Rosenberg) [32]. The predictions from the linear and the one-class SVM are very similar (Figure 7G, S20B). The only difference is that the linear SVM correctly predicts some progenitor or precursor neuronal populations from Rosenberg to be 'neurogenesis' while the one-class SVM rejects these populations.

To assess the effect of the annotation resolution, we repeated the analysis using the higher resolution annotation from the Zeisel dataset (Figure S21-23). Here, we noticed that the 'brain pericytes (TM)' and 'pericytes (Zeisel)'- two populations one would easily match based on the names only- are not in the same subtree. 'Brain pericyte (TM)' forms a perfect match with ‘mural (Saunders)’ and ‘vascular smooth muscle cells (Zeisel)’, while ‘pericytes (Zeisel)’ is a subpopulation of ‘endothelial stalk (Saunders)’ and ‘endothelial cell (TM)’ (Figure S22-23). In the UMAP embedding of the integrated datasets, the ‘pericytes’ and ‘brain pericyte’ are at a different location, but they do overlap with the cell populations they were matched with (Figure S21). This highlights the power of scHPL matching rather than name-based matching.

3.3 Discussion

In this study, we showed that scHPL can learn cell identities progressively from multiple reference datasets. We showed that using our approach the labels of two AMB datasets can successfully be matched to create a hierarchy containing mainly neuronal cell populations and that we can combine three other brain datasets to create a hierarchy containing mainly non-neuronal cell populations. In both experiments, we discovered new relationships between cell populations, such as the mapping of ‘L6b VISp Col8a1 Rprm’ as a subpopulation of ‘L6b Sla’ instead of ‘L6b Rgs12’. This observation would not be possible to make by manually matching populations based on the assigned labels, highlighting the power of automatically constructing cellular hierarchies. In this case, the Cell Ontology database could also not be used to verify this relationship since many brain cell populations are missing. Most of these populations have recently been annotated using scRNA-seq and there is a wide lack of consistency in population annotation and matching between studies [18]. scHPL can potentially be used to map these relations, irrespective of the assigned labels, and improve the Cell Ontology database.

When combining multiple datasets to construct a tree, we expect that cell populations are annotated correctly. However, in the PBMC inter-dataset experiment, this was not the case. At first sight, the constructed tree looked erroneous, but the expression of marker genes revealed that (parts of) several cell populations were mislabeled. Here, we could use the constructed tree as a warning that there was something wrong with the original annotations.

In general, scHPL is robust to sampling differences between datasets or varying parameters such as the matching threshold or the number of genes used. The brain datasets used to construct the tree, for instance, varied greatly in population sizes, which did not cause any difficulties. This is mainly because we rely on reciprocal classification. A match between cell populations that is missed when training a classifier on one dataset to predict labels of the other, can still be captured by the classifier trained on the other dataset.

Since batch effects are inevitable when combining datasets, we require datasets to be aligned before running scHPL. In all inter-dataset experiments in this manuscript, we used Seurat V3 [28] for the alignment, but we would like to emphasize that scHPL is not dependent on Seurat and can be combined with any batch correction tool, such as more computationally efficient methods like Harmony [33]. A current limitation of these tools is that when a new dataset is added, the alignment- and consequently also scHPL- has to be rerun. An interesting alternative would be to project the new dataset to a latent space learned using reference dataset(s), using scArches [34] for example. In that case, scHPL does not have to be rerun but can be progressively updated.

The batch effects between the datasets make it more difficult to troubleshoot errors. Generally, it will be hard to resolve whether mistakes in the constructed tree are caused by the erroneous alignment of datasets or by mismatches created by scHPL.

We would like to note though that there are inherent limitations to the assumption that cell populations have hierarchical relationships. While this assumption is widely adopted in single cell studies as well as the Cell Ontology, there are indeed situations in which a tree is not adequate. For instance, situations in which cells dedifferentiate into other cell types, such as beta to alpha cell conversions in type2 diabetes [35,36].

Considering the classification performance, we showed that using a hierarchical approach outperforms flat classification. On the AMB dataset, the linear SVM outperformed $SVM_{rejection}$, which was the best performing classifier on this dataset [23]. In contrast to $SVM_{rejection}$, the linear SVM did not reject any of the cells but labeled them as an intermediate cell population. During this experiment, there were no cells of unknown populations. Correct intermediate predictions instead of rejection are therefore beneficial since it provides the user with at least some information. When comparing the linear SVM and one-class SVM, we noticed that the accuracy of the linear SVM is equal to or higher than the one-class SVM on all datasets. For both classifiers, we saw a decrease in performance on populations with a small number of cells, but for the one-class SVM this effect was more apparent.

Since the one-class SVM has a low performance on small cell populations, it also cannot be used to combine datasets which consist of small populations. If the classification performance is low, it will also not be possible to construct the correct tree. On the other hand, the performance of the linear SVM seems to be robust to small populations throughout our experiments. This classifier can thus better be used when combining multiple datasets with small populations.

When testing the rejection option, the one-class SVM clearly outperforms the linear SVM by showing a perfect performance on the simulated dataset. Moreover, when cell populations are missing from the simulated data, the linear SVM cannot learn the correct tree anymore, in contrast to the one-class SVM. This suggests that the one-class SVM is preferred when cell populations are missing, although on the AMB dataset, the rejection option of both classifiers was not perfect.

In summary, we present a hierarchical progressive learning approach to automatically identify cell identities based on multiple datasets with various levels of subpopulations. We show that we can accurately learn cell identities and learn hierarchical relations between cell populations. Our results indicate that choosing between a one-class and a linear SVM is a trade-off between achieving a higher accuracy and the ability to discover new cell populations. Our approach can be beneficial in single-cell studies where a comprehensive reference atlas is not present, for instance, to annotate datasets consistently during a cohort study. The first available annotated datasets can be used to build the hierarchical tree, which could subsequently can be used to annotate cells in the other datasets.

3.4 Methods

3.4.1 Hierarchical progressive learning

Within scHPL, we use a hierarchical classifier instead of a flat classifier. A flat classifier is a classifier that doesn't consider a hierarchy and distinguishes between all cell populations simultaneously. For the AMB dataset, a flat classifier will have to learn the decision boundaries between all 92 cell populations in one go. Alternatively, a hierarchical classifier divides the problem into smaller subproblems. First it learns the difference between the 3 broad classes: GABAergic neurons, glutamatergic neurons, and non-neuronal cells. Next, it learns the decision boundaries between the six subtypes of GABAergic neurons and the eight subtypes of glutamatergic neurons, separately. Finally, it will learn the decision boundaries between the different cell populations within each subtype separately.

3.4.2 Training the hierarchical classifier

The training procedure of the hierarchical classifier is the same for every tree: we train a local classifier for each node except the root. This local classifier is either a one-class SVM or a linear SVM. We used the one-class SVM (`svm.OneClassSVM(nu = 0.05)`) from the scikit-learn library in Python [37]. A one-class classifier only uses positive training samples. Positive training samples include cells from the node itself and all its child nodes. To avoid overfitting, we select the first 100 principal components (PCs) of the training data. Next, we select informative PCs for each node separately using a two-sided two-sample t-test between the positive and negative samples of a node ($\alpha < 0.05$, Bonferroni corrected). Negative samples are selected using the siblings policy [38], i.e. sibling nodes include all nodes that have the same ancestor, excluding the ancestor itself. If a node has no siblings, cells labeled as the parent node, but not the node itself are considered negative samples. In some rare cases, the Bonferroni correction was too strict and no PCs were selected. In those cases, the five PCs with the smallest p-values were selected. For the linear SVM, we used the `svm.LinearSVC()` function from the scikit-learn library. This classifier is trained using positive and negative samples. The linear SVM applies L2-regularization by default, so no extra measures to prevent overtraining were necessary.

3.4.3 The reconstruction error

The reconstruction error is used to reject unknown cell populations. We use the training data to learn a suitable threshold which can be used to reject cells by doing a nested 5 fold cross-validation. A PCA ($n_{components} = 100$) is learned on the training data. The test data is then reconstructed by first mapping the data to the selected PCA domain, and then mapping the data back to the original space using the inverse transformation (hence the data lies within the plane spanned by the selected PCs). The reconstruction error is the difference between the original data and the reconstructed data (in other words, the distance of the original data to the PC plane). The median of the q^{th} (default $q = 0.99$) percentile of the errors across the test data is used as threshold. By increasing or decreasing this parameter, the number of false negatives can be controlled. Finally, we apply a PCA ($n_{components} = 100$) to the whole dataset to learn the transformation that can be applied to new unlabeled data later.

3.4.4 Predicting the labels

First, we look at the reconstruction error of a new cell to determine whether it should be rejected. If the reconstruction error is higher than the threshold determined on the training data, the cell is rejected. If not, we continue with predicting its label. We start at the root node, which we denote as parent node and use the local classifiers of its children to predict the label of the cell using the `predict()` function, and score it using the `decision_function()`, both from the scikit-learn package. These scores represent the signed distance of a cell to the decision boundary. When comparing the results of the local classifiers, we distinguish three scenarios:

1. All child nodes label the cell negative. If the parent node is the root, the new cell is rejected. Otherwise we have an internal node prediction and the new cell is labeled with the name of the parent node.
2. One child node labels the cell positive. If this child node is a leaf node, the sample is labeled with the name of this node. Otherwise, this node becomes the new parent and we continue with its children.
3. Multiple child nodes label the cell positive. We only consider the child node with the highest score and continue as in scenario two.

3.4.5 Reciprocal matching labels and updating the tree

Starting with two datasets, *D1* and *D2*, and the two corresponding classification trees (which can be either hierarchical or flat), we would like to match the labels of the datasets and merge the classification trees accordingly into a new classification tree while being consistent with both input classification trees (Figure 1). We do this in two steps: first matching the labels between the two dataset and then updating the tree.

Reciprocal matching labels. We first cross-predict the labels of the datasets: we use the classifier trained on *D1* to predict the labels of *D2* and vice versa. We construct confusion matrices, *C1* and *C2*, for *D1* and *D2*, respectively. Here, $C1_{ij}$ indicates how many cells of population *i* of *D1* are predicted to be population *j* of *D2*. This prediction can be either a leaf node, internal node or a rejection. As the values in *C1* and *C2* are highly dependent on the size of a cell population, we normalize the rows such that the sum of every row is one, now indicating the fraction of cells of population *i* in *D1* that have been assigned to population *j* in *D2*:

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

Clearly, a high fraction is indicative of matching population *i* in *D1* with population *j* in *D2*. Due to splitting, merging, or new populations between both datasets, multiple relatively high fractions can occur (e.g. if a population *i* is split in two populations j_1 and j_2 due to *D2* being of a higher resolution, both fractions NC_{ij1} and NC_{ij2} will be approximately 0.5). To accommodate for these operations, we allow multiple matches per population.

To convert these fractions into matches, *NC1* and *NC2* are converted into binary confusion matrices, *BC1* and *BC2*, where a 1 indicates a match between a population in *D1* with a population in *D2*, and vice versa. To determine a match, we take the value of the fraction as well as the difference with the other fractions into account. This is done for each row (population) of *NC1* and *NC2* separately. When considering row *i* from *NC1,* we first rank all fractions, then the highest fraction will be set to 1 in *BC1*, after which all fractions for which the difference with the preceding (higher) fraction is less than a predefined threshold (default = 0.25) will also be set to 1 in *BC1*.

To arrive at reciprocal matching between *D1* and *D2,* we combine *BC1* and *BC2* into matching matrix *X* (Figure 2):

$$X = BC1^T + BC2$$

The columns in *X* represent the cell populations of *D1* and the rows represent the cell populations of *D2*. If $X_{ij} = 2$, this indicates a reciprocal match between cell population *i* from *D2* and cell populations *j* from *D1*. $X_{ij} = 1$ indicates a one-sided match, and $X_{ij} = 0$ represents no match.

Tree updating. Using the reciprocal matches between *D1* and *D2* represented in *X*, we update the hierarchical tree belonging to *D1* to incorporate the labels and tree structure of *D2*. We do that by handling the correspondences in *X* elementwise. For a non-zero value in *X*, we check whether there are other non-zero values in the corresponding row and column to identify which tree operation we need to take (such as split/merge/create). As an example, if we encounter a split for population *i* in *D1* into j_1 and j_2, we will create new nodes for j_1 and j_2 as child nodes of node *i* in the hierarchical tree of *D1*. Figure 2 and Table S1 explain the four most common scenarios: a perfect match, splitting nodes, merging nodes, and a new population. All other scenarios are explained in Supplementary Note 1. After an update, the corresponding values in *X* are set to zero and we continue with the next non-zero element of *X*. If the matching is impossible, the corresponding values in *X* are thus not set to zero. If we have evaluated all elements of *X*, and there are still non-zero values, we will change *X* into a strict matrix, i.e. we further only consider reciprocal matches, so all '1's are turned into a '0' with some exceptions (Supplementary Note 2). We then again evaluate *X* element wise once more.

3.4.6 Evaluation

Hierarchical F1-score. We use the hierarchical F1-score (HF1-score) to evaluate the performance of the classifiers [39]. We first calculate the hierarchical precision (*hP*) and recall (*hR*):

$$hP = \frac{\sum_i P_i \cap T_i}{\sum_i P_i} \qquad hR = \frac{\sum_i P_i \cap T_i}{\sum_i T_i}$$

Here, P_i is a set that contains the predicted cell population for a cell *i* and all the ancestors of that node, T_i contains the true cell population and all its ancestors, and $P_i \cap T_i$ is the overlap between these two sets. The HF1-score is the harmonic mean of *hP* and *hR*:

$$\mathrm{HF1} = \frac{2hP*2hR}{hP+hR}$$

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

3.4.7 Datasets

Simulated data. We used the R-package Splatter (V1.6.1) to simulate a hierarchical scRNA-seq dataset that consists of 8,839 cells and 9,000 genes and represents the tree shown in Figure S1A (Supplementary Note 3) [26]. We chose this low number of genes to speed up the computation time. In total there are six different cell populations of approximately 1,500 cells each. As a preprocessing step, we log-transformed the count matrix ($\log_2(count+1)$). A UMAP embedding of the simulated dataset shows it indeed represents the desired hierarchy (Figure S1C).

Peripheral Blood Mononuclear Cells (PBMC) scRNA-seq datasets. We used four different PBMC datasets: PBMC-FACS, PBMC-Bench10Xv2, PBMC-Bench10Xv3, and PBMC-eQTL. The PBMC-FACS dataset is the downsampled FACS-sorted PBMC dataset from Zheng et al. [27]. 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 [27]. Each cell population consists of 2,000 cells. The total dataset consists of 20,000 cells and 21,952 genes. During the cross-validation on the PBMC-FACS dataset, we tested the effect of selecting HVG. We used the 'seurat_v3' flavor of scanpy to select 500, 1000, 2000, and 5000 HVG on the training set [28,40]. The PBMC-Bench10Xv2 and PBMC-Bench10Xv3 datasets are the PbmcBench pbmc1.10Xv2 and pbmc1.10Xv3 datasets from Ding et al. [41]. These datasets consist of 6,444 and 3,222 cells respectively, 22,280 genes, and nine different cell populations. Originally the PBMC-Bench10Xv2 dataset contained CD14+ and CD16+ monocytes. We merged these into one population called monocytes to introduce a different annotation level compared to the other PBMC datasets.The PBMC-eQTL dataset was sequenced using 10X Chromium and consists of 24,439 cells, 22,229 genes, and eleven different cell populations [42].

Brain scRNA-seq datasets. We used two datasets from the mouse brain, AMB2016 and AMB2018, to look at different resolutions of cell populations in the primary mouse visual cortex. The AMB2016 dataset was sequenced using SMARTer [4], downloaded from https://portal.brain-map.org/atlases-and-data/rnaseq/data-files-2018. AMB2016 consists of 1,298 cells and 21,413 genes. The AMB2018 dataset, which was sequenced using SMART-Seq V4 [5], downloaded from https://portal.brain-map.org/atlases-and-data/rnaseq/mouse-v1-

and-alm-smart-seq, consists of 12,771 cells and 42,625 genes. Additionally, we used four other brain datasets: Zeisel [2], Tabula Muris [30], Rosenberg [32], and Saunders [31]. These were downloaded from the scArches ‘data’ Google Drive (‘mouse_brain_regions.h5ad’ from https://drive.google.com/drive/folders/1QQXDuUjKG8CTnwWW_u83MDtdrBXr8Kpq) [34]. We downsampled each dataset such that at the highest resolution each cell population consisted of up to 5,000 cells to reduce the computational time for the alignment (Table S7).

Preprocessing scRNA-seq datasets. All datasets were preprocessed as described in Abdelaal et al. [23]. Briefly, we removed cells labeled in the original studies as doublets, debris or unlabeled cells, cells from cell populations with less than 10 cells, and genes that were not expressed. Next, 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 removed cells when the total number of detected genes was below three MAD from the median number of detected genes per cell. During the intra-dataset experiments, we log-transformed the count matrices ($\log_2(count+1)$).

Aligning scRNA-seq datasets. During the inter-dataset experiments, we aligned the datasets using Seurat V3 [28] based on the joint set of genes expressed in all datasets. In the PBMC, AMB, and brain inter-dataset experiment respectively 17,573, 19,197, and 14,858 genes remained. For the PBMC inter-dataset experiment, we also removed cell populations that consisted of less than 100 cells from the datasets used for constructing and training the classification tree (PBMC-eQTL, FACS, Bench10Xv2). To test the effect of the number of genes on scHPL, we integrated this data using 1000, 2000 (default), and 5000 HVGs.

3.5 Code and data availability

The filtered PBMC-FACS and AMB2018 dataset can be downloaded from Zenodo (https://doi.org/10.5281/zenodo.3357167). The simulated dataset and the aligned datasets used during the inter-dataset experiment can be downloaded from Zenodo (http://doi.org/10.5281/zenodo.3736493). Accession numbers or links to the raw data: AMB2016 [4] (GSE71585, https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE71585), AMB2018 [5] (GSE115746, https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE115746), PBMC-FACS [27] (SRP073767, https://support.10xgenomics.com/single-cell-gene-expression/datasets), PBMC-eQTL [42] (EGAS00001002560, https://ega-archive.org/studies/EGAS00001002560), PBMC-Bench10Xv2 and PBMC-Bench10Xv3 [41] (GSE132044, https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE132044), Rosenberg [32] (GSE110823, https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE110823), Zeisel [2] (http://mousebrain.org, file name L5_all.loom, downloaded on 9/9/2019), Saunders [31] (http://dropviz.org, DGE by Region section, downloaded on 30/8/2019), Tabula Muris [30] (https://figshare.com/projects/Tabula_Muris_Transcriptomic_characterization_of_20_organs_and_tissues_from_Mus_musculus_at_single_cell_resolution/27733, downloaded on 14/2/2019). The source code for scHPL is available as a python package that is installable through the PyPI repository (https://github.com/lcmmichielsen/scHPL) [43].

Bibliography

1. van der Wijst MG, de Vries DH, Groot HE, Trynka G, Hon C-C, Bonder M-J, et al. The single-cell eQTLGen consortium. Elife. 2020;9. doi:10.7554/eLife.52155
2. Zeisel A, Hochgerner H, Lönnerberg P, Johnsson A, Memic F, van der Zwan J, et al. Molecular Architecture of the Mouse Nervous System. Cell. 2018;174: 999–1014.e22. doi:10.1016/j.cell.2018.06.021
3. Svensson V, da Veiga Beltrame E, Pachter L. A curated database reveals trends in single-cell transcriptomics. Database. 2020;2020. doi:10.1093/database/baaa073
4. Tasic B, Menon V, Nguyen TN, Kim TK, Jarsky T, Yao Z, et al. Adult mouse cortical cell taxonomy revealed by single cell transcriptomics. Nat Neurosci. 2016;19: 335–346. doi:10.1038/nn.4216
5. 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. doi:10.1038/s41586-018-0654-5
6. Zhang Z, Luo D, Zhong X, Choi JH, Ma Y, Wang S, et al. SCINA: Semi-Supervised Analysis of Single Cells in Silico. Genes . 2019;10: 531. doi:10.3390/genes10070531
7. Pliner HA, Shendure J, Trapnell C. Supervised classification enables rapid annotation of cell atlases. Nat Methods. 2019; 1–4. doi:10.1038/s41592-019-0535-3
8. Kiselev VY, Yiu A, Hemberg M. scmap: projection of single-cell RNA-seq data across data sets. Nat Methods. 2018;15: 359. Available: https://doi.org/10.1038/nmeth.4644
9. Cao Z-J, Wei L, Lu S, Yang D-C, Gao G. Searching large-scale scRNA-seq databases via unbiased cell embedding with Cell BLAST. Nat Commun. 2020;11: 3458. doi:10.1038/s41467-020-17281-7
10. Alquicira-Hernandez J, Sathe A, Ji HP, Nguyen Q, Powell JE. ScPred: Accurate supervised method for cell-type classification from single-cell RNA-seq data. Genome Biol. 2019;20: 264. doi:10.1186/s13059-019-1862-5
11. de Kanter JK, Lijnzaad P, Candelli T, Margaritis T, Holstege FCP. CHETAH: a selective, hierarchical cell type identification method for single-cell RNA sequencing. Nucleic Acids Res. 2019;47: e95–e95. doi:10.1093/nar/gkz543
12. Wang S, Pisco AO, McGeever A, Brbic M, Zitnik M, Darmanis S, et al. Unifying single-cell annotations based on the Cell Ontology. bioRxiv. 2019; 810234. doi:10.1101/810234
13. Zeisel A, Muñoz-Manchado AB, Codeluppi S, Lönnerberg P, La Manno G, Juréus A, et al. Brain structure. Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. Science. 2015;347: 1138–1142. doi:10.1126/science.aaa1934
14. Hodge RD, Bakken TE, Miller JA, Smith KA, Barkan ER, Graybuck LT, et al. Conserved cell types with divergent features in human versus mouse cortex. Nature. 2019; 1–8. doi:10.1038/s41586-019-1506-7
15. Jarvis P. Towards a Comprehensive Theory of Human Learning. Taylor & Francis Ltd; 2006.
16. Yang BH, Asada H. Progressive learning and its application to robot impedance learning. IEEE Trans Neural Netw. 1996;7: 941–952. doi:10.1109/72.508937
17. Fayek HM. Continual Deep Learning via Progressive Learning. RMIT University. 2019.
18. Yuste R, Hawrylycz M, Aalling N, Aguilar-Valles A, Arendt D, Arnedillo RA, et al. A community-based transcriptomics classification and nomenclature of neocortical cell types. Nature Neuroscience. Nature Research; 2020. doi:10.1038/s41593-020-0685-8
19. Svensson V, Beltrame E da V. A curated database reveals trends in single cell transcriptomics. bioRxiv. 2019; 742304. doi:10.1101/742304
20. Wagner F, Yanai I. Moana: A robust and scalable cell type classification framework for single-cell RNA-Seq data. bioRxiv. 2018; 456129. doi:10.1101/456129
21. Bakken TE, Hodge RD, Miller JA, Yao Z, Nguyen TN, Aevermann B, et al. Single-nucleus and single-cell transcriptomes compared in matched cortical cell types. Soriano E, editor. PLoS One. 2018;13: e0209648. doi:10.1371/journal.pone.0209648
22. Aevermann BD, Novotny M, Bakken T, Miller JA, Diehl AD, Osumi-Sutherland D, et al. Cell type discovery using single-cell transcriptomics: implications for ontological representation. Hum Mol Genet. 2018;27: R40–R47. doi:10.1093/hmg/ddy100
23. Abdelaal T, Michielsen L, Cats D, Hoogduin D, Mei H, Reinders MJT, et al. A comparison of automatic cell identification methods for single-cell RNA sequencing data. Genome Biol. 2019;20: 194. doi:10.1186/s13059-019-1795-z
24. Boufea K, Seth S, Batada NN. scID Uses Discriminant Analysis to Identify Transcriptionally Equivalent Cell Types across Single-Cell RNA-Seq Data with Batch Effect. iScience. 2020;23: 100914. doi:10.1016/j.isci.2020.100914
25. Tax D. One-class classification Concept-learning in the absence of counter-examples. TU Delft. 2001.
26. Zappia L, Phipson B, Oshlack A. Splatter: simulation of single-cell RNA sequencing data. Genome Biol. 2017;18: 174. doi:10.1186/s13059-017-1305-0
27. 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. doi:10.1038/ncomms14049
28. Stuart T, Butler A, Hoffman P, Hafemeister C, Papalexi E, Mauck WM, et al. Comprehensive Integration of Single-Cell Data. Cell. 2019;177: 1888–1902.e21. doi:10.1016/j.cell.2019.05.031

29. León B, López-Bravo M, Ardavín C. Monocyte-derived dendritic cells. Semin Immunol. 2005;17: 313–318. doi:10.1016/j.smim.2005.05.013

30. 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. doi:10.1038/s41586-018-0590-4

31. Saunders A, Macosko EZ, Wysoker A, Goldman M, Krienen FM, de Rivera H, et al. Molecular Diversity and Specializations among the Cells of the Adult Mouse Brain. Cell. 2018;174: 1015–1030.e16. doi:10.1016/j.cell.2018.07.028

32. Rosenberg AB, Roco CM, Muscat RA, Kuchina A, Sample P, Yao Z, et al. Single-cell profiling of the developing mouse brain and spinal cord with split-pool barcoding. Science. 2018;360: 176–182. doi:10.1126/science.aam8999

33. Korsunsky I, Millard N, Fan J, Slowikowski K, Zhang F, Wei K, et al. Fast, sensitive and accurate integration of single-cell data with Harmony. Nat Methods. 2019;16: 1289–1296. doi:10.1038/s41592-019-0619-0

34. Lotfollahi M, Naghipourfar M, Luecken M, Khajavi M, Büttner M, Avsec Z, et al. Query to reference single-cell integration with transfer learning. bioRxiv. 2020; 2020.07.16.205997. doi:10.1101/2020.07.16.205997

35. Cinti F, Bouchi R, Kim-Muller JY, Ohmura Y, Sandoval PR, Masini M, et al. Evidence of β-Cell Dedifferentiation in Human Type 2 Diabetes. J Clin Endocrinol Metab. 2016;101: 1044–1054. doi:10.1210/jc.2015-2860

36. Hunter CS, Stein RW. Evidence for Loss in Identity, De-Differentiation, and Trans-Differentiation of Islet β-Cells in Type 2 Diabetes. Front Genet. 2017;8: 35. doi:10.3389/fgene.2017.00035

37. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. 2011 pp. 2825–2830. Available: http://scikit-learn.sourceforge.net.

38. Fagni T, Sebastiani F. On the Selection of Negative Examples for Hierarchical Text Categorization. Proceedings of the 3rd language technology conference. 2007; 24–28.

39. Kiritchenko S, Famili F. Functional Annotation of Genes Using Hierarchical Text Categorization. Proceedings of BioLink SIG, ISMB. 2005.

40. Wolf FA, Angerer P, Theis FJ. SCANPY: Large-scale single-cell gene expression data analysis. Genome Biol. 2018;19: 15. doi:10.1186/s13059-017-1382-0

41. Ding J, Adiconis X, Simmons SK, Kowalczyk MS, Hession CC, Marjanovic ND, et al. Systematic comparison of single-cell and single-nucleus RNA-sequencing methods. Nat Biotechnol. 2020;38: 737–746. doi:10.1038/s41587-020-0465-8

42. Van Der Wijst MGP, Brugge H, De Vries DH, Deelen P, Swertz MA, Franke L. Single-cell RNA sequencing identifies celltype-specific cis-eQTLs and co-expression QTLs. Nat Genet. 2018;50: 493–497. doi:10.1038/s41588-018-0089-9

43. L.C.M. Michielsen, M.J.T. Reinders, A. Mahfouz. Hierarchical progressive learning of cell identities in single-cell data. 2021. doi:10.5281/zenodo.4644285

Supplementary Materials

Supplementary Note 1

When matching the cell populations from two datasets, we distinguish five options: simple, multiple columns, multiple rows, complex, and impossible. When describing the different scenarios within these options, we sometimes make a distinction between leaf nodes and internal nodes. Here, it is important to remember that only *T1* can have internal nodes since this is the tree that is updated. *T2* is always a flat classification tree, so only consists of the root node and leaf nodes.

Simple. In this scenario, we find a unique match between a cell population, P_i, from dataset 1 and a cell population, P_j, from dataset 2. As as consequence, $X_{j,i}$ will be 1 or 2 and the rest of row *j* and column *i* in *X* are zero. Within this scenario, there are three different options:

1. Both cell populations are leaf or internal nodes. This indicates a perfect match. The tree is not updated, but the labels of P_j are renamed to P_i (Figure S24A). This is the same scenario as the 'perfect match' scenario described in the main text.
2. P_i is a leaf or internal node, but P_j is the root node of *T2*. This indicates that P_i is missing in dataset 2. The node, however, is already in the tree, so it is not updated (Figure S24B).
3. P_i is the root of *T1*, but the P_j is a leaf node. This indicates that P_j is missing in dataset 1. The cell population is thus also not in the tree yet, so we will add it as a child to the root (Figure S24C). This is the same scenario as the 'new population' scenario described in the main text.

Multiple rows. In this scenario, a cell population, P_i, from dataset 1 matches multiple populations from dataset 2. In *X* there will be multiple non-zero values in column *i.* Here, we distinguish two different scenarios:

1. P_i matches only cell populations from dataset 2 that are leaf node. We consider the cell populations from dataset 2 subpopulations of P_i, so we add them as descendants to P_i (Figure S25A). This is the same scenario as the 'splitting nodes' scenario described in the main text.
2. The root node of *T2* is also involved. We simple ignore this node and for the rest do the same as above (Figure S25B-C).

Multiple columns. This scenario is quite similar to the multiple rows scenario. Here, however, multiples populations from dataset 1 match one cell population, P_j, of dataset 2. In *X* there will be multiple non-zero values in row *j.* This scenario is a little more complex since the populations 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 distinguish three different scenarios:

1. The root node of *T1* and *T2* are not involved, so multiple cell populations, which can be leaf or internal nodes, from dataset 1 match P_j. We consider the cell populations from dataset 1 subpopulations of P_j, so we need to add P_j as a parent node to these cell populations (Figure S26A). This is same scenario as the 'merging nodes' scenario described in the main text. It could be, however, that this node already exists in this tree

(Figure S26B). 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 of P_j.

2. Besides leaf or internal nodes, the root of *T1* is involved. This indicates that P_j is 'bigger' than the cell populations from dataset 1 as part of it is unlabeled. Therefore, we add P_j as a descendant to the root of *T1*. Next, we rewire the involved cell populations from dataset 1 such that they become descendants of P_j (Figure S26C).
3. The root node of *T2* is involved. This indicates that multiple cell populations from dataset 1 are missing in dataset 2. These nodes, however, are already in the tree, so the tree can remain the same (Figure S26D).

Complex. The scenarios described above were all relatively easy. A cell population from one dataset matches either one or multiple cell populations from another. It could also happen, however, that multiple cell populations from dataset 1 match multiple cell populations from dataset 2 (Figure S27). As a consequence, there will a certain place $X_{j,I}$ which is either 1 or 2 and there are two or more non-zero values in the corresponding row *j* and column *i*. Here, we distinguish three different scenarios:

1. The root node of *T1* 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 S27A). 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 *T2* is involved. Again, we just assume that the boundary should be adjusted, so we remove this `1' from the table (Figure S27B). 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' cell population, P_i, 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 cell populations of dataset 2 as descendants to the complex node (splitting scenario) and attach the involved cell populations of dataset 1, except for P_i, to P_j (Figure S27C).

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 S28A). 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 S28B). 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 Note 2

If there is a complex scenario that cannot be solved immediately, matrix *X* will be changed into a strict matrix. In the strict matrix, only reciprocal matches are considered, so all '1's' are turned into '0'. There are some exceptions to this rule.

- A population can never have a reciprocal match with the root, so these '1's' are never removed.
- If a population from a dataset has only one match, it is also never removed. Consider the following example: If population P1 of Dataset 1 is only predicted to be Population Q of Dataset 2, we know that P1 should be a match with Q as it cannot be matched with any other population or with the root. It could be that this match is not reciprocal if population Q has many different subpopulations (e.g. P1, P2, P3, P4). Imagine that population P2 is really big. Almost all cells of population Q will be predicted to be P2 and so the matches with P1 (and P3 and P4) are missed because of the matching threshold. In case there is a complex scenario caused by any other population (maybe P2 or P3 or P4), we still know that P1 is a subpopulation of Q, since that was super clear and didn't cause any complexity.

Supplementary Note 3

Current scRNA-seq data simulators cannot simulate hierarchical data, so we simulated this dataset step by step (Figure S1B).

First, we simulated the expression of 3,000 genes for 9,000 cells. For this simulation, the cells were divided into three groups. The 3,000 simulated genes represent genes that are differentially expressed between the cell populations at a low resolution, so for example B cells vs. T cells. Next, we simulated *another* 3,000 genes for the *same* 9,000 cells. Now, the cells were divided into five groups. Here, the differentially expressed genes represent genes that distinguish cell populations at a slightly higher resolution, so for example CD4+ T cells vs. CD8+ T cells. We repeated this step for another set of 3,000 genes, but now there were six populations. The third dataset represents the highest resolution, so for instance CD4+ memory T cells vs. CD4+ naïve T cells.

Together this resulted in a dataset of 9,000 cells and 9,000 genes. The cells were labeled at three resolutions. There was some inconsistency between the labels at the different resolutions (e.g. some cells were labeled as 'Group12', 'Group3', 'Group3'). We removed these cells from the dataset, which resulted in a final dataset of 8,839 cells and 9,000 genes.