Hierarchical progressive learning of cell identities in single-cell data

Lieke Michielsen[1,2,3] (l.c.m.michielsen@tudelft.nl)
Marcel J.T. Reinders[1,2,3] (m.j.t.reinders@tudelft.nl)
Ahmed Mahfouz[1,2,3]* (a.mahfouz@lumc.nl)

[1] Department of Human Genetics, Leiden University Medical Center, Einthovenweg 20, 2333ZC, Leiden, The Netherlands
[2] Leiden Computational Biology Center, Leiden University Medical Center, Einthovenweg 20, 2333ZC, Leiden, The Netherlands
[3] Delft Bioinformatics Lab, Delft University of Technology, Van Mourik Broekmanweg 6, 2628XE, Delft, The Netherlands

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

Abstract

Supervised methods are increasingly used to identify cell populations in single-cell data and efforts are underway to generate comprehensive reference atlases which can be used to train these models. Yet, current supervised methods are limited in their ability to learn from multiple datasets simultaneously, are hampered by the annotation of the datasets at different resolutions, and do not preserve annotations when retrained on new datasets. The latter point is especially important as it deprives researchers from the ability to 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* allows researchers to annotate continuously increasing amounts of single-cell data, for example in consortia where datasets are collected at different times. *scHPL* is available at https://github.com/lcmmichielsen/hierarchicalprogressivelearning.

Keywords: cell identity, classification, single-cell RNA-sequencing, progressive learning

Background

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 and Garnett), while others search for similar cells in a reference database (e.g. scmap and Cell-BLAST), or train a classifier using a reference atlas or a labeled dataset (e.g. scPred and CHETAH) [6–11].

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. OnClass tries to overcome these shortcomings by mapping annotations to cell ontology classes and uses this to train a classifier [12]. These cell ontologies, however, were not developed for scRNA-seq data specifically. As a consequence, many new discovered smaller (sub)populations might be missing and relationships between cell populations might be inaccurate.

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 [13]. 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 [14, 15]. 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.

Moreover, the level of detail (resolution) at which datasets are annotated highly depends on the number of cells analyzed [3]. 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 a flat classifier needs to distinguish between many classes, while if we exploit the hierarchy, the classification problem is divided into smaller sub-problems. 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, 10, 16]. 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 [17, 18]. Moreover, we have previously shown that adding prior knowledge is not beneficial [19]. 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 [19], 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 [19]. In most cases, the rejection option is implemented by setting a threshold on the posterior probability [7, 9, 19, 20]. 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.

Here, 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.

Results

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) 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.

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). Secondly, 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 [19], 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 [21].

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 simulated dataset was constructed using Splatter [22] and consists of 8,839 cells, 9,000 genes and 6 different cell populations (Figure 3). PBMC-FACS is the downsampled FACS-sorted PBMC dataset from [23] 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 3A, S1-2).

On all datasets, the linear SVM performs better than the one-class SVM (Figure 4A-D). The simulated dataset is relatively easy since the cell populations are widely separated (Figure 3C). 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 SVM is higher than the one-class SVM (Figure 4B-D). For the AMB dataset, we used the same cross-validation folds as in [19], 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 4D).

The linear (hierarchical) SVM also shows a better performance compared to $SVM_{rejection}$, which is a flat linear SVM with rejection option based on the posterior probability and was the best classifier for this data [19]. $SVM_{rejection}$, 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 4D). The number of rejected cells, on the other hand, is not considered when calculating the median F1-score. This number is relatively high for $SVM_{rejection}$ (19.8%). The linear SVM, on the contrary, has almost no rejected cells, which is also reflected in a higher HF1-score (Figure 4C). Because of this large amount of rejections of $SVM_{rejection}$, 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 4F). 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.

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 S2). 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 4E). On the contrary, the one-class SVM rejects leaf node cells better.

scHPL **correctly 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 5A-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 S1A). In the learned trees, CD4+ memory T-cells are a subpopulation of CD8+ instead of CD4+ T-cells. A t-SNE plot of the PBMC-FACS dataset confirms that CD4+ memory T-cells are more similar to CD8+ than CD4+ T-cells based on their transcriptomic profile (Figure S1B). Using the learned tree instead of the predefined hematopoietic tree also improves the classification performance of the linear SVM slightly (HF1-score = 0.990 vs 0.985). Moreover, when relying on the predefined hematopoietic tree, CD4+ memory T-cells, CD8+ T-cells, and CD8+ naive T-cells were also often confused, indicating that the learned PBMC tree might better reflect the data (Tables S4-5).

Missing populations affect cellular hierarchy learning 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 mis-constructs all trees (Figure 5E). 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 5F). 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.

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). We constructed an expected classification tree based on the names of the cell populations in the datasets (Figure 6A). 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 6A-B). Some of these mistakes are minor. The megakaryocytes from the eQTL dataset are for instance seen as a subpopulation of the megakaryocytes of the Bench10Xv2 dataset. Different marker genes were used to identify these populations, so it could indeed be that one set of marker genes was more specific [24, 25]. Looking at a UMAP embedding of the aligned datasets, we also notice that the two populations do not completely overlap (Figure 6C-D). The same explanation could be used for the CD14+ monocytes of the FACS dataset that match the monocytes from the Bench10Xv2 dataset - a combination of CD14+ and CD16+ monocytes - instead of the CD14+ monocytes of the eQTL dataset. Another minor mistake is that the myeloid dendritic cells (mDC) are seen as a subpopulation of monocytes, which can be explained by the fact that monocytes can differentiate into mDC [26].

There are two mistakes, however, that cannot be explained biologically. The NK cells from the FACS dataset do not match the NK cells from the eQTL and Bench10Xv2 dataset, but the CD8+ T-cells. The second mistake, also caused by the FACS dataset, is that the CD8+ naive T-cells are a subset of the CD4+ T-cells instead of CD8+ T-cells. Since a UMAP embedding of the aligned datasets supports the learned tree, it is unclear whether these mistakes are introduced by the alignment of the datasets or caused by the different annotation methods of the datasets (Figure 6C-D). That is, the eQTL and Bench10Xv2 were both annotated using clustering, while the FACS dataset was annotated based on the expression of cell surface markers.

Most cells of the Bench10Xv3 dataset can be correctly annotated using the learned classification tree (Figure 6E). Interestingly, we notice that the CD16+ monocytes are

predicted to be mDCs and vice versa, which could be explained by the aforementioned fact that monocytes can differentiate into dendritic cells.

When using a one-class SVM instead of a linear SVM, nine mistakes are made (Figure S3). Some mistakes are similar to the linear SVM, such as imperfect matching of the megakaryocytes or the mismatch of the CD8+ naive cells, but some mistakes are specific to the one-class SVM. Since the one-class SVM shows a low performance on cell populations with few cells, it also makes more mistakes when trying to match them. For instance, the NK bright cells from the eQTL dataset are not matching any population of the other datasets. This is probably since there are very little NK bright cells in the eQTL dataset, so the one-class SVM cannot define the decision boundary for this population properly. Larger cell populations, such as B-cells, can be matched without mistake when using the one-class SVM.

Discussion

In this study, we showed that *scHPL* has great potential for automatic cell identification in scRNA-seq data. We showed that using our approach the labels of three different PBMC datasets can successfully be matched to learn a classification tree that largely mimics the known hematopoietic cellular hierarchy. In this experiment, mismatches, such as the megakaryocytes, confirm that matching purely based on the name is not possible and that a precise definition of most cell populations is missing. Two more serious mistakes were made when the PBMC-FACS dataset was added. This could be because of the alignment of the datasets or because of the different labeling procedures of the datasets. That is, the PBMC-FACS data was labeled based on the expression of cell surface markers instead of clustering the cells, and it is known that protein and gene expression poorly correlated [27, 28], which might explain these results.

Furthermore, 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 [19]. 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.

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.

Conclusion

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.

Methods

Hierarchical progressive learning

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 [29]. A one-class classifier only uses positive training samples, which in our case includes samples from the node itself and all its child nodes. To avoid overfitting, we first 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 ($\alpha < 0.05$, Bonferonni corrected). In some rare cases, this 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 also uses negative samples. These are selected using the siblings policy [30], i.e. sibling nodes include all nodes that have the same ancestor, excluding the ancestor itself. The linear SVM applies L2-regularization by default, so no extra measures to prevent overtraining were necessary.

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.

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.

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 depended on the size

of a cell population, we normalize the rows such that the sum of every row is one: $NC1_{ij} = \frac{C1_{ij}}{\sum_{\forall j} C1_{ij}}$, now indicating the fraction of cells of population *i* in *D1* that have been assigned to population *j* in *D2*. 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* (*X = BC1^T + BC2*) (Figure 2). 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'. We then again evaluate *X* element wise once more.

Evaluation

Hierarchical F1-score

We use the hierarchical F1-score (HF1-score) to evaluate the performance of the classifiers [31]. We first calculate the hierarchical precision (*hP*) and recall (*hR*): $hP = \frac{\sum_i P_i \cup T_i}{\sum_i P_i}$ and $hR = \frac{\sum_i P_i \cup T_i}{\sum_i T_i}$. 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 $P_i \cup T_i$ is the overlap between these two sets. The HF1-score is the harmonic mean of *hP* and *hR*: $HF1 = \frac{2hP * hR}{hP + hR}$.

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.

Datasets

Simulated data

We used the R-package Splatter version 1.6.1 to simulate a hierarchical scRNA-seq dataset that represents the tree shown in Figure 3A [22]. As Splatter is originally not developed to

simulate hierarchical data, we created such a dataset by simulating three datasets, each consisting of 9,000 cells and 3000 genes, and stacking them column wise. In these three datasets, we simulated different groups with different frequencies (Figure 3B). The final dataset consists of 8,839 cells and 9,000 genes. 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 3C).

PBMC data

We used four different PBMC datasets: PBMC-FACS, PBMC-Bench10Xv2, PBMC-Bench10Xv3, and PBMC-eQTL. All were preprocessed as described in [19].

The PBMC-FACS dataset is the downsampled FACS-sorted PBMC dataset from [23]. 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 [23]. Each cell population consists of 2,000 cells. The total dataset consists of 20,000 cells and 21,952 genes.

The PBMC-Bench10Xv2 and PBMC-Bench10Xv3 datasets are the PbmcBench pbmc1.10Xv2 and pbmc1.10Xv3 datasets from [25]. 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 [24].

When combining all PBMC datasets for the inter-dataset experiment, we removed genes not present in all datasets (17,573 genes remained), and cell populations that consisted of less

than 100 cells from the datasets used for constructing and training the classification tree (PBMC-eQTL, FACS, Bench10Xv2). Next, we aligned the datasets using Seurat V3 [32].

Allen Mouse Brain data

We used the Allen Mouse Brain (AMB) data to look at different resolutions of cell populations in the primary mouse visual cortex. This dataset, which was sequenced using SMART-Seq V4 [5] and preprocessed as described in [19], consists of 12,771 cells and 42,625 genes.

Declarations

Ethics approval and consent to participate. N/A

Consent for publication. N/A

Availability of data and materials

The aligned PBMC datasets used during the inter-dataset experiment can be downloaded from Zenodo (http://doi.org/10.5281/zenodo.3736493). The source code is available at the Github repository (https://github.com/lcmmichielsen/hierarchicalprogressivelearning).

Competing interests. The authors declare no competing interests.

Funding

This research was partially supported by an NWO Gravitation project: BRAINSCAPES: A Roadmap from Neurogenetics to Neurobiology (NWO: 024.004.012) and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 861190 (PAVE).

Authors' contributions

LM, MJTR, and AM conceived the study designed the experiments. LM performed all experiments and drafted the manuscript. LM, MJTR, and AM reviewed and approved the manuscript.

Acknowledgements. N/A

References

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.

3. Svensson V, Beltrame E da V. A curated database reveals trends in single cell transcriptomics. bioRxiv. 2019;:742304.

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;19:335–46.

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;563:72–8.

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.

7. Pliner HA, Shendure J, Trapnell C. Supervised classification enables rapid annotation of cell atlases. Nat Methods. 2019;:1–4.

8. Kiselev VY, Yiu A, Hemberg M. scmap: projection of single-cell RNA-seq data across data sets. Nat Methods. 2018;15:359.

9. 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.

10. 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.

11. 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.

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.

13. Jarvis P. Towards a Comprehensive Theory of Human Learning. Taylor & Francis Ltd; 2006.

14. Yang BH, Asada H. Progressive learning and its application to robot impedance learning. IEEE Trans Neural Netw. 1996;7:941–52.

15. Fayek HM. Continual Deep Learning via Progressive Learning. RMIT University; 2019.

16. Wagner F, Yanai I. Moana: A robust and scalable cell type classification framework for single-cell RNA-Seq data. bioRxiv. 2018;:456129.

17. 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. PLoS One. 2018;13:e0209648.

18. 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:R40–7.

19. 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.

20. 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.

21. Tax D. One-class classification Concept-learning in the absence of counter-examples. TU Delft; 2001.

22. Zappia L, Phipson B, Oshlack A. Splatter: simulation of single-cell RNA sequencing data. Genome Biol. 2017;18:174.

23. 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.

24. 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;50:493–7.

25. 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;38:737–46.

26. León B, López-Bravo M, Ardavín C. Monocyte-derived dendritic cells. Semin Immunol. 2005;17:313–8.

27. van den Berg PR, Budnik B, Slavov N, Semrau S. Dynamic post-transcriptional regulation during embryonic stem cell differentiation. bioRxiv. 2017;:123497.

28. Vogel C, Marcotte EM. Insights into the regulation of protein abundance from proteomic and transcriptomic analyses. Nat Rev Genet. 2012;13:227–32.

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

30. Fagni T, Sebastiani F. On the Selection of Negative Examples for Hierarchical Text Categorization. Proceedings of the 3rd language technology conference. 2007;:24–8.

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

32. Stuart T, Butler A, Hoffman P, Hafemeister C, Papalexi E, Mauck WM, et al. Comprehensive Integration of Single-Cell Data. Cell. 2019;177:1888–902.e21.

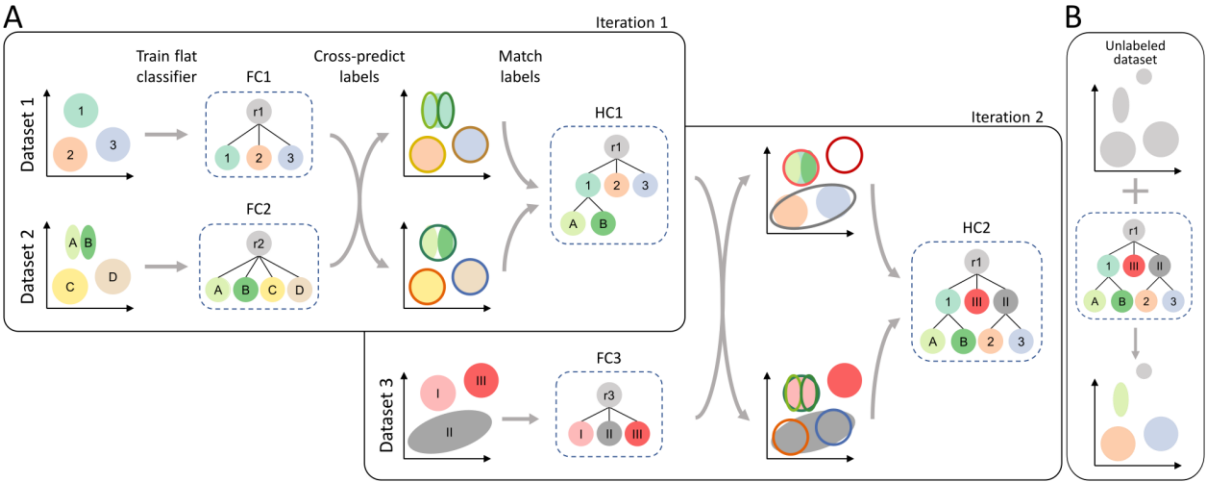

Figure 1. Schematic overview of *scHPL*. (A) Overview of the training phase. In the first iteration, we start with two labeled datasets. The colored areas represent the different cell populations. For both datasets a flat classifier (FC1 & FC2) is constructed. Using this tree and the corresponding dataset, 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.

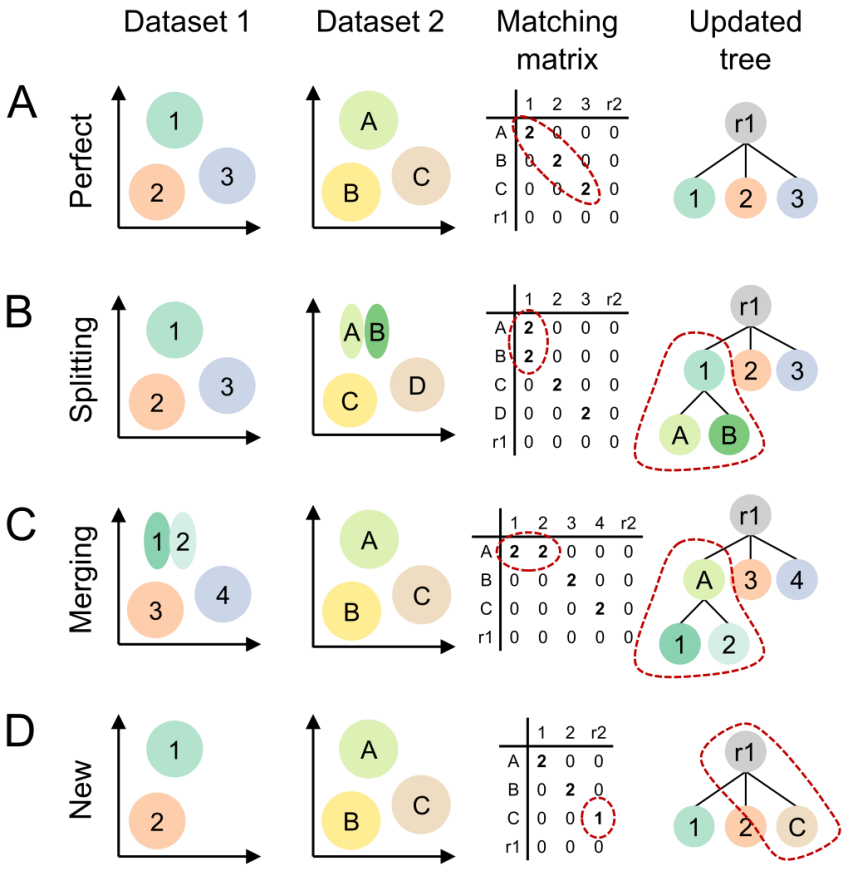

Figure 2. Schematic examples of different matching scenarios. (A) Perfect match, **(B)** splitting, **(C)** merging, **(D)** new population. The first two columns represent a schematic representation of two datasets. After cross-predictions, the matching matrix (*X*) is constructed using the confusion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

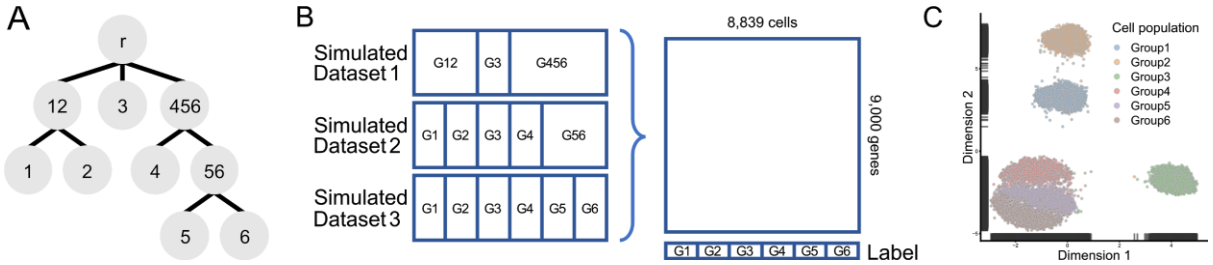

Figure 3. Simulated dataset. (A) Classification tree for the simulated dataset. **(B)** We simulated three datasets separately and concatenated them in one dataset. The labels and their proportion are indicated in the simulated datasets. **(C)** UMAP of the final dataset.

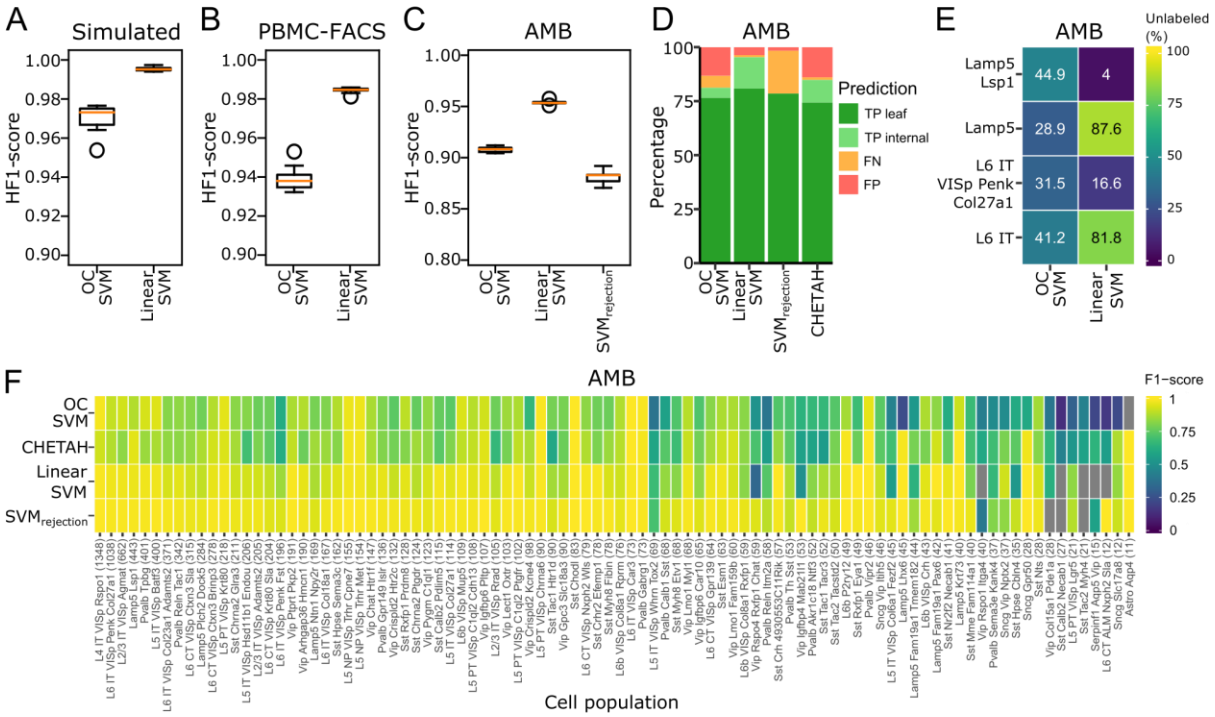

Figure 4. Classification performance. (A-C) Boxplots showing the HF1-score of the one-class and linear SVM on the **(A)** simulated **(B)** PBMC-FACS, and **(C)** AMB dataset. **(D)** Barplot showing the percentage of true positives (TP), false negatives (FN), and false positives (FP) per classifier 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.

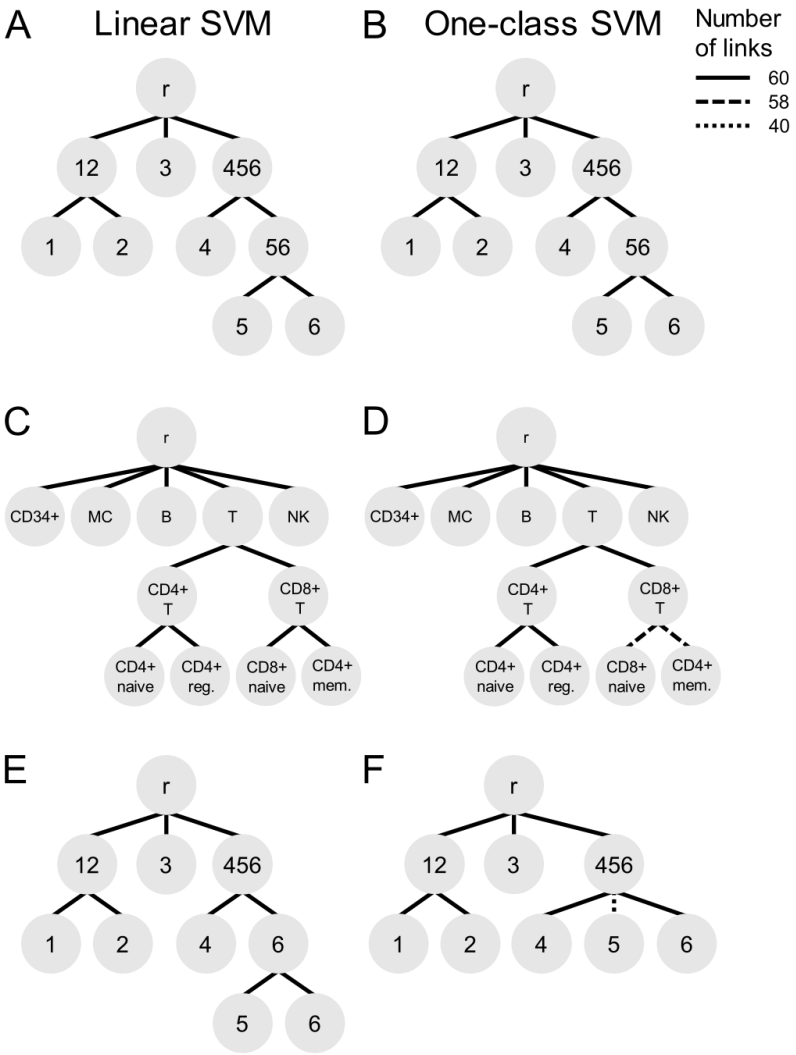

Figure 5. Tree learning evaluation. Classification trees learned when using a **(A,C,E)** linear SVM or **(B,D,F)** one-class SVM during the **(A-B)** simulated, **(C-D)** PBMC-FACS, and **(E-F)** simulated rejection experiment. The line pattern of the links indicates how often that link was learned during the 60 training runs.

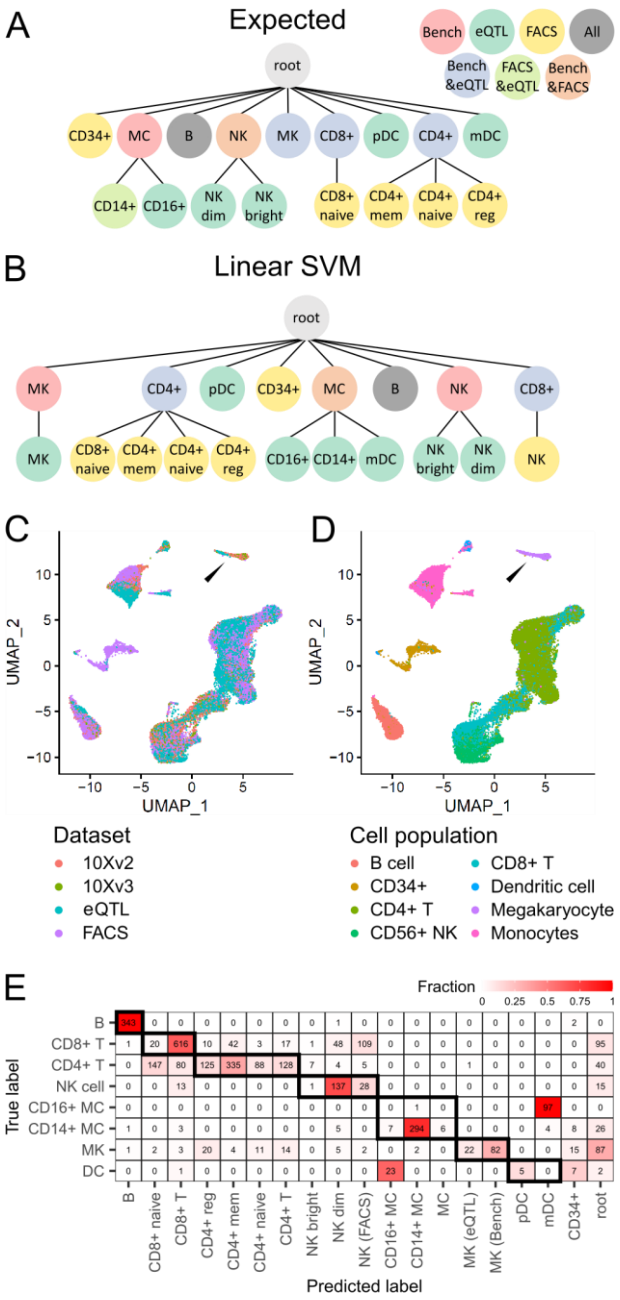

Figure 6. Inter-dataset evaluation. (A) Expected and **(B)** learned classification tree when using a linear SVM on the PBMC datasets. The color of a node represents the agreement between dataset(s) regarding that cell population. **(C-D)** UMAP of the aligned datasets colored by **(C)** dataset and **(D)** cell populations. The arrowhead points to the megakaryocytes, which are a clear example of why the learned tree is supported by the UMAP. **(E)** Confusion matrix when using the learned classification tree to predict the labels of PBMC-Bench10Xv3.

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			