Advance Access Publication Date: Day Month Year
Manuscript Category

Subject Section

Hierarchical progressive learning of cell identities in single-cell data

Lieke Michielsen [1,2,3], Marcel J. T. Reinders [1,2,3] and Ahmed Mahfouz [1,2,3,*]

[1] Leiden Computational Biology Center, Leiden University Medical Center, Leiden, The Netherlands, [2] Delft Bioinformatics Laboratory, Delft University of Technology, Delft, The Netherlands and [3] Department of Human Genetics, Leiden University Medical Center, Leiden, The Netherlands

[*] To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: In single-cell RNA-sequencing datasets, cell identification is mainly done manually, which is subjective and time-consuming. As a consequence, most datasets are annotated at a different resolution. This is not surprising as cell populations form a hierarchy, but it can be problematic for downstream analysis or comparison of datasets. Several supervised methods have been developed to overcome the drawbacks of unsupervised learning, but none of these combines the information of multiple datasets and preserves the old definition of the cell populations in each dataset.
Results: To overcome these challenges, we developed a hierarchical progressive learning method which automatically finds relationships between populations of multiple datasets and uses this to construct a classification tree. We implemented the tree with a one-class and linear SVM for each node and evaluated the classification performance, including the rejection option, and tree construction. At the moment, choosing between a one-class and linear SVM is a trade-off between the ability of discovering new cell populations and a higher accuracy. Both the one-class and linear SVM also outperform other hierarchical classifiers. Furthermore, we show that it is possible to construct a correct classification tree for immune cells when combining three PBMC datasets and predict the labels of the fourth dataset with high accuracy.
Availability and implementation: The pipeline is implemented in Python and available at Github (https://github.com/lcmmichielsen/hierarchicalprogressivelearning).
Contact: a.mahfouz@lumc.nl

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 (van der Wijst *et al.*, 2020). Currently, cells in single-cell RNA-sequencing (scRNA-seq) datasets are primarily annotated using clustering and visualisation 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 (Zeisel *et al.*, 2018). 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 (Svensson and Beltrame, 2019; Tasic *et al.*, 2016, 2018).

Recently, many supervised methods have been developed to replace unsupervised techniques. The underlying principles of these methods vary greatly. Some of these 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) (Zhang *et al.*, 2019; Pliner *et al.*, 2019; Kiselev *et al.*, 2018; Cao *et al.*, 2019; Alquicira-Hernandez *et al.*, 2019; de Kanter *et al.*, 2019).

Supervised methods rely either on a reference atlas or a 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. Aberrant cell populations in particular might be missing as a huge number of diseases exist. Furthermore, mutations could always 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 (Wang *et al.*, 2019). 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

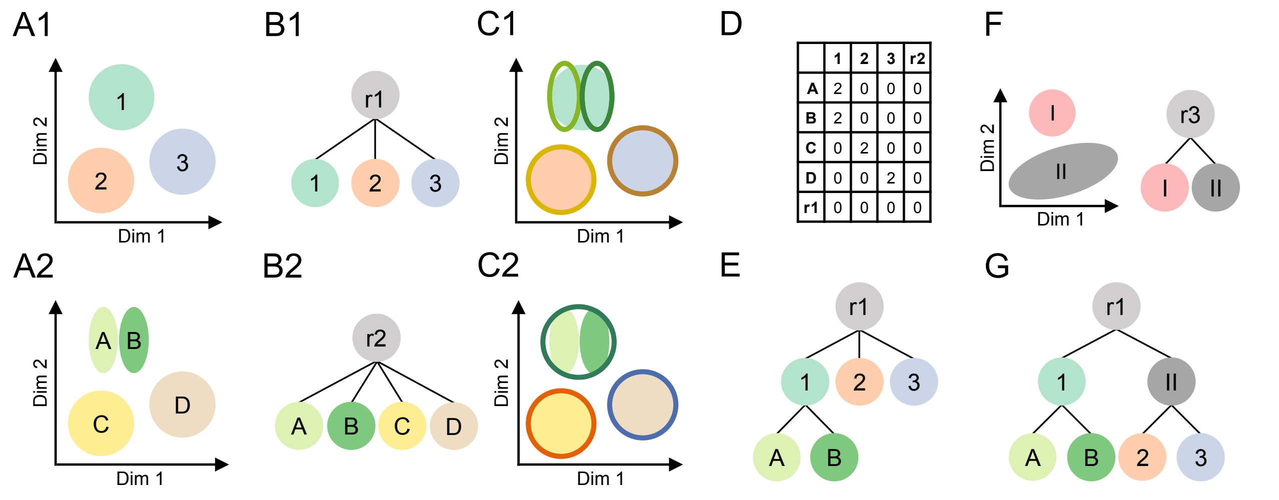


Fig. 1. Schematic overview of the hierarchical progressive learning pipeline. (A1-A2) We start with two labeled datasets. (B1-B2) For both datasets a flat classification tree is constructed. Using this tree and the corresponding dataset, a classifier is trained for each node in the tree except for the root. (C1-C2) We use the trained classification tree of one dataset to predict the labels of the other. The classification tree in B2 is thus used to predict the labels of A1 (C1) and vice versa (C2). (D) We compare the predicted labels to the cluster labels and summarize these results in matching matrix X, which is used to find matches between the labels of the two datasets. (E) The tree belonging to the first dataset (B1) is updated according to these matches. In the dataset A2, for example, subpopulations of population ‘1’ of dataset A1 are found. Therefore, these cell populations, ‘A’ and ‘B’, are added as children to the ‘1’ population. (F-G) If more labeled datasets are available, the same process can be repeated.

should be added to the knowledge of the classifier. We advocate that this can be realized by using progressive learning, a learning strategy inspired by humans. Human learning is a continuous process that never ends (Jarvis, 2006). 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 (Yang and Asada, 1996; Fayek, 2019). This way of learning allows combining the information of multiple existing datasets and retaining the possibility to add more datasets afterwards. This strategy, however, 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 (Svensson and Beltrame, 2019). 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 could be done with a hierarchical classifier. This has the advantage that the cell population definition of multiple datasets can be combined, which gives 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 a large number of classes, while if we exploit the hierarchy, the classification problem is actually 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 could always label it as an intermediate cell population (i.e. at a lower resolution). This way, we still gain some knowledge about this cell.

Currently, some classifiers, such as Garnett, CHETAH, and Moana, already exploit this hierarchy between classes (Pliner *et al.*, 2019; de Kanter *et al.*, 2019; Wagner and Yanai, 2018). Garnett and Moana both depend on prior knowledge about these hierarchies in the form of marker genes for each of the different classes. Especially for deeper annotated datasets it can be difficult to define such marker genes for each cell population that are robust across scRNA-seq datasets (Bakken *et al.*, 2018; Aevermann *et al.*, 2018). Moreover, we have previously shown that adding prior knowledge is not beneficial (Abdelaal *et al.*, 2019). 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 outperformed CHETAH (Abdelaal *et al.*, 2019) indicating that a successful strategy to exploit this hierarchy is still mising.

Here, we propose a hierarchical progressive learning approach to learn cell population identities from multiple annotated datasets and identify relationships between them (Figure 1). During the training phase, we exploit the different resolutions of the datasets (Figure 1A). We match the labels of these datasets by learning a classifier on one dataset and using it to predict the labels of another dataset (Figure 1B-C). Reciprocal matches between the labels are used to update the classification tree (Figure 1E). The updated tree and both datasets are used to retrain the classifier. If there is an additional dataset, we can again match the labels and update the tree (Figure 1F-G). This continues until there are no more labeled datasets. The final classification tree can be used to annotate cells in a new dataset.

Even if we combine multiple datasets, there might be unseen populations in an unlabeled dataset. Identifying these cells as a new population is a challenging problem. Even though many methods have implemented an option to reject cells, they usually fail when being tested in a realistic scenario (Abdelaal *et al.*, 2019). In most cases, the rejection option is implemented by setting a threshold on the posterior probability (Pliner *et al.*, 2019; Abdelaal *et al.*, 2019; Alquicira-Hernandez *et al.*, 2019; Boufea *et al.*, 2020). 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 will test our progressive learning strategy with two different classifiers: a linear and a one-class support vector machine (SVM). The linear SVM had a high performance when benchmarking scRNA-seq classifiers (Abdelaal *et al.*, 2019). However, we noticed that the linear SVM does not have a good working rejection option. We hypothesize that a one-class SVM might be able to solve this problem since it only uses positive training samples to fit a tight decision boundary around (Tax, 2001).

To summarize, our contributions are: (i) we propose a progressive learning approach to merge differently annotated data sets into one classification framework, (ii) we exploit the hierarchical relationships between cell populations when progressively merging the data sets, (iii) and we include a one-class classifier to be able to discover new cell (sub)populations.

2 Methods

2.1 Hierarchical progressive learning

2.1.1 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 or binary classifier. For the one-class, we used the one-class SVM (`svm.OneClassSVM(nu = 0.05)`) from the scikit-learn library in Python (Pedregosa *et al.*, 2011). A one-class classifier only uses positive training samples, which include samples from the node itself and its child nodes. To avoid overfitting, we first select the principal components (PCs) which explain 90% of the variance of the 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. If so, the five PCs with the smallest p-values were selected. For the binary classifier, we used the `svm.LinearSVC()` function from the scikit-learn library. This classifier also uses negative samples. These are selected using the siblings policy (Fagni and Sebastiani, 2007), 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 dimensionality reduction techniques were used.

2.1.2 Predicting the labels

To predict the label of a new cell, we start at the root node, which we denote as parent node. We use the local classifiers of its children to predict the label of the cell using the `predict()` function, and score it using `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.

2.1.3 Reciprocal matching labels and updating the tree

When merging two classifiers trained on different datasets, we use the classifier trained on dataset 1, $D1$, to predict the labels of $D2$ and vice versa (Figure 1C). We construct confusion matrices, $C1$ and $C2$, for $D1$ and $D2$, respectively. Here, $C1_{ij}$ indicates how many cells of population i (P_i) of $D1$ are predicted to be P_j of $D2$. This prediction can be a leaf node, internal node or a rejection. These matrices are used to find matches between the cell populations of the two datasets and update the classification tree (Algorithm 1). The values in $C1$ and $C2$ are highly depended on the size of a cell population. Therefore, we normalize the rows such that the sum of every row is one ($NC1_{ij} = \frac{C1_{ij}}{\sum_{\forall j} C1_{ij}}$) (line 2, Algorithm 1). Consequently, for each cell population, we now know what fraction is predicted to be a certain cell population. High values in $NC1$ and $NC2$ are most interesting since these indicate that a high fraction of a cell population of $D1$ is predicted to be a cell population of $D2$, or vice versa.

To establish reciprocal matching between the two datasets, we convert $NC1$ and $NC2$ into binary confusion matrices $BC1$ and $BC2$, where we indicate matches between cell populations with a 1 (line 3). By realizing that the rows in $NC1$ represent the different cell populations in $D1$ for which we want to find a match in $D2$, that the columns represent the different cell populations in $D2$, and that a value indicates the fraction of cells of a cell population in $D1$ that are predicted a cell population in $D2$, we can match a cell population in $D1$ to a cell population in $D2$ by selecting the column with the highest value as this indicates the best match. Therefore, for every row i the index of the maximum value of $NC1_i$ will be set to 1 in $BC1_i$ (line 13). Next, we look if a cell population from $D1$ has matches to multiple cell populations in $D2$. If the difference between the highest and the second highest value of a row in $NC1$ is smaller than a threshold (default = 0.25), and if this second highest value is also higher than 0.1, the corresponding value in $BC1_i$ is also set to 1 (line 15-17). This indicates that a population from $D1$ thus matches two populations from $D2$. If so, the third highest value is compared with the second highest value etc. The multiple matching populations from $D2$ might for instance be subpopulations of the cell population from $D1$ (Figure 1). The same procedure is applied to $NC2$ to construct $BC2$

Next, we combine $BC1$ and $BC2$ into matching matrix X ($X = B1^T + B2$) (Figure 1D). 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 P_i from $D2$ and P_j from $D1$. If $X_{ij} = 1$, this is indicates a one-sided match. A zero indicates that there is no match between the two cell populations.

Finally, we iterate over X to match the labels of the two datasets and update the tree (line 5). If we encounter a non-zero value in X, we check whether there are other non-zero values in the corresponding row and column to identify the scenario at hand. Table 1 and Figure 2 explain the five most common scenarios: a perfect match, splitting nodes, merging

Algorithm 1 Pseudocode to match the labels of two datasets and update the classification tree.

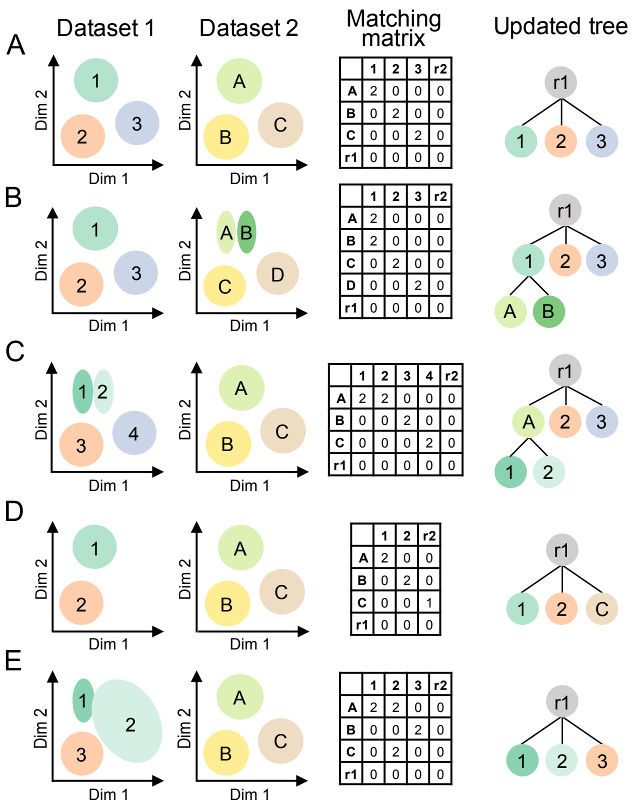
```
1: function MatchLabels(matrix $C1$, matrix $C2$, threshold $t$, tree $T$)
2:     $NC_1 \leftarrow \frac{C1}{\mathrm{sum}(C1, \mathrm{axis} = 1)}$, $NC_2 \leftarrow \frac{C2}{\mathrm{sum}(C2, \mathrm{axis} = 1)}$
3:     $BC_1 \leftarrow$ FillBinary($NC_1, t$), $BC_2 \leftarrow$ FillBinary($NC_2, t$)
4:     $X \leftarrow BC_1' + BC_2$
5:     $X, T \leftarrow$ Iterate($X, T$)
6:     if $X \neq 0$ then
7:         $X \leftarrow X > 1$
8:         $X, T \leftarrow$ Iterate($X, T$)
9:     return $T$

10: function FillBinary(matrix $NC$, threshold $t$)
11:     matrix $BC \leftarrow 0$
12:     for $i$, $r$ in enumerate($NC$) do
13:         $BC_{i,\arg\max(r,1)} \leftarrow 1$
14:         counter $x \leftarrow 2$
15:         while $\max(r, x-1) - \max(r, x) < t$ & $\max(r, x) > 0.1$ do
16:             $BC_{i,\arg\max(r,x)} \leftarrow 1$
17:             $x{+}{+}$
18:     return $BC$

19: function Iterate(matrix $X$, tree $T$)
20:     for every element $i$ in $X$ do
21:         if $i \neq 0$ then
22:             $S \leftarrow$ FindScenario($X, T, i$)
23:             $T \leftarrow$ UpdateTree($T, S$)
24:             $X \leftarrow$ RemoveValues($X, S$)
25:     return $X, T$
```

Table 1. Explanation of the five most common scenarios

Scenario	Biological explanation	Identification in X	Solution
Perfect	P_i of $D1$ exactly matches P_j from $D2$	$X_{j,i} = 1$ or 2, rest of row j and column i is zero	Rename labels of P_j to P_i
Splitting	Multiple populations from $D2$ are sub-populations of P_i of $D1$	Multiple non-zero values in column i of X	Add corresponding populations from $D2$ as children to P_i
Merging	Multiple populations from $D1$ are sub-populations of P_j of $D2$	Multiple non-zero values in row j of X	Add P_j as a parent to corresponding populations from $D1$
New	$D2$ contains a new population, P_j.	$X_{j,root1} = 1$, rest of row j is zero	Add P_j as a child to the root node
Impossible		$X_{j,i} = 1 \vee 2$, two or more non-zero values in row j and column i	Skip



	1	2	3	r2
A	2	0	0	0
B	0	2	0	0
C	0	0	2	0
r1	0	0	0	0

	1	2	3	r2
A	2	0	0	0
B	2	0	0	0
C	0	2	0	0
D	0	0	2	0
r1	0	0	0	0

	1	2	3	4	r2
A	2	2	0	0	0
B	0	0	2	0	0
C	0	0	0	2	0
r1	0	0	0	0	0

	1	2	r2
A	2	0	0
B	0	2	0
C	0	0	1
r1	0	0	0

	1	2	3	r2
A	2	2	0	0
B	0	0	2	0
C	0	2	0	0
r1	0	0	0	0

Fig. 2. Schematic example of a (A) perfect match, (B) splitting, (C) merging, (D) new, and (E) impossible scenario. For each scenario, we show a two dimensional representation of the cell populations in the datasets, the matching matrix X and the updated tree. For the impossible scenario, the updated tree is the tree belonging to dataset 1, since the tree could not be updated based on X.

nodes, a new population, and impossible matching. All other scenarios are explained in Supplementary Note 1. If we need to update the tree, we always update the tree belonging to $D1$ (Figure 1E, G). After an update, the corresponding values are removed from X. If the matching is impossible, the corresponding values in X are thus not set to zero. If we have iterated over X, and there are still non-zero values, we will change X into a strict matrix, i.e. we only consider reciprocal matches now, so all ‘1’s are turned into a ‘0’. We again iterate over the matrix to see if we can solve impossible matches now (lines 6-8).

2.2 Evaluation

2.2.1 Hierarchical F1-score

We use the hierarchical F1-score (HF1-score) to evaluate the performance of the classifiers (Kiritchenko and Famili, 2005). 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 the 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: $HF1 = \frac{2hP \cdot hR}{hP + hR}$.

2.2.2 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.

2.2.3 Classification tree evaluation

We evaluated the classification tree construction by constructing multiple trees with different orders of datasets and counting how often each link occurred. These trees, in which every edge has a number indicating how often this link was seen, were drawn and compared to the ground truth.

2.3 Datasets

2.3.1 Simulated data

We used the R-package Splatter version 1.6.1 to simulate a hierachical scRNA-seq dataset that represents the tree shown in Figure 3A (Zappia *et al.*, 2017). 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 columnwise. 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 ($log_2(x+1)$). A UMAP embedding of the simulated dataset shows it indeed represents the desired hierarchy (Figure 3C).

2.3.2 PBMC data

We used four different PBMC datasets: PBMC-FACS, PBMC-Bench10Xv2, PBMC-Bench10Xv3, and PBMC-eQTL. All were preprocessed as described in (Abdelaal *et al.*, 2019).

The PBMC-FACS dataset is the downsampled FACS-sorted PBMC dataset from (Zheng *et al.*, 2017). 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 (Zheng *et al.*, 2017). 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 (Ding *et al.*,

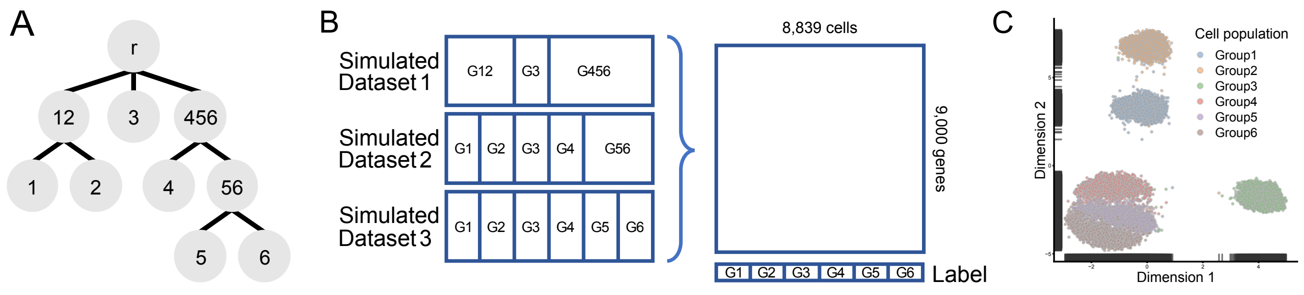


Fig. 3. (A) Classification tree for the simulated dataset. (B) The three separately simulated datasets and how they are concatenated into one dataset. The labels and their proportion are indicated in the simulated datasets. (C) UMAP of the final dataset.

2019). 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 (Van Der Wijst *et al.*, 2018).

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 (Stuart *et al.*, 2019).

2.3.3 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 (Tasic *et al.*, 2018) and preprocessed as described in (Abdelaal *et al.*, 2019), consists of 12,771 cells and 42,625 genes.

3 Results

3.1 Linear SVM has a higher 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 the simulated, PBMC-FACS, and AMB dataset using 10, 10, and 5 fold cross-validation respectively. The classifiers were trained on predefined trees (Figure 3A, S1-2), so the progressive learning was not applied here.

On all datasets, the performance of the linear SVM is higher than the performance of the one-class SVM (Figure 4A-D). The simulated dataset is relatively easy since the different cell populations are widely separated (Figure 3C). The linear SVM shows a perfect performance here. The one-class SVM labels 93.6% of the cells correctly, the rest is labeled as an internal node (2.3%) or rejected (4.1%), which results in a median HF1-score of 0.977. Interestingely, the one-class SVM has a better performance on the test data compared to the training data, where 3.7% was labeled as an internal node and 5.0% was rejected. This is in agreement with μ, an upperbound on the fraction of training errors, used when training the one-class SVM ($\mu = 0.05$).

As expected, the performance of the classifiers on real data drops, but the HF1-scores all 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 (Abdelaal *et al.*, 2019), which enables us to compare our results. When comparing to CHETAH, which allows hierarchical classification, we notice that both the one-class and linear SVM have a better performance based on the median F1-score (0.81 vs 0.86 and 0.97 respectively). In this case, the HF1-scores cannot be compared, since CHETAH uses a different classification tree. It is not fair to compare these HF1-scores as this can be influenced by, for instance, the depth of the tree.

The linear SVM also shows a better performance compared to SVM$_{\text{rejection}}$, which was the best classifier for this data (Abdelaal *et al.*, 2019). SVM$_{\text{rejection}}$, however, has a slightly higher median F1-score (0.98 vs 0.97). This is mainly because it makes almost no mistakes, only 1.7% of the cells is 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$_{\text{rejection}}$ (19.8%). The linear SVM, on the contrary, has no rejected cells, which is also reflected in a higher HF1-score (Figure 4C). Because of this large amount of rejections of SVM$_{\text{rejection}}$, the one-class SVM also has a higher HF1-score.

The AMB dataset contains 92 different cell populations ranging in size from 11 to 1,348 cells. The performance of both classifiers decreases when the number of cells per cell population becomes smaller, but the performance of the one-class SVM is affected more (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.

3.2 One-class detects new cells better than linear SVM

Besides a high accuracy, it is also important that the classifiers can detect 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 correctly rejects all these cells, while the linear SVM rejects none. Here, the one-class SVM clearly outperforms the linear SVM.

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 detects these cells better as a new population than the one-class SVM (Figure 4E). On the contrary, the one-class SVM shows a higher performance when a leaf node is removed.

3.3 Correct reconstruction of classification trees

Next, we tested whether it is possible to construct the classification trees for the simulated and PBMC-FACS data using the hierarchical progressive learning. During both experiments, we performed a 10 fold

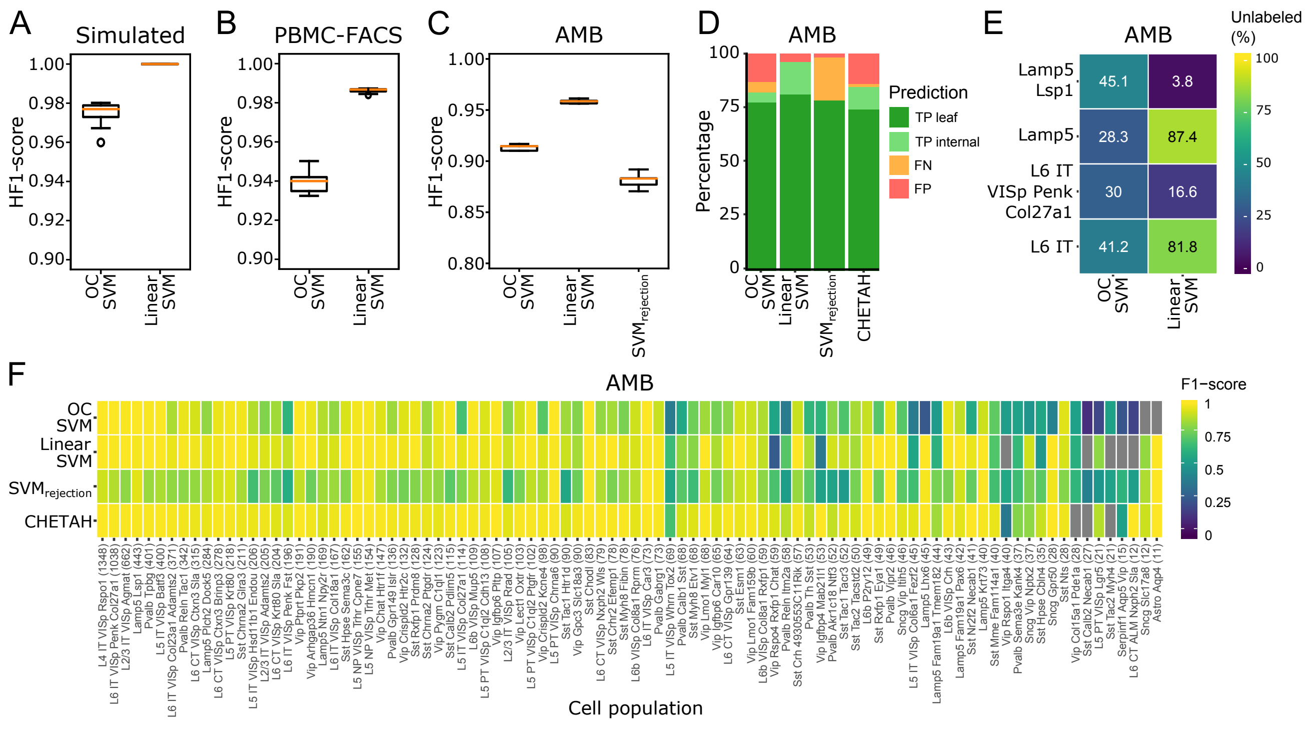


Fig. 4. (A-C) Boxplots showing the HF1-score of the one-class and linear SVM on the (A) simulated (B) PBMC-FACS, and (C) AMB dataset. (D) Barplot showing the percentage of true positives (TP), false negatives (FN), and false positives (FP) per classifier on the AMB dataset. For the TPs a distinction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

cross-validation and split 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 some cell populations were removed from certain batches (Tables S1-2). Batch 1 contains the lowest resolution and Batch 3 the highest. When constructing the trees, we try all (six) different orders of the batches to see whether this affects the tree construction. Combining this with the 10 fold cross-validation, 60 trees were constructed in total by each classifier. A final tree was constructed in which for each edge it was counted in how many of the 60 trees it was learned.

Using a linear SVM, the final tree for the simulated data always looks as we expected (Figure 5A). This classifier is thus stable and the order of the batches does not matter. The one-class SVM, on the other hand, makes some mistakes. In respectively 14, 14, and 12 experiments, ‘Group1’, ‘Group2’, and ‘Group4’ are added to the tree twice (Figure 5B). This only occurs 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.

On the PBMC-FACS dataset, the one-class SVM shows slightly more stable results than the linear SVM (Figure 5C-D). For the one-class SVM, all 60 constructed trees look similar, while the linear SVM shows little variation. During four experiments, the CD8+ naive T-cells and CD4+ memory T-cells are connected directly to the T-cells, bypassing the internal node containing the CD8+ T-cells.

The trees constructed using the linear and one-class SVM, however, are both slightly different from the predefined hematopoietic tree (Figure S1A). In the constructed trees, CD4+ memory T-cells are a subpopulation of CD8+ instead of CD4+ T-cells. A t-SNE plot of the PBMC-FACS dataset also confirms that CD4+ memory T-cells are more similar to CD8+ than CD4+ T-cells (Figure S1B). Rewiring this node also improves the classification performance of the linear SVM slightly (HF1-score = 0.992 vs 0.987). Using the hematopoietic tree, the CD4+ memory T-cells, CD8+ T-cells, and CD8+ naive T-cells were also confused mostly, while now less mistakes are made (Tables S3-4).

3.4 Missing populations affect construction with linear SVM

We tested whether new or missing cell populations in the training set could influence tree construction. We mimicked this scenario using the simulated dataset and the same batches as in the tree construction experiment of section 3.3. In this experiment, ‘Group5’ and ‘Group6’ were merged into ‘Group56’ in Batch 2, but now we removed ‘Group5’ completely from this batch (Table S5). Here, the linear SVM misconstructs 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’. Therefore, ‘Group6’ is added as a parent node to ‘Group5’ and ‘Group6’. The one-class SVM, on the other hand, has no problems with the missing cell population (Figure 5F). Some trees (14 out of 60) are not constructed correctly, but this is similar to the results shown in section 3.3.

3.5 Linear SVM can reconstruct classification tree during inter-dataset experiment

Finally, we tested the pipeline in a realistic scenario by using three PBMC datasets, PBMC-eQTL, PBMC-Bench10Xv2, and PBMC-FACS, to construct a classification tree and using this tree to predict the labels of a fourth dataset, PBMC-Bench10Xv3 (Table 2). We constructed an expected classification tree based on the names of the cell populations in the datasets (Figure 6A). This type of matching, however, could result in an erroneous tree since every dataset was labeled using different clustering techniques, marker genes, and their own name conventions.

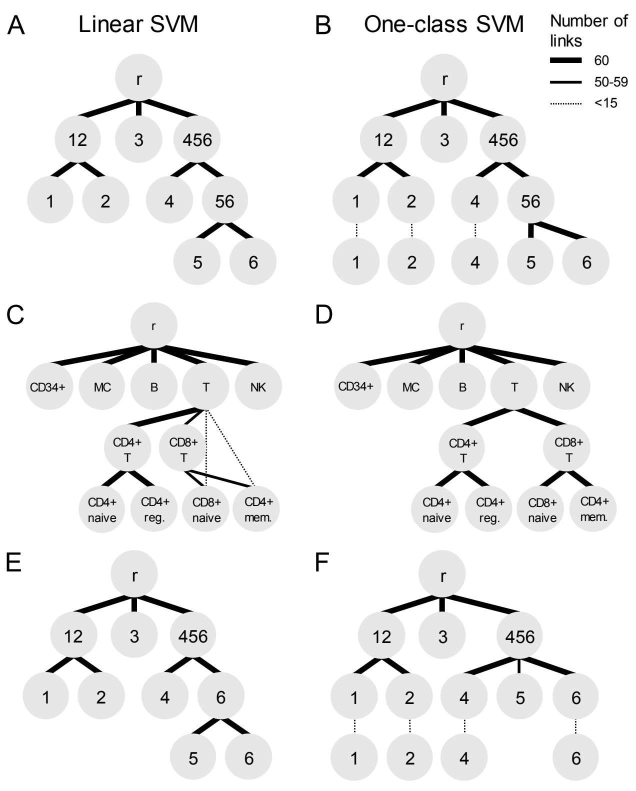


Fig. 5. Classification trees constructed when using a (A,C,E) linear SVM or (B,D,F) one-class SVM during the (A-B) simulated, (C-D) PBMC-FACS, and (E-F) simulated rejection experiment. The thickness of the links indicates how often that link was added to the classification tree during the 60 training runs.

Table 2. Number of cells per cell population in the different training datasets (batches) and test dataset. Subpopulations are indicated using an indent.

Cell population	Batch1 eQTL	Batch2 Bench 10Xv2	Batch3 FACS	Testset Bench 10Xv3
CD19+ B	812	676	2,000	346
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				38
Plasmacytoid (pDC)	101			
Myeloid (mDC)	455			
CD34+			2,000	

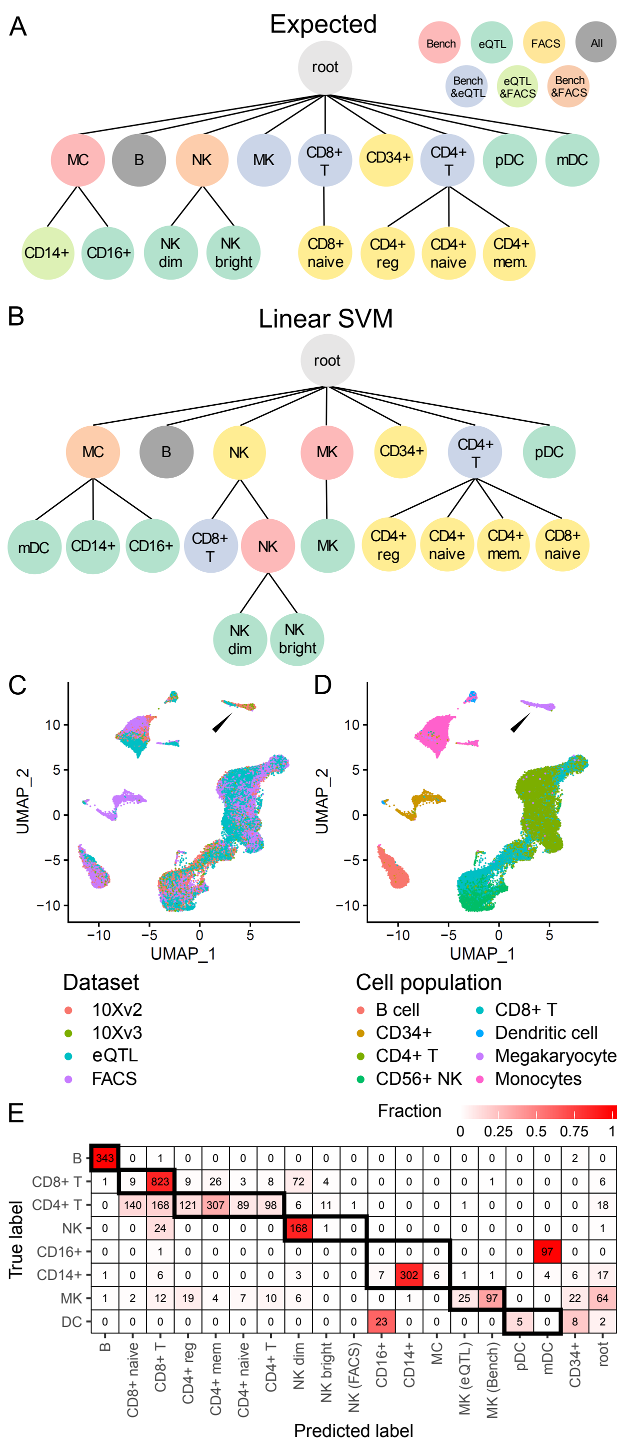


Fig. 6. (A) Expected and (B) reconstructed classification tree when using a linear SVM on the PBMC datasets. The color of a node represents the agreement between dataset(s) regarding that cell population. (C-D) UMAP of the aligned datasets colored by (C) dataset and (D) cell populations. The arrowhead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

When comparing the tree constructed 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 (Van Der Wijst *et al.*, 2018; Ding *et al.*, 2019). 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 can be used for the CD14+ monocytes from the FACS dataset. Instead of matching the CD14+ monocytes from the eQTL dataset, they matched the monocytes from the Bench10Xv2 dataset, which includes CD14+ and CD16+ monocytes. 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 (León *et al.*, 2005).

There are two mistakes, however, that cannot be explained biologically. The NK cells from the FACS dataset do not only match the NK cells from the eQTL and Bench10Xv2 dataset, but also the CD8+ T-cells. As a consequence, the NK cells from the FACS dataset become the parent of these populations. 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 constructed tree, it is unclear whether these mistakes are introduced by the alignment 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 constructed classification tree (Figure 6E). Interestingly, we notice that the CD16+ monocytes are predicted to be mDCs and vice versa, which might be explained by the fact that monocytes can differentiate into dendritic cells.

When using a one-class SVM instead of a linear SVM, more mistakes are made (Figure S3). Since the one-class SVM shows a low performance on cell populations with a few number of cells, it also makes more mistakes when trying to match them. For instance, the NK dim and bright cells from the eQTL dataset are almost all predicted to be CD8+ T-cells instead of NK cells. This is probably since there are very little NK cells in the 10Xv2 dataset, so the one-class SVM cannot define the decision boundary for this population properly. Larger cell populations, such as CD4+ T-cells and B-cells, can be matched without mistake when using the one-class SVM.

4 Discussion

In this study, we showed that hierarchical progressive learning has great potential for automatic cell identification in scRNA-seq data. In particular, we showed that using a hierarchical approach outperforms flat classification. On the AMB dataset, the linear SVM outperformed SVM$_{\text{rejection}}$, which was the best performing classifier on this dataset (Abdelaal *et al.*, 2019). In contrast to SVM$_{\text{rejection}}$, 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.

Furthermore, we showed that using our approach the labels of three different PBMC datasets can succesfully be matched to construct a classification tree. During this experiment, small mistakes, such as the imperfect match of 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 of the datasets. PBMC-FACS was labeled based on the expression of cell surface markers instead of clustering the cells. Protein and gene expression are known to be poorly correlated (Berg *et al.*, 2017; Vogel and Marcotte, 2012), which might explain these results.

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

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

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, in contrast to the one-class SVM, cannot reconstruct the correct tree anymore. This again confirms that the one-class SVM is preferred when cell populations are missing. On the AMB dataset, the performance of both classifiers could be improved.

5 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 reconstruct 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 could be used to build the hierarchical tree, which could subsequently be used to annotate cells in the other datasets.

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)

References

Abdelaal, T. *et al.* (2019). A comparison of automatic cell identification methods for single-cell RNA sequencing data. *Genome Biology*, **20**(1), 194.

Aevermann, B. D. *et al.* (2018). Cell type discovery using single-cell transcriptomics: implications for ontological representation. *Human Molecular Genetics*, **27**(R1), R40–R47.

Alquicira-Hernandez, J. *et al.* (2019). ScPred: Accurate supervised method for cell-type classification from single-cell RNA-seq data. *Genome Biology*, **20**(1), 264.

Bakken, T. E. *et al.* (2018). Single-nucleus and single-cell transcriptomes compared in matched cortical cell types. *PLOS ONE*, **13**(12), e0209648.

Berg, P. R. v. d. *et al.* (2017). Dynamic post-transcriptional regulation during embryonic stem cell differentiation. *bioRxiv*, page 123497.

Boufea, K. *et al.* (2020). scID Uses Discriminant Analysis to Identify Transcriptionally Equivalent Cell Types across Single-Cell RNA-Seq Data with Batch Effect. *iScience*, **23**(3), 100914.

Cao, Z.-J. *et al.* (2019). Cell BLAST: Searching large-scale scRNA-seq databases via unbiased cell embedding. *bioRxiv*, page 587360.

de Kanter, J. K. *et al.* (2019). CHETAH: a selective, hierarchical cell type identification method for single-cell RNA sequencing. *Nucleic Acids Research*, **47**(16), e95–e95.

Ding, J. *et al.* (2019). Systematic comparative analysis of single cell RNA-sequencing methods. *bioRxiv*, page 632216.

Fagni, T. and Sebastiani, F. (2007). On the Selection of Negative Examples for Hierarchical Text Categorization. *Proceedings of the 3rd language technology conference*, pages 24–28.

Fayek, H. M. (2019). *Continual Deep Learning via Progressive Learning*. Ph.D. thesis, RMIT University.

Jarvis, P. (2006). *Towards a Comprehensive Theory of Human Learning*. Taylor & Francis Ltd.

Kiritchenko, S. and Famili, F. (2005). Functional Annotation of Genes Using Hierarchical Text Categorization. *Proceedings of BioLink SIG, ISMB*.

Kiselev, V. Y. *et al.* (2018). scmap: projection of single-cell RNA-seq data across data sets. *Nature Methods*, **15**, 359.

León, B. *et al.* (2005). Monocyte-derived dendritic cells. *Seminars in Immunology*, **17**(4), 313–318.

Pedregosa, F. *et al.* (2011). Scikit-learn: Machine Learning in Python. Technical report.

Pliner, H. A. *et al.* (2019). Supervised classification enables rapid annotation of cell atlases. *Nature Methods*, pages 1–4.

Stuart, T. *et al.* (2019). Comprehensive Integration of Single-Cell Data. *Cell*, **177**(7), 1888–1902.

Svensson, V. and Beltrame, E. d. V. (2019). A curated database reveals trends in single cell transcriptomics. *bioRxiv*, page 742304.

Tasic, B. *et al.* (2016). Adult mouse cortical cell taxonomy revealed by single cell transcriptomics. *Nature neuroscience*, **19**(2), 335–46.

Tasic, B. *et al.* (2018). Shared and distinct transcriptomic cell types across neocortical areas. *Nature*, **563**(7729), 72–78.

Tax, D. (2001). *One-class classification Concept-learning in the absence of counter-examples*. Ph.D. thesis, TU Delft.

Van Der Wijst, M. G. *et al.* (2018). Single-cell RNA sequencing identifies celltype-specific cis-eQTLs and co-expression QTLs. *Nature Genetics*, **50**(4), 493–497.

van der Wijst, M. G. *et al.* (2020). The single-cell eQTLGen consortium. *eLife*, **9**.

Vogel, C. and Marcotte, E. M. (2012). Insights into the regulation of protein abundance from proteomic and transcriptomic analyses. *Nature Reviews Genetics*, **13**(4), 227–232.

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

Wang, S. *et al.* (2019). Unifying single-cell annotations based on the Cell Ontology. *bioRxiv*, page 810234.

Yang, B. H. and Asada, H. (1996). Progressive learning and its application to robot impedance learning. *IEEE Transactions on Neural Networks*, **7**(4), 941–952.

Zappia, L. *et al.* (2017). Splatter: simulation of single-cell RNA sequencing data. *Genome Biology*, **18**(1), 174.

Zeisel, A. *et al.* (2018). Molecular Architecture of the Mouse Nervous System. *Cell*, **174**(4), 999–1014.

Zhang, Z. *et al.* (2019). SCINA: Semi-Supervised Analysis of Single Cells in Silico. *Genes*, **10**(7), 531.

Zheng, G. X. Y. *et al.* (2017). Massively parallel digital transcriptional profiling of single cells. *Nature Communications*, **8**, 14049.