Supplementary Methods

Description of the *Moana* framework

The *Moana* framework consists of methods for clustering and cell type annotation, classification, and validation, all designed to address the specific challenges posed by high-throughput single-cell RNA-Seq data. These challenges consist primarily of 1) the high-dimensionality of the data, 2) the high levels of technical noise, and 3) the increasingly large scale of scRNA-Seq datasets, now routinely comprising more than ten thousand cells. Our general approach to address each of these challenges is described the following paragraphs.

Fundamental assumptions

Our overall approach relies on a few fundamental assumptions that we would like to state explicitly here. First, we assume that the expression data on which we perform clustering and classification (training as well as prediction) represents UMI-filtered read counts (*i.e.*, transcript counts). To our knowledge, all microfluidics- or nanowell-based scRNA-Seq protocols incorporate UMIs, but some popular protocols (*e.g.*, SMART-Seq2) do not[1]. The reason we restrict our framework to UMI-filtered data is that we rely on the assumption that the only sources of technical noise are efficiency noise and sampling noise (see below), while data that is not UMI-filtered also exhibits amplification noise[2].

Second, we assume that whenever a *Moana* classifier is applied to new dataset, the set of genes contained in this dataset is identical to the set of genes contained in the data used for training the classifier. This assumption is necessary because we normalize the transcript counts in each cell to a specific number *C* (the median transcript count across all cells in the training data), and the inclusion (omission) of expressed genes in the new dataset would lead to an increased (decreased) transcript count, and therefore in an artifactual deflation (inflation) of expression values following normalization to *C*. In practice, this assumption can be violated when the new data was processed using a pipeline that relies on different genome annotations, which may not include some of the genes in the training data, and may include additional genes not found in the training data. In some cases, missing genes can be recovered by fixing discrepancies between gene names, and genes not present in the training data (*e.g.*, non-coding genes) can simply be excluded in order to satisfy this assumption.

Third, we would like to emphasize that *Moana* classifiers are not designed to reject any expression profiles as “unknown”. This means that when a classifier is applied to a new dataset which contains a cell type that was not present in the training data, the expected behavior is that cells of this type will be incorrectly classified as one of the cell types known to the classifier. Furthermore, there is no guarantee that all of these cells will be misidentified as the same cell type. Therefore, *Moana* classifiers are highly tissue-specific and designed to detect previously defined cell types, while being tolerant to systematic differences owing to technical factors (different experiment or protocol) or biological perturbations owing to experimental treatments or differences in biological conditions.

General approach

To design an effective approach for reducing the dimensionality of scRNA-Seq data, it is important to consider the specific noise characteristics of scRNA-Seq data. The two main sources of technical noise affecting transcript counts (UMI-filtered reads) originate from the fact that only a small fraction of transcripts from each cell is captured (“sampling noise”), and the fact that overall capture efficiencies

vary across cells ("efficiency noise")[3]. Efficiency noise can be addressed (practically removed) in a straightforward fashion by normalizing the expression profiles to a certain constant *C*. "Normalizing the data to *C* transcripts" means scaling each expression profile so that the sum of its transcript counts, taken over all genes, is *C*. While C is sometimes set to an arbitrary number such as 10^6 or 10^4, we and others have shown that when *C* is chosen as the median transcript count across all cells in the raw data, an approach we refer to as "median-normalization", the remaining technical noise (i.e., the sampling noise component) is approximately Poisson-distributed[3,4]. Consequently, the Freeman-Tukey (FT) transformation ($y = \sqrt{x} + \sqrt{x+1}$) serves as an effective variance-stabilizing transformation of the normalized data, which helps to ensure that genes exhibit comparable absolute levels of variance, independent of their expression level[4,5]. We found that this helps increase the effectiveness of principal component analysis (PCA), meaning that each principal component (PC) captures a larger share of the total variance and is less biased towards genes with a particular expression level. This allowed us to apply PCA without first filtering for highly variable genes, which is a key step in most scRNA-Seq analysis frameworks[6]. The omission of a gene selection step simplifies the analysis of scRNA-Seq data by reducing the number of steps and parameters involved. In summary, we generally apply PCA to the median-normalized and FT-transformed data, without gene filtering, and keep the scores for the first 20 or so principal components. As a result, the dimensionality of the data is approximately reduced by a factor of 1,000, from about 20,000 (genes) to 20 (principal components).

To address the issue of noise in scRNA-Seq data, it is important to be able to quantify the level of technical noise associated with each expression measurement. Since we know that the technical noise in UMI-filtered scRNA-Seq data is approximately Poisson-distributed following median-normalization (see above), it is clear that the measured transcript (UMI) counts for most genes are affected by very high levels of noise. To illustrate this point with an example, we consider scRNA-Seq data obtained from a homogeneous population of cells. For a gene with an average observed transcript count of $\mu = 4$, the standard deviation of the technical noise, as calculated based on the Poisson distribution, is $\sigma = \sqrt{\mu} = 2$. Therefore, the coefficient of variation (CV) for this gene is $\sigma/\mu = 50\%$, indicating a very high level of technical noise. Depending on the observed number of transcripts per cell, an average expression level of 4 could correspond to a very highly expressed gene. In our example, we could assume a (median) total transcript count per cell of 2,000, which is representative to what we observe for T cells in the 10X version 2 datasets analyzed. In this case, an expression level of 4 would correspond to 2,000 TPM, indicating an extremely high expression level. Most genes in a cell can be expected to be expressed well below 2,000 TPM, and would therefore exhibit even higher noise levels (larger than 50% CV) in our example. This example highlights the need to reduce technical noise levels in the analysis of scRNA-Seq data. Several computational approaches for denoising (or, more generally, *smoothing*) of scRNA-Seq data have been proposed, which generally employ different implementations of the idea of improving the signal-to-noise ratio by aggregating data across cells. To perform denoising/smoothing in the context of our framework, we rely on our previously developed *kNN-smoothing* algorithm (version 2), which uses the dimensionality reduction strategy outlined above, in order to identify a set of *k* highly similar cells (neighbors) for each cell in the data (where *k* is defined so that the cell itself is counted as its own closest neighbor). The identification of neighbors is conducted in a stepwise fashion, in order to achieve the highest possible accuracy. Assuming that all identified neighbors originate from the same, homogeneous subpopulation of cells, the algorithm reduces the noise level (CV) associated with each gene expression measurement by a factor of $\sqrt{k}$. Moreover, as the sum of independent Poisson

variables is again a Poisson variable, the technical noise associated with the smoothed (aggregated) transcript counts can still be approximated using the Poisson distribution (again after median-normalization), and this is true whether the neighbors are homogeneous or not[4]. As a result, for dimensionality reduction of scRNA-Seq data smoothed using the *kNN-smoothing* algorithm, we can apply the exact same approach as described above. We can therefore treat smoothing as a simple preprocessing step that we may or may not invoke, depending on the number of cells available to us, the absolute levels of noise observed, and the level of resolution necessary for the analysis.

To address the issue of scale, we rely on a hierarchical approach to clustering and classification. The key idea (or assumption) behind this approach is that since subtypes of cells are usually biologically related, we only need a limited number of cells to distinguish between the main cell types in any given dataset. We can therefore take a random sample of cells from the full dataset, and construct a subclassifier on this subset of cells (see below). Using this subclassifier, we can then efficiently predict the cell type identities for all cells in the original data (specifically those that were excluded from the analysis). We then recursively apply this approach on each subpopulation. In this way, we can train a full hierarchical classifier for all cell types, without ever having to perform clustering or classification on more than a few thousand cells. Given the clustering and classification methods proposed here, our approach is highly scalable in the sense that it stays computationally efficient independent of the total number of cells in the original dataset. However, it assumes that each biologically distinct subpopulation is represented at a certain minimum level of abundance. For the analyses in this work, we generally used random samples of 2,000 cells, and ignored rare subpopulations (<2%). As a result, we for example ignored plasmacytoid dendritic cells (pDCs), which represent a very rare cell population in peripheral derived that is derived from an independent hematopoietic lineage [ref], and therefore did not cluster with either lymphocytes or myeloid cells. For less extreme cases of imbalanced subpopulation sizes, we propose specific approaches to counteract the negative effects of class imbalance on classification accuracy and robustness, one of which involves the use of two clustering/classification cycles that help to create a more balanced training dataset. The details of these approaches can be found under “Classification method” below.

Clustering and cell type annotation method

In the context of our hierarchical Moana framework, the main purpose of clustering is to partition the cells in a dataset into a small number of clusters (typically between two and five), which represent biologically distinct subpopulations of cells. Depending on the number of cell types present in the data, each cluster can either represent a single cell type, a number of biologically related cell types, or span multiple unrelated cell types. In any case, the cluster labels can then be used to train a machine learning classifier, and if a cluster represents more than one cell type, the clustering and classification procedures can be repeated on only the cells belonging to that cluster, eventually giving rise to a hierarchical cell type classifier that distinguishes between individual cell types. To efficiently identify a small number of clusters, we apply kNN-smoothing with a relatively large setting of k (e.g., k=128 for PBMCs - smaller settings of k can be sufficient for cells with higher RNA content), in order to remove as much noise as possible. Depending on the size of the dataset and the abundance of the individual cell types, a large k may induce oversmoothing, meaning that the algorithm aggregates expression values from cells belonging to different cell types. As long as these cell types end up belonging to the same cluster, we do not try to avoid oversmoothing during this step, and may even use it to our advantage (as it can simplify the density-based definition of clusters, see below). Here, our goal in applying smoothing is to reduce

technical noise levels to the point where the first two principal components clearly separate between at least two clusters of cells. After smoothing is completed, the presence or absence of clusters along the first two PCs can easily be assessed visually, using a scatter plot that shows all cells in PC space. When there are biological subpopulations present in the data, we hope to see at least two more or less compact clusters of cells that are linearly separable along the first two PCs.

In applying *kNN-smoothing* to multiple datasets, and particularly to different subsets of PBMCs, we found that *d*, the parameter of the kNN-smoothing algorithm that determines the number of PCs used to identify neighbors in each smoothing step, sometimes requires adjustment in order for the smoothed data to clearly separate into clusters that correspond to specific subpopulations of cells. By default, we attempt to apply kNN-smoothing with *d*=20, which we found to provide good results in many instances. However, in some cases, we found that it was necessary to *reduce d*, using for example to *d*=16 or *d*=12. We speculate that for datasets where the expression differences between subpopulations are small, and noise levels are high, the inclusion of too many PCs reduces the accuracy with which neighbors are identified. We currently do not know of a robust method for automatically determining an appropriate setting of *d* for a given dataset. However, since the Python implementation of the kNN-smoothing algorithm finishes in approximately 30-40 seconds on a modern computer (for *k*=128 and a dataset of 2,000 cells), and given the fact that it is fairly straightforward to verify the presence or absence of biologically meaningful clusters in the smoothed data (see below), we simply try different values of *d* to test if there is any value that results in a good separation.

Once there is clear visual evidence for the presence of clusters in two-dimensional PC space, we apply *DBSCAN* to the two-dimensional data to obtain cluster assignments for all cells. *DBSCAN* is a simple and widely used clustering algorithm that identifies points (here, cells) with more than *n* neighbors, defined as other points located within a certain distance ε. These points with a large number of neighbors are referred to as “core points”, and are used to define clusters as arbitrarily-shaped areas of high density. We set *n* to 2% of the total number of cells in the dataset, requesting that each cluster be comprised of at least 2% of all cells. As the range and distribution of the principal component scores can vary from one analysis to the next, we use the following procedure to specify ε: We calculate the range of scores (maximum values minus minimum value) for each PC, calculate the L2 norm of the resulting vector, and multiply this value by $\varepsilon^* = 15\%$. In this way we have defined an ε that corresponds to the distance associated with moving a certain fraction (15%) of the score range along each principal component. Typically, we try different combinations of n and ε^*, in order to obtain a clustering that agrees with our desired grouping of cell types (as inferred from the expression profiles of marker genes, see below). In other words, we use DBSCAN as a tool to efficiently assign cells to clusters in a way that we see fit within the context of the construction of our classifier.

To ensure that the obtained clusters represent biologically meaningful subpopulations of cells, as well as to elucidate the likely cell type identities of the cells belonging to each cluster, we then use the cell cluster assignments to systematically search for known cell type marker genes with cluster-specific expression patterns. To perform this search, we first quantify, for all genes and clusters, the extent to which a gene is “overexpressed” in a given cluster. To do so, we employ the following method: We first apply median-normalization to the unsmoothed data, and then calculate the mean expression value of each gene in each cluster. A naïve way to quantify overexpression would then be to calculate an “overexpression ratio” (analogous to a fold change value) between the mean expression of a gene in one cluster, and the mean of the mean expression values of that gene in all the other clusters. However,

for lowly expressed genes, the ratio between two mean expression values can be very large, even though the absolute difference is very small. Additionally, if a gene is only expressed in one cluster, the ratio for a gene in that cluster is undefined, as it would require dividing by zero. To obtain more robust estimates of the overexpression ratio, we therefore determine an expression threshold *t*, and set all mean expression values that are *below* this threshold to *t*. The larger we choose t, the less sensitive our estimated ratios will be to noise. At the same time, our estimates will become more conservative, meaning that it becomes increasingly likely that we underestimate the true ratio. To determine a sensible value for *t*, we again use rely on the observation that the technical noise is Poisson-distributed in the unsmoothed and median-normalized data. Since the expression measurements for a gene in different cells are statistically independent, the sum of gene expression values across cells is Poisson-distributed as well. This allows us to calculate the technical noise level associated with different mean values. For example, if a cluster contains n = 100 cells, and the mean expression of a gene in this cluster is $\bar{e}$ = 0.04, this implies that the sum of expression measurements across cells is 0.04 * 100 = 4. As discussed previously, a Poisson variable with mean μ = 4 has a coefficient of variation (*CV*) of 50%. The technical noise associated with the mean expression value of our example gene, expressed in terms of the standard deviation relative to the observed value, is therefore 50%. More generally, the technical noise level associated with the mean value of any gene can be calculated as $CV = \sqrt{\bar{e} * n}/(\bar{e} * n)$. To ensure that our expression ratios are somewhat robust to noise, we require that the technical noise level of the mean values used in their calculation are at most 20% (CV), which is the case whenever $\bar{e} * n \geq 25$. As each cluster can contain a different number of cells, we use the size of the smallest cluster n_{min} to determine t. In conclusion, we use $t = 25 / n_{\text{min}}$ in the calculation of overexpression ratios. For each cluster, we rank the genes by their overexpression ratios for that cluster, and keep the first 10 or 20 genes. We then search this list for genes that are known to be specifically expressed in a certain cell type, or that are known to carry out an important biological function that is specific for a certain cell type. For example, CD3 genes (*CD3D*, *CD3E*, *CD3G*) encode individual chains of the T cell co-receptor, and are therefore useful markers for T cells. To visually confirm the cluster-specific expression of such a marker gene, the smoothed or unsmoothed expression profile of that gene can be visualized in a PCA scatterplot, by color-coding cells using the gene expression values.

Classification method

Using terminology from graph theory, a Moana cell type classifier consists of a rooted tree of independent Moana subclassifiers (a "classification tree"). We refer to the classifier at the root of the tree as the "top-level classifier". To predict the cell type identities of cells in a given dataset, the subclassifiers are applied successively: First, the top-level classifier is applied, and then, for each cell type (class) predicted by this classifier, a different subclassifier is applied to distinguish between cell type-specific subtypes (subclasses), and so on, until all subclassifiers at the leaves of the classification tree have been applied. As a result, each cell in the dataset will be assigned exactly one cell type from certain a set of cell types. This set of cell types is the union of all cell types associated with the subclassifiers at the leaves of the classification tree. For convenience we will henceforth simply refer to Moana subclassifiers as "classifiers", and assume that it is understood that a Moana cell type classifier for a given tissue generally consists of a hierarchically organized ensemble of such independent (sub-)classifiers.

Cell type classification in the Moana framework fundamentally relies on support-vector machine (SVM) classifiers with a linear kernel. Training and prediction is performed on smoothed and PCA-transformed

data, although the smoothing step may be skipped when the signal-to-noise ratio is sufficiently high to accurately distinguish between the classes in the raw, unsmoothed data. Due to our hierarchical approach, a "class" can represent a diverse subpopulation of cells (such as lymphocytes, or "non-ductal cells"), an individual cell type with multiple subtypes (such as T cells) or a specific subtype (such as naïve CD4+ T cells), but for convenience we will henceforth simply use "cell type" to refer to all of these cases. Each Moana classifier in consists of three individual models: 1) a smoothing model, 2) a PCA model, 3) an SVM classification model. The three models are applied successively to obtain the cell type predictions. The smoothing model reduces the technical noise levels in the data, the PCA model generates a low-dimensional representation of the data based on the smoothed data, and the SVM model predicts the cell type of each cell based on this low-dimensional representation.

The desired properties of properly trained and validated Moana classifiers are 1) accuracy, even in discriminating between highly similar cells types, 2) robustness, allowing the accurate identification of cell types in the presence of batch effects or biological perturbations, and 3) computational efficiency, allowing the assignment of cell type identities for datasets containing thousands or tens of thousands of cells in seconds or a few minutes, using a modern computer.

To describe the details of our classification method, we will first describe the motivation behind its design, then describe the three model components (smoothing model, PCA model, SVM model) that together perform classification, and finally describe an effective strategy for training Moana classifiers. Our validation method will be described in the next subsection.

Motivation

To motivate the design of our classification method, it is useful to point out that in the absence of noise (both biological and technical), principal components (PCs) would be highly effective instruments for separating between individual cell types. The leading PCs would capture modules of mutually correlated and anticorrelated genes (for example, a set of genes with high expression in one cell type, and low expression in all the others), and the projection of the data onto these PCs would allow a clear separation of individual subpopulations, eventually allowing all cell types to be linearly separable in this low-dimensional space. We rely on this observation in our clustering method (see above), in which we try to remove as much as noise as possible using smoothing, with the aim of allowing a few subpopulations to become separable by the first two PCs. Obviously, real scRNA-Seq expression measurements are affected by both biological and technical noise. We can adopt a definition of cell types that requires expression differences between cell types to always be larger than cell-to-cell heterogeneity within a cell type, which allows us to ignore biological noise. However, technical noise levels in scRNA-Seq data are typically very high (see above), and can significantly impact the ability of PCs to clearly separate between cell populations. First of all, even if we had access to the PC space obtained from the noiseness data, the cells belonging to different cell types might no longer be linearly separable in that space when cell type differences are small and technical noise levels are high. More importantly however, the presence of technical noise that affects each measurement independently leads to reduced gene-gene correlations (covariances), which form the basis for the definition of PCs. Consequently, when we apply PCA to scRNA-Seq data, we cannot expect the PCs do capture expression modules as effectively as in noiseless data, and therefore each PC is less effective in separating between cell types.

These thoughts motivated us to combine our kNN-smoothing algorithm, which performs computational denoising of scRNA-Seq data, with PCA, in order to obtain a low-dimensional representation of the data that would allow cell types to become linearly separable. Since each principal component represents an aggregation of expression values from many genes, we thought that this approach would also make prediction robust to batch effects and biological pertubations. When batch effects are present, the expression measurements for many genes can exhibit relatively small systematic biases, but these effects would likely average out when the expression values of many genes are aggregated. In contrast, biological pertubations could strongly affect a small set of genes, but if we assume that each principal component represents a much larger set of genes, these differences would not affect the PC scores very much.

To construct a classification method that combines smoothing, PCA, and SVM models, it is necessary to precisely define how each of these models are first trained and then applied during prediction. This is relatively straightforward for the PCA and SVM models: During training, we perform PCA, keep a certain number of leading PCs, and then train an SVM classifier on this transformed data. During prediction, we project the data onto the PCs defined during training, and then apply the SVM classifier on this transformed data. Attention must be paid to the fact that we perform PCA after applying a non-linear transformation (the Freeman-Tukey transform). We therefore have to ensure that for prediction, the data is scaled to the same median transcript count that was used during training (see "PCA model" below for details). However, it is less obvious how a smoothing model can be learned during training and then applied during prediction. During each smoothing step, the kNN-smoothing algorithm internally performs PCA to identify the nearest neighbors of each cell. We therefore realized that the smoothing model needed to contain its own PCA models (learned from the training data), so that smoothing can be effectively applied to a new dataset, independent of the cell type composition of this new dataset. In addition, we needed to define a criterion for how many smoothing steps to perform, as different scRNA-Seq datasets of the same tissue can exhibit different median transcript counts, and therefore require different number of smoothing steps in order to attain an appropriate transcript count for the PCA model.

PCA model

The PCA model takes as input a smoothed or raw (unsmoothed) gene expression matrix $\boldsymbol{X}$, containing approximately 20,000 genes, and returns a matrix $\boldsymbol{Y}$ with far fewer dimensions, typically around 20. To obtain this result, the cells in the input matrix are first scaled to a certain fixed transcript count t_{pca}. Next the Freeman-Tukey (FT) transformation ($y = \sqrt{x} + \sqrt{x+1}$) is applied, and the FT-transformed expression profiles are projected onto a low-dimensional subspace with a d-dimensional orthonormal basis $\boldsymbol{W}_{\text{pca}}$. In summary, the PCA model has two parameters, t_{pca} and $\boldsymbol{W}_{\text{pca}}$.

To train a PCA model on an expression matrix $\boldsymbol{X}$, we set t_{pca} to be the median transcript count across all cells in the training data, normalize each expression profile so that its transcript counts sum to this number, apply the Freeman-Tukey transform, and finally perform PCA on the normalized and transformed data, computing only the n_{pc} leading PCs. The parameter n_{pc} has to be provided during training. The matrix $\boldsymbol{W}_{\text{pca}}$ then consists of the loadings of the n_{pc} leading PCs.

Smoothing model

The smoothing model takes as input a raw gene expression matrix $\boldsymbol{X}$, and returns a smoothed expression matrix $\boldsymbol{S}$ with the same dimensions as $\boldsymbol{X}$. To obtain $\boldsymbol{S}$, a modified version of the *kNN-smoothing*

algorithm[7] (version 2) is applied to $\boldsymbol{X}$. In this modified version, instead of performing PCA on the partially smoothed expression matrix in the *i*'th smoothing step, a PCA model $\mathcal{M}_i$ as described above is used to obtain the low-dimensional matrix $\boldsymbol{Y}_i$, which is then used to determine the nearest neighbors of each cell, allowing the calculation of the partially smoothed matrix $\boldsymbol{S}_i$. The PCA model $\mathcal{M}_i$ is selected from a set of PCA models $\{\mathcal{M}_j\}$, each of which with a specific parameters t_j (see above). The specific model $\mathcal{M}_i$ that is selected for the *i*'th smoothing step is the model with the largest t_j such that $t_j * t_{\text{frac}} \leq c_i$, where c_i is the median transcript count of $\boldsymbol{S}_{i-1}$ (for i=1, this is the median transcript count of $\boldsymbol{X}$). Smoothing stops once we obtain a smoothed matrix with a median transcript count of at least $t_{\text{target}} * t_{\text{frac}}$. In summary, the parameters of the smoothing model consist of t_{target}, t_{frac}, and a set of PCA models $\{\mathcal{M}_j\}$.

To train a smoothing model on an expression matrix $\boldsymbol{X}$, we apply the *kNN-smoothing* algorithm (version 2), as described by Wagner et al.[7], to $\boldsymbol{X}$, resulting in the smoothed expression matrix $\boldsymbol{S}_0$. (This requires the specification of the *kNN-smoothing* parameters *k* and *d*.) We then set t_{target} to be the median transcript count of $\boldsymbol{S}_0$. We generally set t_{frac} to a default value of 0.9. To train the PCA models $\{\mathcal{M}_j\}$, we use the following approach: To obtain $\mathcal{M}_1$, we divide all expression values in $\boldsymbol{S}_0$ by two, resulting in a scaled matrix $\boldsymbol{S}_1$, and then a PCA model on $\boldsymbol{S}_1$, as described above. To obtain $\mathcal{M}_2$, $\mathcal{M}_3$, …, we repeat this procedure ($\boldsymbol{S}_2 = 0.5 * \boldsymbol{S}_1$, etc.), until the median transcript count of the scaled matrix falls below a certain threshold t_{low}. At that point, we train one last PCA model and then stop. t_{low} must be provided during training, but can be set to a default value of 500 transcripts. As a result of this procedure, we will have generated a series of PCA models $\{\mathcal{M}_j\}$ where $t_j = 0.5 * t_{j-1}$.

It may not be clear why a smoothing model relies on multiple PCA models, trained on differentially scaled versions of the same matrix ($\mathbf{S}_1$). Certainly, it would be much simpler to use a single PCA model in our modified version of the *kNN-smoothing* algorithm. We recall that each PCA model operates on an expression matrix scaled to specific transcript count *t* (see above). This means that during each step of the smoothing algorithm, we would scale the unsmoothed or partially smoothed expression matrix to a fixed transcript count *t*, before the FT transform is applied and the data is projected onto the PCs. The question that then arises is whether this PCA model should be trained on data with a high or low median transcript count (corresponding to a high or a low value of *t*)? In other words, should we train this PCA model on smoothed or unsmoothed data? If we train it on smoothed data (resulting in a larger *t*), then lowly expressed genes will exhibit much lower noise levels, thus allowing the PCA to better capture the correlation structure of those genes. However, when we apply this model to perform smoothing during the prediction phase, we start with an unsmoothed expression matrix, in which lowly expressed genes exhibit a very high level of technical noise. By scaling these values up, we amplify this noise, resulting in an unnecessarily noisy low-dimensional representation, and potentially reducing the accuracy with which neighbors are identified for smoothing. Conversely, if we train the PCA model on unsmoothed data, we waste useful information contained in lowly expressed genes during the later steps of the smoothing algorithm. Therefore, by providing PCA models trained on differentially scaled versions of the same matrix, we can maximize the amount of information we use during each smoothing step.

SVM classification model

For our SVM classification model, we use ν-support vector classification[8] (*ν*-SVC) with a linear kernel. SVM classifiers attempt to define a hyperplane that maximizes the margin between the two classes of points. The space in which this is attempted can either be the original space in which the data lives, or a potentially infinite-dimensional feature space that is obtained using an approach known as the kernel

trick. By using a linear kernel, we have selected the former option. For cases where the classes are not linearly separable, slack variables allow the classifier to tolerate a certain fraction of outliers, i.e., data points that end up lying on the wrong side of the respective class margin. ν-SVC is a reparameterization of the original C-SVC formulation, which enables a more intuitive tuning of the tolerance to outliers. Under certain conditions, ν represents an upper bound to the fraction of margin errors in the training data[8]. Therefore, by training an SVM with a low setting of ν, one requests that the hyperplane be defined in a way that results in few margin errors in the training data (which can increase classification accuracy, but can also result in overfitting). Finally, a multi-class classifier can be constructed by training an SVM classifier for each class pair, in which case the predicted class is selected using a majority vote rule.

There are several reasons for why we have decided to restrict ourselves to using a linear kernel: First, as the foregoing discussion has shown, we can in principle expect cell types to be linearly separable in PC space, and we therefore expect very limited benefits (in terms of classification accuracy) from using a non-linear kernel. Second, the use of a linear kernel reduces the number of parameters that needs to be specified during training (from two to one), thus simplifying the training procedure and significantly reducing the risk of overfitting. Third, when a linear kernel is used, the learned feature (PC) coefficients can be interpreted as weights, thereby providing insights into which PCs (and ultimately genes; see below) are the most informative for classification.

The accuracies of SVM classifiers can be negatively impacted by a class imbalance in the training data, which can result in the decision boundary being pushed towards the minority class[9]. We adopt a simple oversampling approach to counteract this effect, in which we oversample cells from the minority class (in the smoothed and PCA-transformed data) until the minority class reaches the same cell count as the majority class. We do so by sampling without replacement until we have selected all cells in the minority class once, and then repeat the sampling cycle as necessary. We did not explore more sophisticated approaches for addressing class imbalance or systematically compare classification accuracies with and without oversampling of the minority class.

Summary

In summary, each trained Moana classifier consists of the following parameters:

- For the smoothing model: t_{target}, t_{frac}, $\{\mathcal{M}_j\}$ (PCA models, each with parameters t_j and $\boldsymbol{W}_j$)
- For the PCA model: t_{pca}, $\boldsymbol{W}_{pca}$
- For the SVM model: decision function parameters (support vectors, their coefficients, intercept)

(Note that in practice, the parameters t_{pca} and t_{target} are generally identical.)

Training a Moana classifier on an expression matrix $\boldsymbol{X}$, annotated with cell type labels $\boldsymbol{y}$, generally requires the manual specification of the following parameters:

- For the smoothing model: k, d, t_{frac} (default value: 0.9), t_{low} (default value: 500)
- For the PCA model, all parameters can be determined automatically (see below)
- For the SVM model: ν (default value: 0.02)

The parameters required for training the PCA model can be determined automatically as follows: To set t_{pca}, we use the median transcript count of the smoothed matrix. To set n_{pca}, we can simply use the value provided for d. However, setting d is also straightforward, as we can simply use the value of d that we

found during our clustering analysis, in which we generated the cell type labels ***y*** (see above). As a result, the only parameter that needs to be tuned (adjusted) for the training of a Moana classifier is *k*, which determines the degree of smoothing. When cell types are sufficiently different relative to the noise levels in the data, we can simply use *k*=1 (no smoothing).

General strategy for training a Moana classifier

In the subsection describing our clustering method, we have described our general strategy for clustering scRNA-Seq datasets and annotating clusters with cell types. When this step has successfully been completed, we have an expression matrix with annotated cell types, which can then be used for training a Moana classifier. When more than two clusters were identified, it is first necessary to decide on whether to train a binary or a multi-class classifier. On the one hand, accurately distinguishing between three or more cell types is generally a more difficult problem than distinguishing between only two cell types, and may require more smoothing, resulting in a slower prediction speed. Moreover, the identification of the most informative genes based on the SVM feature coefficients is only feasible for binary classifiers. On the other hand, training a classifier for three or more classes reduces the total number of classifiers that needs to be trained, which can be convenient. It may also improve prediction speed, as a single smoothing step can form the basis for distinguishing between multiple cell types. Generally speaking, we recommend building a multi-class classifier, using as cell types all subpopulations that clearly separate in PC space during the clustering stage.

Once the cell types have been determined, we train Moana classifiers with increasing settings of *k* (*k*=1, 2, 4, 8, 16, etc.), using the approach outlined previously (see "Summary"). By default, we use *v*=0.02 for training the SVM classifier, which requests that at most 2% of cells violate the classification margin[8]. (When this is not possible, for example when trying to classify highly similar cell types based on insufficiently smoothed data, this can result in very low-accuracy classifiers.) When training is finished, we quantify the prediction accuracy of the classifier on the training dataset. To obtain a meaningful measure of the prediction accuracy even when classes are highly imbalanced, we calculate the precision for each class, defined as TP / (TP + FP), where TP and FP represent the number of true and false positives, respectively. We then calculate the average precision across all classes. Our general strategy for choosing *k* is to use the lowest *k* such that the average precision on the training data is at least 99%. In cases where classes are extremely imbalanced, it may be necessary to further reduce *v* to obtain such high average precision, for example to *v*=0.01. Using this strategy, we were able to train classifiers with good performance without having to use a validation dataset for parameter tuning, suggesting that the degree of overfitting was generally low.

The choice of *k* not only affects the accuracy of the classifier, but also several other key properties. First, the time it takes to perform smoothing increases linearly with k[7], so doubling *k* will also double the time it takes to apply the classifier's smoothing model. Second, during the training phase it is important to ensure that the data is not oversmoothed, as this would result in the classifier being trained on artificial expression profiles. Therefore, depending on the number of cells available, choosing *k* too large could compromise prediction accuracy. Third, during the prediction phase, the smoothing model attempts to match the degree of smoothing performed during training. Again, in order for this to not result in oversmoothing, the data needs to contain a sufficient number of cells from each type. Therefore, choosing *k* too large could mean that the classifier cannot be accurately applied to small datasets. Finally, increasing *k* could help improve the robustness of the classifier, as even lowly expressed genes can contribute useful information when sufficient smoothing is applied. In summary, *k* represents an

important tuning parameter in the training a Moana classifier, and different settings of *k* could result in classifiers with drastically different performance.

As cell types are frequently represented in datasets at widely different proportions, the issue of highly imbalanced arises frequently. As cell type imbalance not only affects SVM performance (see above), but also PCA results (rare cell types tend to be represented by higher PCs), we generally recommend performing a second clustering/classification iteration with the goal of producing a training dataset in which the selected classes are more balanced. (This is only applicable in cases when the training dataset used in the first iteration represented only a subset of the available data.) The idea is to use the classifier trained in the first iteration to find additional cells belonging to the minority class in the entire dataset, and then to create a synthetic dataset with at least the same number of cells as the first training dataset, but where ideally the two classes are represented at equal proportions. This results in more robust classifiers in which the first few PCs represent the differences between the classes.

Mirror validation method

Mirror validation requires a trained Moana classifier, the training dataset, and an independent "mirror dataset". The goal of mirror validation is to test, for each cell type included in the classifier, whether the cells predicted to belong to this cell type form a coherent group that can be accurately identified in an independent dataset. Since this does not necessarily imply that the cell type label is correct (relabeling a cell type does not affect mirror validation results), we avoid the use of the term "accuracy" in describing classifiers with good mirror validation results. Instead, we refer to those classifiers as having a high degree of "coherence". The mirror validation approach consists of using the trained classifier to predict cell type identities in the mirror dataset, using these predictions to train a new "mirror classifier" on the mirror dataset, and then using this new classifier to test if its prediction on the original training dataset match those of the original classifier.

To apply this approach in the context of our framework, a few issues must be addressed: First, since a Moana classifier typically consists of a hierarchy of subclassifiers, training a new classifier is a multi-step procedure, and each step typically involves some manual analyses. This would make the implementation of mirror validation quite time-consuming. To overcome this issue, we automated the entire training process for cases in which we simple try to "retrain" a classifier on a new dataset. To automatically re-train a Moana classifier on a new dataset, we apply a recursive algorithm that mirrors the manual training of Moana classifiers: First, we retrain the top-level classifier, and then retrain its subclassifiers on the subsets of cells predicted to belong to each cell type. If the mirror dataset contains more than 2,000 cells, we maintain computational efficiency by training the top-level classifier on a subset of 2,000 cells, sampled in a way that balances the cell type proportions as best as possible. For example, when the top-level classifier distinguishes between two cell types, we aim to sample the cells from the mirror dataset in a way that produces a training dataset consisting of 1,000 cells predicted to be of cell type "A", and 1,000 cells predicted to be of type "B". We apply the same sampling/balancing approach during the training of each subclassifier, whenever there are more than 2,000 cells available.

We found that retraining a subclassifier on a new dataset was non-trivial even in cases where the validation dataset contains exactly the same cell type composition as the original training data. This was because the effectiveness of the *kNN-smoothing* algorithm often depended strongly on the choice of the *d* parameter, which required dataset-specific tuning and was therefore difficult to automate. To address this problem, we decided to apply the smoothing model from the original classifier to the mirror

dataset, and to then train the PCA model of the mirror classifier on this smoothed expression matrix. To train the smoothing model of the mirror classifier, we artificially downscale this smoothed expression matrix to train the individual PCA models that constitute the smoothing model. To train the SVM model of the mirror classifier, we train an SVM model on the smoothed and PCA-transformed mirror dataset, using the same value for *v* as in the original classifier. In this way, we were able to train the smoothing, PCA, and SVM models of each subclassifier on the mirror dataset, giving rise to a new Moana classifier that mirrored the original classifier but was fully trained on the mirror dataset. Finally, we applied this "mirror classifier" to the original data, and compared the prediction results to the results obtained with the original classifier. We treat the predictions from the original classifier as the ground truth, allowing us to calculate precision and recall for each cell type.

We note that high mirror validation accuracies can only be interpreted to reflect high classification coherence across datasets as long as the *v* values used in the training of the SVM models of the mirror classifier are very small (e.g., $v \leq 0.02$). Otherwise, the ability of SVM models to fit the training data in the presence of a large number of outliers can result in a significant overestimation of coherence. To avoid these kinds of biases, we perform mirror validation for each classification model individually, and test a range of values for *v*. In cases where classification coherence is poor, we generally observe a significant increase in the observed mirror validation accuracies with larger settings of *v* (reflecting the aforementioned overestimation effect), whereas the opposite is true for classifiers with high coherence.

Implementation

We implemented all components of the *Moana* framework in Python 3 using an objected-oriented programming approach (all further references to "Moana" in this paragraph refer to our Python implementation, not the conceptual framework with the same name). We relied on scikit-learn[10] implementations for all standard machine learning methods used in Moana (PCA, SVM classification, DBSCAN), For visualization, we relied on the plotly[11] (version 2) library. Moana relies on a new, plain-text file format for storing sparse scRNA-Seq expression matrices, which is an extension of the MatrixMarket format in which cell IDs and gene names are stored in comment fields. Moana provides functions to import expression data in other formats to be imported, including tab-separated or comma-separated values and a compressed (gzip'ed) format produced by the 10X genomics CellRanger software. Moana also uses new data structures to represent expression matrices and cell type annotations / predictions, which are derived from the pandas[12] *DataFrame* and *Series* classes, respectively. A detailed technical documentation of our implementation is forthcoming (https://github.com/yanailab/moana).

List of datasets analyzed

We analyzed the following datasets published online by 10X Genomics:

- PBMC-8k: 8k PBMCs from a Healthy Donor
  (10X v2 chemistry; processed with Cell Ranger 2.1.0):
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/2.1.0/pbmc8k
- PBMC-4k: 4k PBMCs from a Healthy Donor
  (10X v2 chemistry; processed with Cell Ranger 2.1.0):
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/2.1.0/pbmc4k
- PBMC1-6k: 6k PBMCs from a Healthy Donor
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/pbmc6k

- PBMC1-16k: 33k PBMCs from a Healthy Donor (random subset of 16,000 cells)
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/pbmc33k

We analyzed the following 10X datasets first reported by Zheng et al.[13]:

- B-cell: CD19+ B Cells
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/b_cells
- CD14-Mono: CD14+ Monocytes
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/cd14_monocytes
- NK-cell: CD56+ Natural Killer Cells
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/cd56_nk
- CD4-Tcell: CD4+ Helper T Cells
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/cd4_t_helper
- CD8-Tcell: CD8+ Cytotoxic T cells
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/cytotoxic_t
- CD4-Tnaive: CD4+/CD45RA+/CD25- Naive T cells
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/naive_t
- CD8-Tnaive: CD8+/CD45RA+ Naive Cytotoxic T Cells
  (10X v1 chemistry; processed with Cell Ranger 1.1.0)
  https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/naive_cytotoxic

We analyzed the following datasets first reported by Kang et al.[14]:

- Kang18-Ctl: batch 2 control
  (10X v1 chemistry; processed with Cell Ranger 1.2.0)
  https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2560248
- Kang18-Tx: batch 2 stim (IFN-beta)
  (10X v1 chemistry; processed with Cell Ranger 1.2.0)
  https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2560249

We analyzed the following pancreas datasets first reported by Baron et al.[15]:

- Baron16-1: mouse pancreatic islets, sample 1
  (inDrop; processed with the SingleCell[16] pipeline)
  https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2230761
- Baron16-2: mouse pancreatic islets, sample 2
  (inDrop; processed with the SingleCell[16] pipeline)
  https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2230758

- Baron16-3: mouse pancreatic islets, sample 3
  (inDrop; processed with the SingleCell[16] pipeline)
  https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2230759
- Baron16-4: mouse pancreatic islets, sample 4
  (inDrop; processed with the SingleCell[16] pipeline)
  https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2230760

Preprocessing of datasets downloaded from the 10x Genomics website

First, we extracted a list of 19,950 protein-coding genes from the human Ensembl genome annotations (release 92; http://ftp.ensembl.org/pub/release-92/gtf/homo_sapiens/Homo_sapiens.GRCh38.92.gtf.gz) using the ensemble.get_protein_coding_genes() function from the Moana package. For each gene, we store both its Ensembl ID and its gene symbol. We next curated a list of 80 human genes encoding cytosolic ribosomal proteins based on their gene names (all names starting with "RPL" or "RPS", as well as the genes *FAU* and *UBA52*) and a comparison with the Ribosomal Protein Gene Database (RPGDB; http://ribosome.med.miyazaki-u.ac.jp/). Similarly, we curated a list of 77 human genes encoding mitochondrial ribosomal proteins based on their gene names (all names starting with "MRPL" or "MRPL", as well as the gene *DAP3*), and comparison with the RPGDB. We further curated a list of all 13 human protein-coding genes located on the mitochondrial genome.

We then applied the following processing procedure to all 10x Genomics PBMC datasets: For each dataset, we downloaded its "Gene / cell matrix (filtered)" from the 10x Genomics website (https://support.10xgenomics.com/single-cell-gene-expression/datasets). This matrix contains the UMI-filtered expression values (transcript counts) for all genes and cells. We then filtered the genes against our list of protein-coding genes, based on Ensembl ID. We next removed all genes without any expression in any of the cells. We then converted gene names from Ensembl ID to gene symbols. In rare cases where multiple Ensembl IDs mapped to the same gene symbol, we summed the expression values of all Ensembl IDs.

Next, we used our curated gene lists to remove genes encoding cytosolic ribosomal proteins and genes located on the mitochondrial genome. We found that depending on the cell type, the 80 genes encoding cytosolic ribosomal proteins comprised up to 40% of all transcripts per cell, and we decided to exclude these genes so that our analyses would not be unduly impacted by fluctuations of these extremely highly expressed genes. In addition, as different scRNA-Seq protocols could exhibit different capture efficiencies for ribosomal transcripts, this would avoid batch effects resulting from these differences. Similarly, we found that the 13 mitochondrially encoded genes had very high expression levels that comprised between 2-4% of total transcripts per cell, and excluded those genes for the similar reasons. We did not remove any of the 77 genes encoding mitochondrial ribosomal proteins, which only comprised a median of 0.5-0.7% of all transcripts per cell after gene filtering.

For the datasets generated using 10x genomics' "version 1" chemistry, we also filtered cells for which the proportion of transcripts from mitochondrially encoded genes was above 10%, or which had less than 500 transcripts after removing these genes as well as the ribosomal genes.

Preprocessing of pancreas data by Baron et al.

To preprocess the pancreas data first reported by Baron et al.[15], we applied the same procedure as for the "version 1" data from the 10X genomics website, except that we required a minimum transcript count of 1,000 and a maximum fraction of transcripts from mitochondrially encoded genes of 15%.

Preprocessing of PBMC data by Kang et al.

To preprocess the PBMC data first reported by Kang et al.[17], we applied the same procedure as described for the "version 1" data from the 10X genomics website.

t-SNE analyses

All t-SNE analyses were conducted by first applying median-normalization, FT-transformation, and PCA, keeping the first 20 principal components, and then applying the t-SNE implementation from scikit-learn (version 0.19.1) with a perplexity value of 30.

Construction of a Moana classifier for PBMCs

We constructed the Moana classifier for PBMCs exclusively on data from the PBMC-8k dataset (*n*=8,381). The hierarchy of classification models is shown in **Figure 2a**. To train each classification model (e.g., for T cells), we first used the "upstream" classification models to obtain the set of all cells predicted to be of the current cell type (e.g., T cells). If the number of those cells exceeded 3,000, we generated a random sample of 2,000 cells on which we performed smoothing, using *k*=128 and the *d* value indicate in Figure 2a (in most cases, *d*=20). We then used DBSCAN to assign cells to clusters, and examined the expression patterns of genes that were highly overexpressed in individual clusters and known to be expressed in a cell type-specific fashion (**Supplementary Figures 2 and 3**). Key marker genes that we used for this purpose included *CD79A*/ *CD79B* for B cells, *CD3D*/*CD3E*/*CD3G* for T cells, *GNLY* and *NKG7* for NK cells, CD14 and *FCGR3A* (*CD16*) for monocytes, *FCER1A* for myeloid dendritic cells, *JCHAIN* for plasmacytoid dendritic cells, *CCR7* and *SELL* for naïve T cells, and *CD8A*/*CD8B* for CD8+ T cells. Markers that we found to be very useful despite not currently enjoying widespread use for identifying the respective cell types in scRNA-Seq data included *FTL* for monocytes, *CCL5* for memory T cells, and *GPR183* for naïve CD4+ T cells. After clustering and cell type annotation, in cases where we had sampled a subset of cells with a certain type from the data, we trained a preliminary Moana classification model with high accuracy on the training dataset, and applied this model to predict the cell type identities for all cells of that type. We then tried to create a balanced dataset of 2,000 cells to train the final Moana classification model. For example, we performed clustering and cell type annotation on 2,000 lymphocytes, randomly sampled from a total of 6,008 cells predicted to be lymphocytes. This resulted in the annotation of 1,620 cells as T or NK cells, and 380 cells as B cells. We then used the cell type annotations to train a Moana classification model on these 2,000 cells, and applied the classification to all 6,008 lymphocytes, resulting in the prediction of 4,775 T and NK cells and 1,233 B cells. We then used this information to construct a new training dataset comprising 1,000 predicted T or NK cells, and 1,000 B cells, and trained the final classification model for distinguishing between T/NK cells and B cells on this dataset. Classification accuracies are generally improved when the training dataset contains balanced class proportions. In cases where there were less than 1,000 cells available for one cell type in the entire dataset, we balanced the classes as much as possible (we did not use training datasets with less than 2,000 cells as long as there were sufficient cells from the majority cell type available). All final classification models were trained using *v*=0.02. All individual classification models were then assembled into a single hierarchical classifier, so that cells are first classified using the top-

level classification model, and then the classification models of the next level of the hierarchy are invoked for the cells predicted to have their respective types (e.g., for cells predicted to be Monocytes or DCs by the top-level classification model, the Monocyte/DC classification model is invoked, which distinguishes between monocyte and DC identities). We provide our classifier as a binary file that can be loaded using our Moana Python package (**Supplementary File 1**).

Clustering of all cells in the PBMC-4k dataset

We performed clustering on all cells in the PBMC-4k dataset by pursuing a hierarchical approach analogous to the clustering strategy for the construction of the PBMC classifier (**Supplementary Figure 5**). However, we did not use random subsets of cell subsets, and instead performed clustering on all cells in each step. We manually identified a set of 37 pDC cells by their expression of the *JCHAIN* gene (**Supplementary Figure 4c**).

Quantification of classification accuracy using clustering results as the ground truth

We quantified accuracy by calculating precision and recall for each cell type, where precision is defined as (TP + FP) / TP, and recall is defined as TP / (TP + FN). (TP = true positives, FP = false positives, FN = false negatives)

Calculation of coherence scores based on mirror validation results

We defined the coherence of each cell type as its F_1 score, which is defined as the harmonic mean of precision and recall: $F_1 = ((\text{precision}^{-1} + \text{recall}^{-1}) / 2)^{-1}$.

Extracting the most informative genes from a trained Moana classification model

To determine how informative each gene is in a Moana classification model, we combine its PCA loadings with the linear SVM coefficients by calculating a dot product. To enable comparisons of these scores across classification models, we standardize the resulting value so that a hypothetical gene that perfectly loads onto each PC (value 1/-1) with the sign matching that of the corresponding SVM coefficient, would receive a score of 1 (or -1). To do so, we divide the dot product values of all genes by the sum of the absolute SVM coefficient values.

We rely on the scikit-learn implementation of v-SVC (sklearn.svm.NuSVC), which in turn relies on libsvm. Libsvm implements multi-class SVM classification using a one-vs-one scheme, so that each set of SVM coefficients corresponds to a pair of cell types. Our implementation therefore does not allow a straightforward determination of the most informative genes for classification models that distinguish between more than two cell types. (However, it would be possible to calculate gene scores for each pair and to then calculate the average score.)

Comparison of cell type-specific expression profiles

To compare cell type-specific expression profiles across datasets, we calculated the mean UMI count for each gene across all cells of a given type (without normalization), and set mean values below 0.05 to 0.05. For each cell type, we then filtered for genes with an expression of greater than 0.,05 in either dataset, applied the transformation $y = log_{10}(x)$, and calculated the Pearson correlation coefficient. We applied the same approach for calculating correlations between expression profiles of different cell types in the same dataset.

Implementation of *scmap-cluster*

We implemented *scmap-cluster* in Python following the description of the method provided by Kiselev et al. [18]. To ensure that we correctly implemented each step, we reproduced the feature selection and cell type prediction results presented by Kiselev et al. for cross-predicting cell types between the "Xin" and "Baron (human)" datasets (Supplementary Figure 8), which the authors kindly made accessible on their website (https://hemberg-lab.github.io/scRNA.seq.datasets/). Briefly, the *scmap-cluster* cell type classification method comprises a training step and a prediction step. The training step involves the selection of features (genes) based on their expression level and dropout rate, and the calculation of average expression profiles in the training data. The prediction step involves the calculation of different measures of correlation with the average profiles from the training data and the application of a voting rule. We included our *scmap-cluster* implementation in our Moana Python package (https://github.com/yanailab/moana).

For training, we implemented the gene selection and expression profile calculation steps as follows: To select genes, we first normalized the expression values of each cell to the mean transcript count across all cells. We then excluded genes that had no expression or that had a dropout rate of zero (where "dropout" refers to the observation of zero transcripts in a given cell). Next, we applied the transformation $y=\log_2(x+1)$, and performed linear regression, using the mean expression level as the independent variable, and the $\log_2$-dropout rate as the dependent variable. The selected features are the *d* genes with the largest residuals. We confirmed that we obtained results that were identical to those presented by Kiselev et al.[18] (**Supplementary Figure 8a**). After gene selection, we calculated median gene expression levels for each cell type, using the mean-normalized and $log_2(x+1)$-transformed data.

To predict cell type identities in a new dataset, we first applied mean-normalization and *log2(x+1)*-transformation to the dataset. For each cell (expression profile), we then applied the following procedure: We calculated the cosine similarity, Pearson correlation, and Spearman correlation with each of the cell type-specific expression profiles calculated during training (see above), and assigned cells to a cell type whenever this cell type was associated with maximum values for at least two of the similarity measures, and when, additionally, the highest similarity value observed was equal to or greater than 0.7 . In all other cases, we left the cell type "unassigned". We confirmed that we obtained prediction results that were identical to those presented by Kiselev et al.[18], by predicting cell type identities in the "Baron (human)" dataset after training on the "Xin" dataset, and vice versa (**Supplementary Figure 8b**).

Construction of a Moana classifier for human pancreas

We constructed a Moana classifier for human pancreas based on the `Baron16-3` dataset (*n*=4,591) using the same strategy as for the PBMC classifier: We trained binary classification models, successively distinguishing cells, beta cells, ductal cells, other endocrine cells, immune cells, endothelial cells, and stellate cells. We trained additional binary classification models for the other endocrine cells, successively distinguishing delta cells, alpha cells, and gamma cells. We relied on the cell type markers described by Baron et al.[15] to identify each of the cell types.

Systematic identification of marker genes for pancreas cell types

To systematically determine marker genes for each pancreas cell type, we aimed to identify genes that were much more highly expressed in one cell type than in all other cell types. To do so, we first applied

median normalization to the `Baron16-3` expression matrix, and then calculated mean gene expression levels for each cell type. We then set all mean expression values below 2.326 to 2.326. This threshold was chosen so that the mean expression values represented a total of at least 100 transcripts for the cell type with the smallest number of cells (gamma cells, *n*=43; 2.326 * 43 ≈ 100). To identify the top marker genes for each cell type, we calculated, for each gene, the ratio between its expression in this cell type and each other cell type, and retained the smallest ratio. We then ranked genes by their calculated ratios.

Construction of a Moana classifier for human cultured PBMCs

We constructed a Moana classifier for human cultured PBMCs based on the Kang18-Ctl dataset (n=12,757) using the same strategy as for our first PBMC classifier (**Supplementary Figure 12**). The main difference was that we were able to identify and classify red blood cells (which initially clustered with lymphocytes), and that we were unable to accurately distinguish between T cell subtypes using our clustering method.

Quantifying cell type-specific induction of genes in the PBMC experiment by Kang et al.

To quantify the degree to which individual genes were induced following exposure to IFN-β in the experiment by Kang et al.[17], we first defined a cell type-independent "physiological" expression level for each gene as follows: Using the cell type identities predicted by our cultured PBMC classifier, we calculate mean cell type expression levels for all genes, for both the control and the treatment dataset. We then normalized all mean expression profiles to the same total number of transcripts. This number was determined as the median total transcript count across all mean expression profiles in the control dataset. To obtain robust expression ratios, we then set all normalized mean expression values below 0.1 to 0.1. For each gene, we then determined the "physiological" expression level as the maximum normalized expression level across all cell types in the control dataset. We then calculated induction values (expression ratios) for each cell type by dividing the normalized mean expression values in the treatment dataset by the physiological expression level of each gene. Finally, we retained the 200 genes with the highest maximum expression ratios across all cell types, and visualized their induction profiles as a heatmap (**Figure 5d**), following agglomerative hierarchical clustering using the Euclidean distance metric.

References

1. Dal Molin, A. & Di Camillo, B. How to design a single-cell RNA-sequencing experiment: pitfalls, challenges and perspectives. *Brief. Bioinform.* (2018). doi:10.1093/bib/bby007
2. Ziegenhain, C. *et al.* Comparative Analysis of Single-Cell RNA Sequencing Methods. *Mol. Cell* **65**, 631-643.e4 (2017).
3. Grün, D., Kester, L. & van Oudenaarden, A. Validation of noise models for single-cell transcriptomics. *Nat. Methods* **11**, 637–640 (2014).
4. Wagner, F., Yan, Y. & Yanai, I. K-nearest neighbor smoothing for high-throughput single-cell RNA-Seq data. *bioRxiv* (2017). doi:10.1101/217737
5. Freeman, M. F. & Tukey, J. W. Transformations Related to the Angular and the Square Root. *Ann. Math. Stat.* **21**, 607–611 (1950).
6. Yip, S. H., Sham, P. C. & Wang, J. Evaluation of tools for highly variable gene discovery from single-cell RNA-seq data. *Brief. Bioinform.* (2018). doi:10.1093/bib/bby011
7. Wagner, F., Yan, Y. & Yanai, I. K-nearest neighbor smoothing for high-throughput single-cell RNA-Seq data. *bioRxiv* 217737 (2018). doi:10.1101/217737

8. Schölkopf, B., Smola, A., Williamson, R. C. & Bartlett, P. L. New Support Vector Algorithms. *Neural Comput.* **12**, 1207 – 1245 (2000).
9. Akbani, R., Kwek, S. & Japkowicz, N. Applying Support Vector Machines to Imbalanced Datasets. in *Machine Learning: ECML 2004* (eds. Boulicaut, J.-F., Esposito, F., Giannotti, F. & Pedreschi, D.) 39–50 (Springer Berlin Heidelberg, 2004).
10. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *J Mach Learn Res* **12**, 2825–2830 (2011).
11. Inc, P. T. Plot.ly. (2015). Available at: https://plot.ly.
12. McKinney, W. Data Structures for Statistical Computing in Python. in *Proceedings of the 9th Python in Science Conference* (eds. Walt, S. van der & Millman, J.) 51–56 (2010).
13. Zheng, G. X. Y. *et al.* Massively parallel digital transcriptional profiling of single cells. *Nat. Commun.* **8**, 14049 (2017).
14. Kang, H. M. *et al.* Multiplexing droplet-based single cell RNA-sequencing using natural genetic barcodes. *bioRxiv* 118778 (2017). doi:10.1101/118778
15. Baron, M. *et al.* A Single-Cell Transcriptomic Map of the Human and Mouse Pancreas Reveals Inter- and Intra-cell Population Structure. *Cell Syst.* **3**, 346-360.e4 (2016).
16. SingleCell Python package.
17. Kang, H. M. *et al.* Multiplexed droplet single-cell RNA-sequencing using natural genetic variation. *Nat. Biotechnol.* **36**, 89–94 (2018).
18. Kiselev, V. Y., Yiu, A. & Hemberg, M. scmap: projection of single-cell RNA-seq data across data sets. *Nat. Methods* **15**, 359–362 (2018).