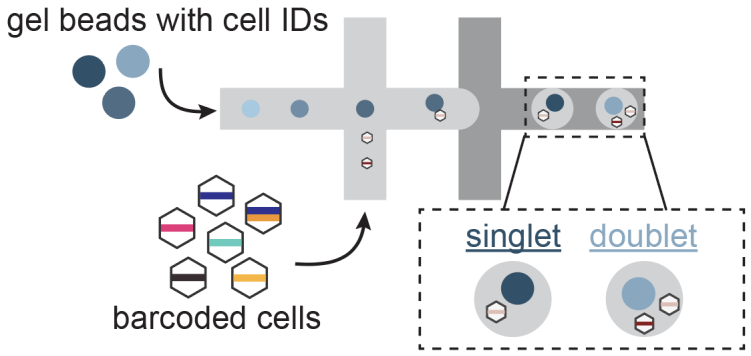
Welcome to the documentation of singletCode!

singletCode (link to preprint) uses cell barcoding technology to identify **true singlets** in scRNAseq data. In our preprint, we used singletCode to benchmark existing doublet detection methods and train a classifier to detect singlets in non-barcoded datasets as well of the same cell type.

doublets in sequencing

How do doublets form?

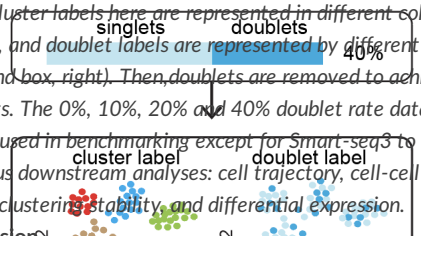
Single-cell RNA sequencing (scRNA-seq) datasets contain true single cells, or singlets, in addition to cells that randomly coalesce during the protocol, or doublets. Sometimes, there are higher rates of doublets which can be attributed to cellular physiology and experimental protocols can lead to cell clumping. Doublet percentage in a sample be as high as 40%. Doublets can be two very transcriptionally different cells captured together (heterotypic) or two transcriptionally similar cells captured together (homotypic). Doublets are difficult to identify because just because two cells are captured together does not mean there is simply more absolute RNA fragments present or sequenced in doublet cases, and cells exist on a transcriptional continuum, making identification of valid singlet cells difficult, especially if cells are in a transitioning or reprogrammed state.



Schematic of how cells with lineage barcodes appear in the single-cell sequencer where droplets add a unique cell ID. The singlet depicted is one reaction droplet with a single cell such that there is a 1:1:1 mapping of a cell to a barcode to a cell ID. Alternatively, an example of a doublet is when one droplet has two or more cells, each having a unique barcode.

Downstream impact of doublets

Doublets are problematic because they can impact the conclusions of scRNA-seq downstream functional analyses. We found that doublets confound downstream analyses when we tested common downstream scRNA-seq data analysis protocols (differential expression, cell trajectory, clustering stability, and cell-cell communication) on datasets of various doublet percentages. A schematic for how these datasets were generated and examples of our results for differential expression are below:



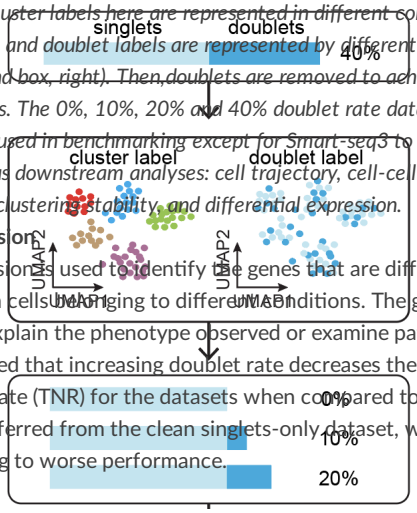
Schematic of datasets generated for all functional analyses. The dataset is first processed to have 40% doublets and undergo standard Seurat processing with Leiden clustering, where cluster labels here are represented in different colors in UMAP space (second box, left), and doublet labels are represented by different shades of blue in UMAP space (second box, right). Then,doublets are removed to achieve 20%, 10% and 0% doublet datasets. The 0%, 10%, 20% and 40% doublet rate datasets are generated for each dataset used in benchmarking except for Smart-seq3 to evaluate doublet impact on various downstream analyses: cell trajectory, cell-cell communication, clustering stability, and differential expression.

droplets add a unique cell ID. The singlet depicted is one reaction droplet with a single cell such that there is a 1:1:1 mapping of a cell to a barcode to a cell ID. Alternatively, an example of a doublet is when one droplet has two or more cells, each having a unique barcode.

Downstream impact of doublets

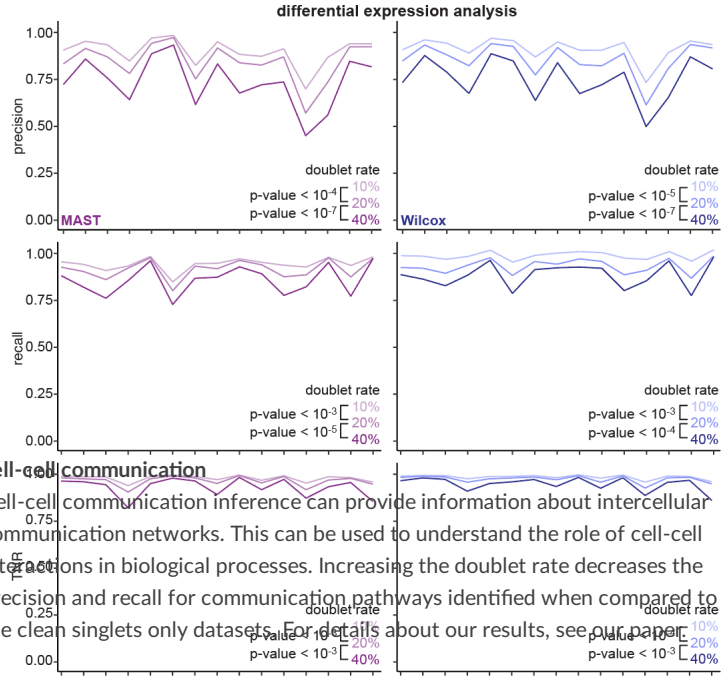
Doublets are problematic because they can impact the conclusions of scRNA-seq downstream functional analyses. We found that doublets confound downstream analyses when we tested common downstream scRNA-seq data analysis protocols (differential expression, cell trajectory, clustering stability, and cell-cell communication) on datasets of various doublet percentages. A schematic for how these datasets were generated and examples of our results for differential expression are below.

Schematic of datasets generated for all functional analyses. The dataset is first processed to have 40% doublets and undergo standard Seurat processing with Leiden clustering, where cluster labels here are represented in different colors in UMAP space (second box, left), and doublet labels are represented by different shades of blue in UMAP space (second box, right). Then,doublets are removed to achieve 20%, 10% and 0% doublet datasets. The 0%, 10%, 20% and 40% doublet rate datasets are generated for each dataset used in benchmarking except for Smart-seq3 to evaluate doublet impact on various downstream analyses: cell trajectory, cell-cell communication, clustering stability, and differential expression.



differential expression

Differential expression is used to identify the genes that are differentially expressed between cells belonging to different conditions. The genes identified could be used to explain the phenotype observed or examine pathways involved. Our results indicated that increasing doublet rate decreases the precision, recall, and true negative rate (TNR) for the datasets when compared to the differentially expressed genes inferred from the clean singlets-only dataset, with a higher doublet rate leading to worse performance.



Differential expression analysis results for all datasets using MAST (purple, left) and Wilcox (blue, right) on two randomly chosen clusters at various doublet rates. Line color shade represents the % doublet rate with a more saturated color indicative of a higher doublet rate. Precision, recall, and TNR are calculated against ground truth differentially expressed genes in the corresponding dataset without doublets. P-values between variable doublet rate datasets are calculated with a paired Wilcoxon signed-rank test.

cell-cell communication

Cell-cell communication inference can provide information about intercellular communication networks. This can be used to understand the role of cell-cell interactions in biological processes. Increasing the doublet rate decreases the precision and recall for communication pathways identified when compared to the clean singlets only datasets. For details about our results, see our paper.

cell trajectory

Increasing doublet rate caused the trajectory to deviate away from the trajectory inferred from the clean singlet only datasets and there were gain/loss of lineages when doublets were introduced. For details about our results, see our paper.

clustering stability

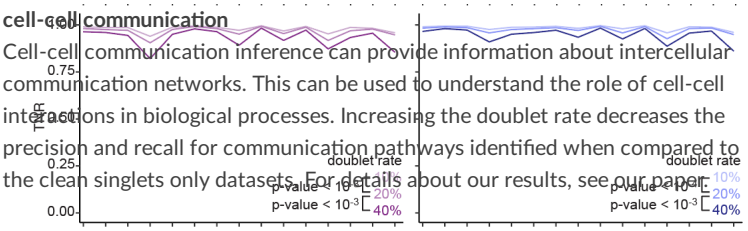
Clustering is one of the most common analyses done to infer the identity of similar cells. When the number of doublets were increased, the probability of getting the correct number of cell clusters decreased. This might lead to spurious clusters or a genuine sub-type of cells not being identified as a distinct group. For details about our results, see our paper.

what is singletCode?

singletCode is a framework to extract true singlets from barcoded scRNA-seq data. Our pipeline identified barcoded singlets from 10 different publications and 6 original experimental scRNA-seq samples generated in this study, encompassing 7 different barcoding technologies, 94 scRNA-seq samples, 3 unique sequencing technologies, and a total of 564,579 cells. Of the 338,948 barcoded cells, we extracted 293,618 singlets.(read more about the barcoded datasets we used singletCode with here).

cell-cell communication

Cell-cell communication inference can provide information about intercellular communication networks. This can be used to understand the role of cell-cell interactions in biological processes. Increasing the doublet rate decreases the precision and recall for communication pathways identified when compared to the clean singlets only datasets. For details about our results, see our paper.



Differential expression analysis results for all datasets using MAST (purple, left) and Wilcox (blue, right) on two randomly chosen clusters at various doublet rates. Line color shade represents the % doublet rate, with a more saturated color indicative of a higher doublet rate. Precision, recall and TNR are calculated against ground truth differentially expressed genes identified from corresponding dataset without doublets. P-values between variable doublet rate datasets are calculated with a paired Wilcoxon signed-rank test.

cell trajectory

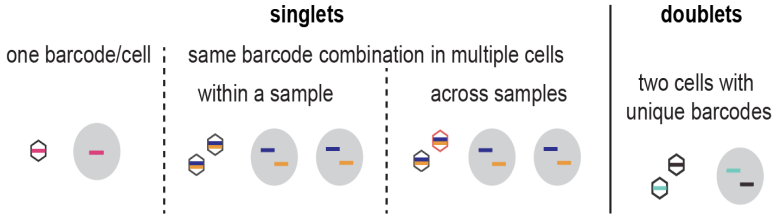
Increasing doublet rate caused the trajectory to deviate away from the trajectory inferred from the clean singlet only datasets and there were gain/loss of lineages when doublets were introduced. For details about our results, see our paper.

clustering stability

Clustering is one of the most common analyses done to infer the identity of similar cells. When the number of doublets were increased, the probability of getting the correct number of cell clusters decreased. This might lead to spurious clusters or a genuine sub-type of cells not being identified as a distinct group. For details about our results, see our paper.

What is singletCode?

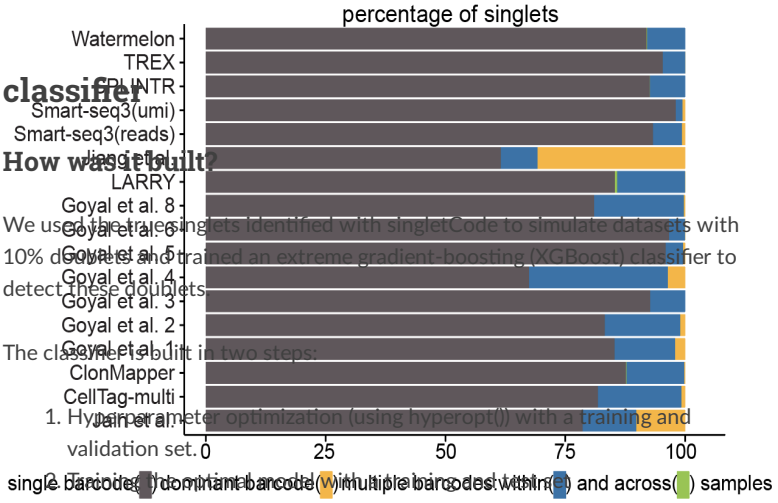
singletCode is a framework to extract true singlets from barcoded scRNA-seq data. Our pipeline identified barcoded singlets from 10 different publications and 6 original experimental scRNA-seq samples generated in this study, encompassing 7 different barcoding technologies, 74 scRNA-seq samples, 3 unique sequencing technologies, and a total of 564,579 cells. Of the 338,748 barcoded cells, we extracted 293,618 singlets.(read more about the barcoded datasets we used singletCode with here).



Singlet identification

singletCode leverages the fact that the lineage barcodes are present within the cell before sequencing. As a result of multiplicity of infection, singletCode identifies cells which meet any of the following conditions as a true singlet:

1. 1 barcode/1 cell ID
2. >1 barcode per 1 cell ID, but 1 barcode has significantly more UMI counts than the other barcodes within the same cell
3. M barcodes per 1 cell ID, but the same combination of M barcodes are found in other cells in the same sample
4. M barcodes per 1 cell ID, but the same combination of M barcodes are found in other cells across samples within the same experimental design (common in barcoding studies)



Stacked bar chart showing types of singlets in each dataset. Each color represents one type of [illegible] one cell ID has [illegible] has the highest [illegible] scenarios where [illegible] oss twin

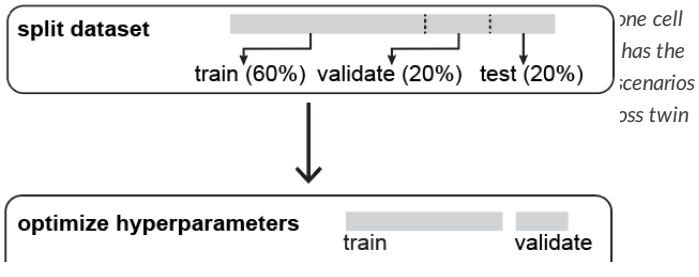
classifier

How was it built?

We used the true singlets identified with singletCode to simulate datasets with 10% doublets and trained an extreme gradient-boosting (XGBoost) classifier to detect these doublets.

The classifier is built in two steps:

1. Hyperparameter optimization (using hyperopt()) with a training and validation set.
2. Training the optimal model with training and validation set.



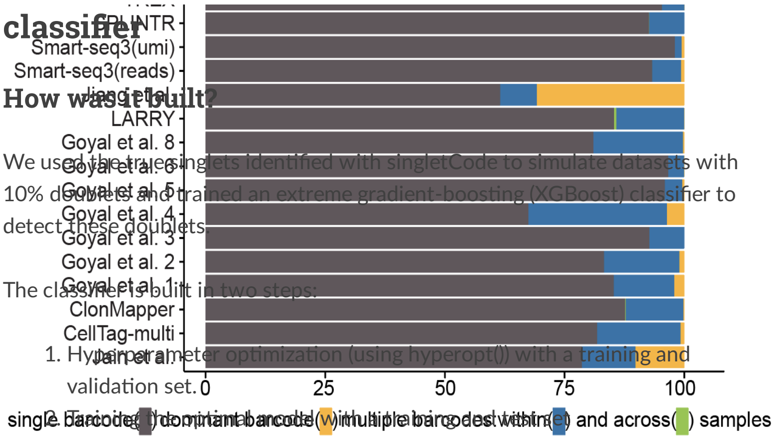
classifier

How was it built?

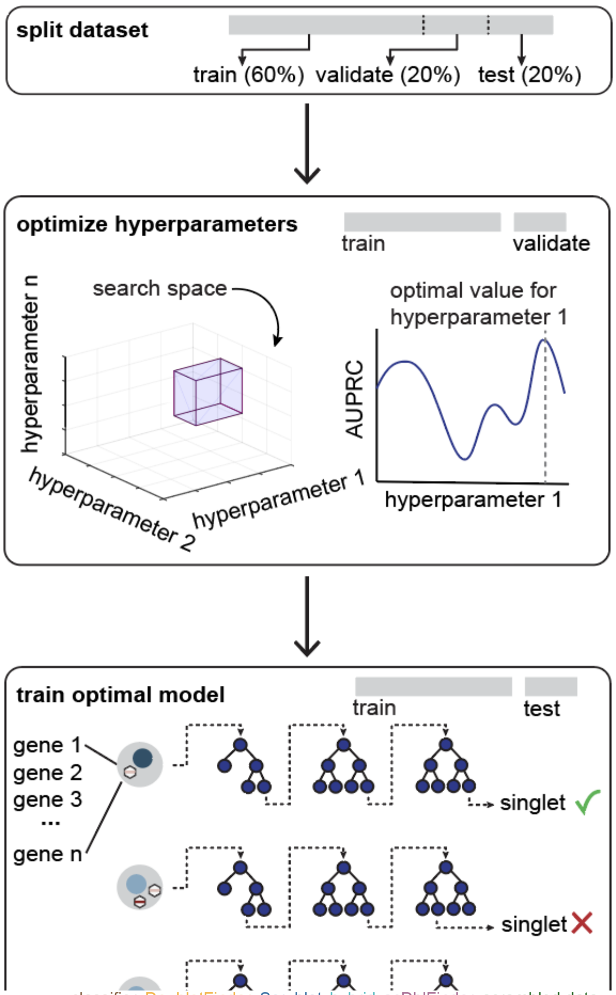
We used the true singlets identified with singletCode to simulate datasets with 10% doublets and trained an extreme gradient-boosting (XGBoost) classifier to detect these doublets.

The classifier is built in two steps:

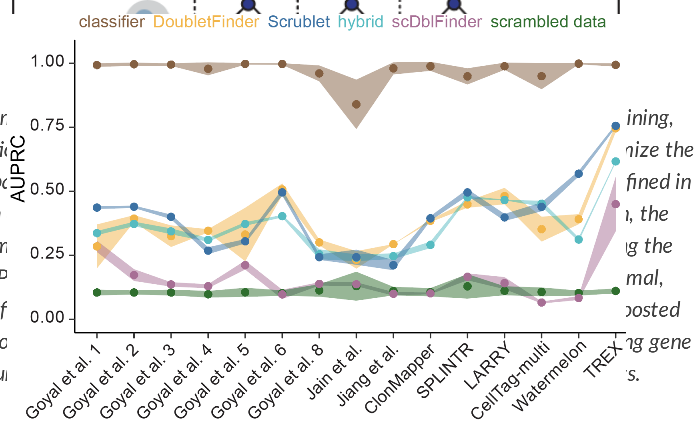
1. Hyperparameter optimization (using hyperopt()) with a training and validation set.
2. Training the optimal model



Stacked bar chart showing types of singlets in each dataset. Each color represents one type of ... one cell ID has ... has the highest ... scenarios where t... oss twin



Scher... validatio... hyperp... high... optim... AUP... classif... (XGBoo... cou... ining, ...ize the ...fined in ..., the ...g the ...mal, ...osted ...ng gene ...s.



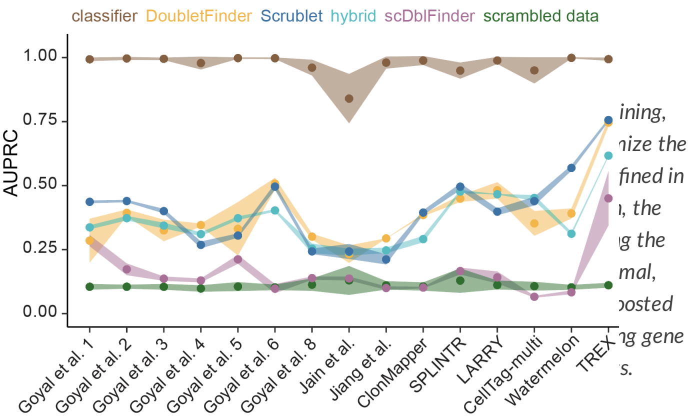
How well does it perform?

We achieved significantly higher AUPRC and AUROC scores using our classifier compared to the other methods benchmarked for doublet detection.

... compared to other doublet detection methods as measured by AUPRC. Each dot represents the average AUPRC score of a doublet detection method (color) on a given dataset (x-axis). The ribbon has the width of the standard deviation of the AUPRC score for each doublet detection method.

Classifying doublets in non-barcoded datasets

Although barcoding experiments are becoming increasingly prevalent, they are still relatively uncommon. Therefore, we sought to train a doublet classifier on barcoded data that could detect doublets in non-barcoded data. We trained a classifier on cell samples from melanoma, mouse brain, leukemia, and bone



Scher… validati… hyperp… high… optim… AUP… classif… (XGBoo… cou… …ining, …ize the …fined in …, the …g the …mal, …oosted …g gene …s.

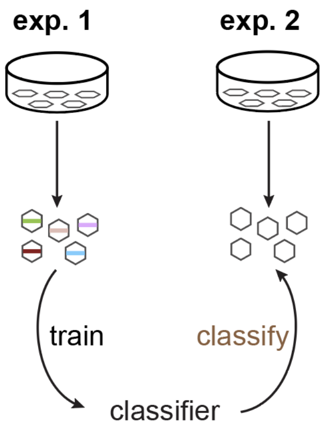
…ts of singletCode classification compared to other doublet detection methods as measured by AUPRC. Each dot represents the average AUPRC score of a doublet detection method (color) on a given dataset (x-axis). The ribbon marks the width of the standard deviation of the AUPRC score for each doublet detection method.

How well does it perform?

We achieved significantly higher AUPRC and AUROC scores using the classifier compared to the other methods benchmarked for doublet detection.

Classifying doublets in non-barcoded datasets

Although barcoding experiments are becoming increasingly prevalent, they are still relatively uncommon. Therefore, we sought to train a doublet classifier on barcoded data that could detect doublets in non-barcoded data. We trained a classifier on cell samples from melanoma, mouse brain, leukemia, and bone marrow/leukemia using true singlet labels from singletCode and successfully identified doublets in a similar cell sample without needing barcoding data. Further, we integrated all the data from mouse and human samples together, split it in half, and it to train a classifier that was then tested on individual sample components of the opposite half. This, too, proved to outperform other doublet detection methods.



Schematic of training a doublet classifier on barcoded data from 1 experiment and using that classifier to identify doublets in a biological replicate, experiment 2.

Non-barcoded datasets

As barcoded scRNA-seq data becomes more abundant, experimenters can train a classifier specific to their cell type using a barcoded subset of data, and, eventually, there could be enough data to train a classifier on multiple barcoded cell types which can be used more generally.

If the data is non-barcoded, the information from true singlets identified from singletCode can still be harnessed using a classifier trained on barcoded data from a different experiment using the cell type of interest. Find more information about this in the classifier page here.

Implementing singletCode on your data

benchmarking doublet detection methods

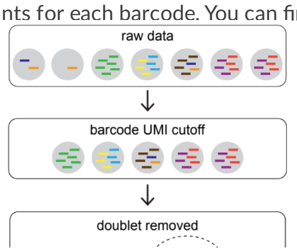
singletCode is a framework that can be extended to most scRNA-seq or even scATAC-seq data to accurately identify singlets.

Simulation datasets

To simulate doublets for benchmarking, we randomly selected the gene expression counts data from two cells that were found to be true singlets by singletCode. We averaged the counts from these two cells to generate simulated doublets and create datasets with various doublet percentages for benchmarking.

Barcoded datasets

We created both a python package called singletCode and a command-line interface. The only input needed is a .csv file with cell ID, barcode, sample information, and UMI counts for each barcode. You can find more information to use it here



Non-barcoded datasets

As barcoded scRNA-seq data becomes more abundant, experimenters can train a classifier specific to their cell type using a barcoded subset of data, and, eventually, there could be enough data to train a classifier on multiple barcoded cell types which can be used more generally.

If the data is non-barcoded, the information from true singlets identified from singletCode can still be harnessed using a classifier trained on barcoded data from a different experiment using the cell type of interest. Find more information about this in the classifier page here.

implementing singletCode on your data benchmarking doublet detection methods

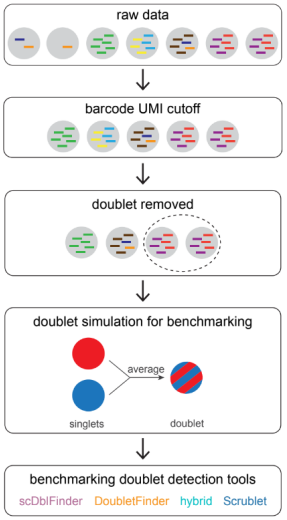
singletCode is a framework that can be extended to most scRNA-seq or even scATAC-seq data to accurately identify singlets.

Simulation datasets

Barcoded datasets

To simulate doublets for benchmarking, we randomly selected the gene expression counts data from two cells that were found to be true singlets by singletCode. We averaged the counts from these two cells to generate simulated doublets and create datasets with various doublet percentages for benchmarking.

We created both a python package called singletCode and a command-line interface. The only input needed is a .csv file with cell ID, barcode, sample information, and UMI counts for each barcode. You can find more information to use it here



Workflow schematic of how raw data is processed to generate true singlets based on a barcode UMI cutoff and doublet removal based on singletCode specifications, followed by the simulation of doublets for benchmarking doublet detection methods.

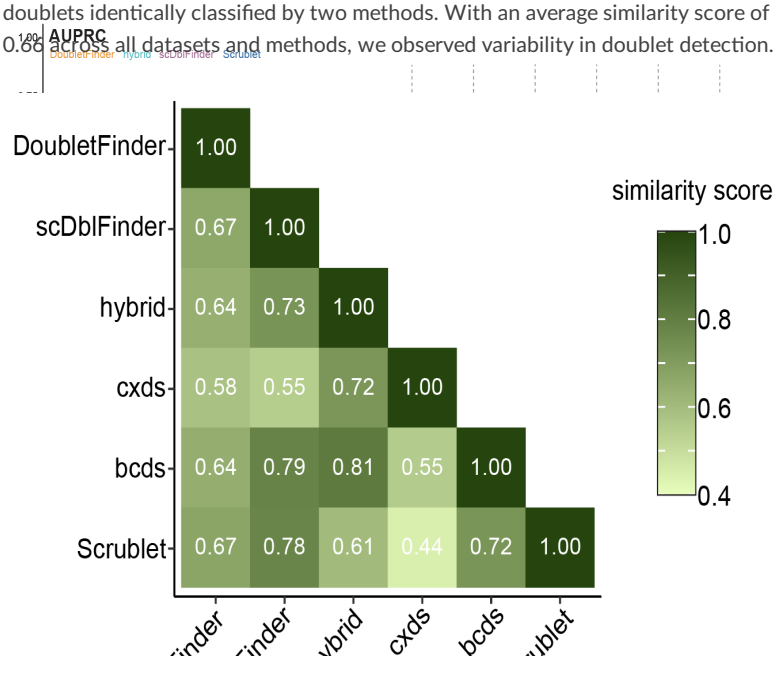
Benchmarking

We used these datasets to benchmark four doublet detection methods:

1. scDblFinder,
2. DoubletFinder,
3. Scrublet and
4. Hybrid

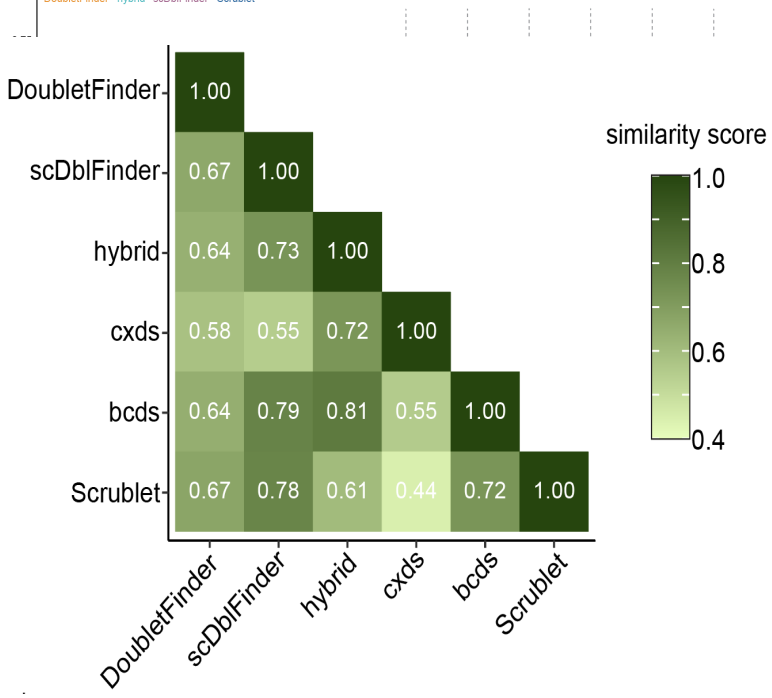
We evaluated the AUPRC, AUROC, TNR, and doublet scores and calls of the four methods and found lower than expected performance for all methods. A plot of our results for AUPRC value is found below.

We examined the consistency of doublet labeling across different doublet detection methods by introducing a 'similarity score'—a measure of the fraction of doublets identically classified by two methods. With an average similarity score of 0.66 across all datasets and methods, we observed variability in doublet detection.



methods and found lower than expected performance for all methods. A plot of our results for AUPRC value is found below.

We examined the consistency of doublet labeling across different doublet detection methods by introducing a 'similarity score'—a measure of the fraction of doublets identically classified by two methods. With an average similarity score of 0.66 across all datasets and methods, we observed variability in doublet detection.

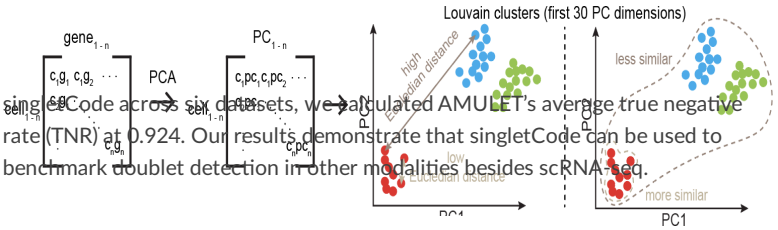


Pairwise similarity score between all benchmarked methods. A score of 1 indicates perfect identity and a 0 indicates complete disagreement. Average similarity score across all methods and datasets is 0.66.

We further evaluated doublet detection on ensemble doublet detection methods (hybrid, Chord) and across sequencing technologies (10X Genomics, Smart-seq3). For a more detailed evaluation of our results, refer to our paper.

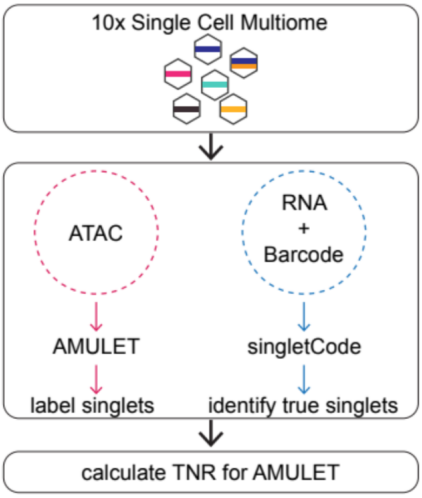
Heterogeneity effects

We wanted to know whether heterogeneity of a dataset affects the performance of doublet detection methods. Because heterogeneity can be impacted by many properties of a dataset, such as experimental design and data processing, we made conclusions based on heterogeneity within a sample. We did this by subsampling singlets and doublets within a single PC cluster for a sample (less heterogeneous, low Euclidean distance), and across all clusters for a sample (more heterogeneity, higher Euclidean distance).



singleCode across six datasets, we calculated AMULET's average true negative rate (TNR) at 0.924. Our results demonstrate that singletCode can be used to benchmark doublet detection in other modalities besides scRNA-seq.

Schematic of how t sampled for subsequent within- component analysis (left) was p e used to identify Louvain clusters, re eterogeneous) cells are identified f tance in PC space. Transcriptionally dis from all clusters within the dataset space.

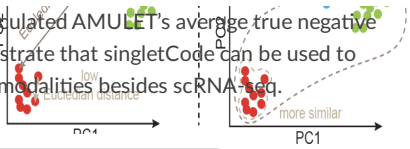


singletCode f

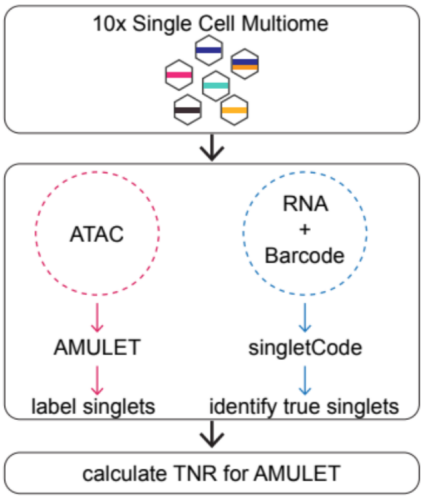
We evaluated our scATAC-seq datasets using AM erated Watermelon-barcc plied AMULET to scATAC-seq fragm urrently, we identified true sing same cells. By comparing the true negative and false positive rates between AMULET and

Schematic for 10X Single Cell Multiome dataset workflow resulting in TNR calculation. The same cells in the Multiome dataset undergo scATAC-seq, scRNA-seq, and barcode sequencing. Barcodes were processed with singletCode to identify ground truth singlets

singletCode across six datasets, we calculated AMULET's average true negative rate (TNR) at 0.924. Our results demonstrate that singletCode can be used to benchmark doublet detection in other modalities besides scRNA-seq.



Schematic of how t sampled for subsequent within- component analysis (left) was p e used to identify Louvain clusters, re ieterogeneous) cells are identified f tance in PC space. Transcriptionally dis rom all clusters within the dataset space.



singletCode fo

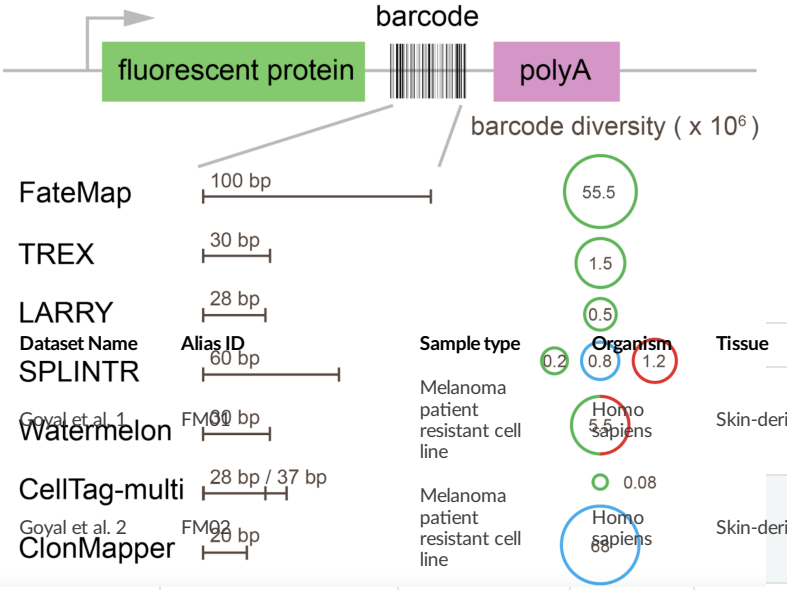
We evaluated our scATAC-seq datasets using AM erated Watermelon-barco plied AMULET to scATAC-seq fragm urrently, we identified true sing same cells. By comparing the true negative and false positive rates between AMULET and

Schematic for 10X Single Cell Multiome dataset workflow resulting in TNR calculation. The same cells in the Multiome dataset undergo scATAC-seq, scRNA-seq, and barcode sequencing. Barcodes were processed with singletCode to identify ground truth singlets while the respective fragments file from ATAC was used to label doublets with AMULET. The AMULET-labeled singlets were compared with ground truth singletCode singlets to calculate TNR.

dataset information

barcoding technologies assessed

7 different barcoding technologies We incorporated datasets using FateMap, ClonMapper, SPLINTR, LARRY, CellTag-multi, Watermelon, and TREX barcodes. Each have their own recovery and analysis specifications that we outline in Box 1 of our paper.



Schematic of the structure of different lineage-tracing barcodes, their length, diversity, and fluorescent colors. Each barcode contains an expressed fluorescent protein, a unique sequence, known as the barcode, and a polyA tail for capture. The lengths of barcodes in each method are shown as scale bars, and library diversities are shown as circles, sized proportional to their diversity and colored according to fluorescent proteins used.

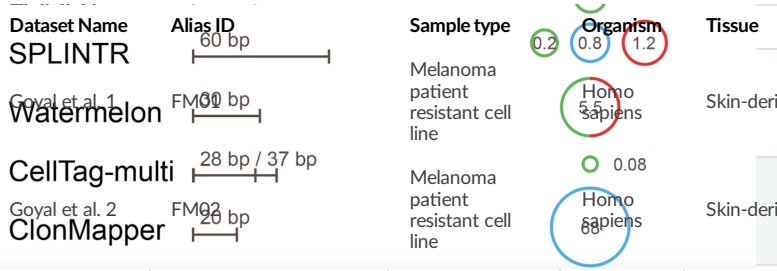
dataset details

Dataset Name	Alias ID	Sample type	Organism	Tissue	Cell line	Genotype	Treatment	Total sam
Goyal et al. 1	FM01	Melanoma patient resistant cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	Targeted therapy	4
Goyal et al. 2	FM02	Melanoma patient resistant cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	Targeted therapy	4
Goyal et al. 3	FM03	Melanoma patient resistant cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	Targeted therapy	2
Goyal et al. 4	FM04	Breast cancer patient resistant cell line	Homo sapiens	Breast; mammary gland	MDA-MB-231-D4	BRAF – G464V, KRAS – G13D, TERT – c.1-124C>T, TP53 – R280K	Chemotherapy, cytotoxic	2
Goyal et al. 5	FM05	Melanoma patient resistant cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	Targeted therapy	2
		Melanoma						

Specifications for barcoded and non-barcoded datasets

Dataset Name	Alias ID	Sample type	Organism	Tissue	Cell line	Genotype	Treatment	Total sam

barcoded scRNA-seq datasets



Schematic of the structure of different lineage-tracing barcodes, their length, diversity, and fluorescent color. Each barcode contains an expressed fluorescent protein, a unique sequence, known as the barcode, and a polyA tail for capture. The lengths of barcodes in each method are shown as scale bars, and library diversities are shown as circles, sized proportional to their diversity and colored according to fluorescent proteins used.

dataset details

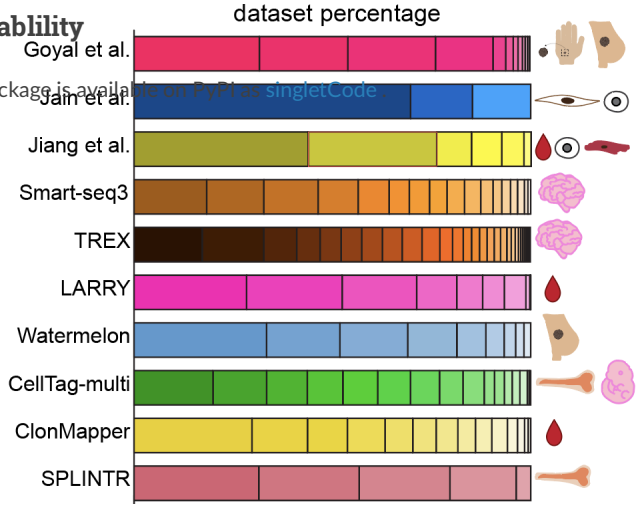
Specifications for barcoded and non-barcoded datasets

Dataset Name	Alias ID	Sample type	Organism	Tissue	Cell line	Genotype	Treatment	Total sam
Goyal et al. 1	FM01	Melanoma patient resistant cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	Targeted therapy	4
Goyal et al. 2	FM02	Melanoma patient resistant cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	Targeted therapy	4
Goyal et al. 3	FM03	Melanoma patient resistant cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	Targeted therapy	2
Goyal et al. 4	FM04	Breast cancer patient resistant cell line	Homo sapiens	Breast; mammary gland	MDA-MB-231-D4	BRAF – G464V, KRAS – G13D, TERT – c.1-124C>T, TP53 – R280K	Chemotherapy, cytotoxic	2
barcoded scRNA-seq datasets								
Goyal et al. 6	FM06	Melanoma patient untreated cell line	Homo sapiens	Skin-derived	WM989 A6-G3	BRAF – V600E	No treatment	2
Goyal et al. 8	FM08	Primary melanocytes	Homo sapiens	Skin-derived	FOM230-1	WT	No treatment	2
Jiang et al.	non_cancer	HIPS differentiation	Homo sapiens	PBMCs	PENN123i-SV20	WT	Cardiac differentiation signal	2
Jain et al.	Biorxiv	Stem cell reprogramming	Homo sapiens	Fibroblasts	hiF-T	WT	OKSM	6
TREX	TREX	neuroepithelial progenitor cells differentiation	Mus musculus	Neural	CD-1	•	No treatment	4
LARRY	LARRY, LK	pan-myeloid differentiation	Mus musculus	Blood	Lin-Kit+	•	cytokines and growth factors	18
LARRY	LARRY, LSK	pan-myeloid differentiation	Mus musculus	Blood	Lin-Kit+Sca-1+	•	cytokines and growth factors	15
SPLINTR	SPLINTR_chemo	Acute myeloid leukaemia	Mus musculus	Bone marrow	C57BL/6	MLL-AF9 + KrasG12D	chemotherapy	10
nonbarcoded sc-RNA seq datasets								
SPLINTR	SPLINTR_clone	Acute myeloid leukaemia	Mus musculus	Bone marrow	C57BL/6	MLL-AF9 + KrasG12D	No treatment	4
Smart-seq3	Smart-seq3	neuroepithelial progenitor cells differentiation	Mus musculus	Neural	CD-1	MLL-AF9, MLL-AF9 + KrasG12D, MLL-AF9 + Flt3ITD	No treatment	2
SPLINTR	SPLINTR_chemoretrans	Acute myeloid leukaemia	Mus musculus	Bone marrow and spleen	C57BL/6	MLL-AF9 + KrasG12D	No treatment	1
non-barcoded Watermelon	hm-12k Watermelon	Synthetic dataset	Homo sapiens, Mus musculus	Human kidney, Murine mammary fibroblasts	HEK293T, NIH3T3	WT	No treatment	1
non-barcoded CellTag-multi	hm-6k CellTag-multi_d	Synthetic multilineage differentiation (hematopoiesis)	Homo sapiens, Mus musculus	Human kidney, Murine bone marrow fibroblasts	HEK293T, NIH3T3 T-47D	PIK3CA – H1047R; Tp53 – L194F	Targeted therapy	6
non-barcoded CellTag-multi	HMEC-orig-MULTI CellTag-multi_B4	HMECs iEP reprogramming	Homo sapiens Mus musculus	mammary gland embryo	HMEC C57BL/6J	WT •	No treatment EGF	1 1
non-barcoded	HMEC-rep-MULTI	HMECs	Homo sapiens	mammary gland	HMEC	WT	No treatment	1
ClonMapper	ClonMapper	CLL resistant cell lines	Homo sapiens	blood	HG3	•	Chemotherapy	4
non-barcoded	HEK-HMEC-MULTI	Synthetic dataset	Homo sapiens	Human kidney, mammary gland	HEK293T, HMEC	WT	No treatment	1

Dataset Name	Alias ID	Sample type	Organism	Tissue	Cell line	Genotype	Treatment	Total sam
SPLINTR	SPLINTR_clone	Acute myeloid leukaemia	Mus musculus	Bone marrow	C57BL/6	MLL-AF9 + KrasG12D, MLL-AF9 + Flt3ITD	No treatment	4
Smart-seq3	Smart-seq3	neuroepithelial progenitor cells differentiation	Mus musculus	Neural	CD-1	•	No treatment	2
SPLINTR	nonbarcoded sc-RNA seq datasets SPLINTR_chemoretrans seq datasets	Acute myeloid leukaemia	Mus musculus	Bone marrow and spleen	C57BL/6	MLL-AF9 + KrasG12D	No treatment	1
non-barcoded Watermelon	hm-12k Watermelon	Synthetic dataset Breast cancer patient resistant cell line	Homo sapiens, Mus musculus	Human kidney, Murine mammary fibroblasts gland	HEK293T, NIH3T3 T47D	PIK3CA – H1047R; Tp53 – L194F	No treatment Targeted therapy	1 6
non-barcoded CellTag-multi	hm-6k CellTag-multi_d	Synthetic dataset multilineage differentiation (hematopoiesis)	Homo sapiens, Mus musculus	Human kidney, Murine bone marrow fibroblasts	HEK293T, NIH3T3 LSK	WT	No treatment cytokines and growth factors	1 3
non-barcoded CellTag-multi	HMEC-orig-MULTI CellTag-multi_B4	HMECs iEP reprogramming	Homo sapiens Mus musculus	mammary gland embryo	HMEC C57BL/6J	WT •	No treatment EGF	1 1
non-barcoded	HMEC-rep-MULTI	HMECs	Homo sapiens	mammary gland	HMEC	WT	No treatment	1
ClonMapper	ClonMapper	CLL resistant cell lines	Homo sapiens	blood	HG3	•	Chemotherapy	4
non-barcoded	HEK-HMEC-MULTI	Synthetic dataset	Homo sapiens	Human kidney, mammary gland	HEK293T, HMEC	WT	No treatment	1
non-barcoded	mkidney-ch	Mouse kidney cells	Mus musculus	Mouse kidney	C57BL/6J	WT	No treatment	1
non-barcoded	pbmc-2ctrl-dm	Patient PBMCs	Homo sapiens	Systemic lupus erythematosus (SLE) PBMCs	Patient-derived	Patient-derived	No treatment	1
non-barcoded	pbmc-2stim-dm	Patient PBMCs	Homo sapiens	Systemic lupus erythematosus (SLE) PBMCs	Patient-derived	Patient-derived	No treatment	1
non-barcoded	cline-ch	Human HEK, K562, KG1, and THP1	Homo sapiens	blood	human - derived	human - derived	No treatment	1
non-barcoded	pbmc-ch	Human PBMCs	Homo sapiens	blood	human - derived	human - derived	No treatment	1
non-barcoded	pdx-MULTI	Synthetic dataset	Homo sapiens, Mus musculus	Human breast cancer, mouse immue	PDX mouse model	PDX mouse model	No treatment	1
non-barcoded	nuc-MULTI	Synthetic dataset	Homo sapiens, Mus musculus	nuclei	•	•	No treatment	1
	Multiome datasets							
ATAC	Watermelon Multiome	Breast cancer patient resistant cell line	Homo sapiens	Breast; mammary gland	MCF7	GATA3 – D336G; PIK3CA – E545K	Targeted therapy	6

Stacked bar chart of cell types in all datasets. Each distinct color corresponds to one study, with the proportions of different cell types arranged from most to least prevalent (deepest to lightest) and separated by vertical lines. Adjacent to the bar chart, sketched tissues and cells provide a visualization of the cell types used in the particular study.

singletCode PyPI package



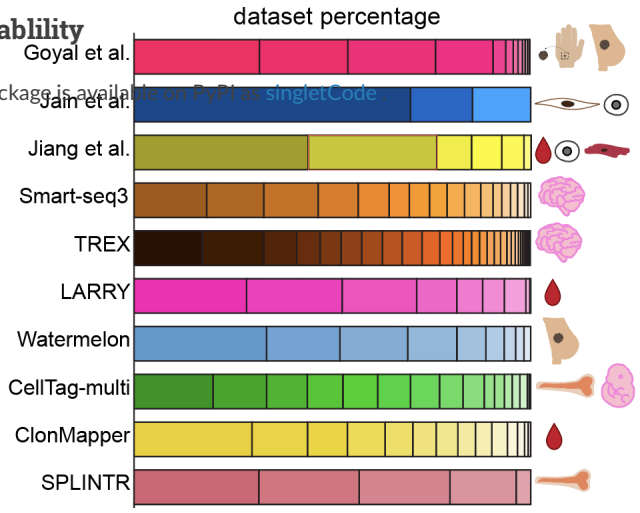
Availablility

The package is available on PyPI as singletCode .

Multiome datasets							
ATAC	Watermelon Multiome	Breast cancer patient derived resistant cell line	Homo sapiens	breast; mammary gland	MCF7	GATA3 – D336G; PIK3CA – E545K	Targeted therapy	6

Stacked bar chart of cell types in all datasets. Each distinct color corresponds to one study, with the proportions of different cell types arranged from most to least prevalent (deepest to lightest) and separated by vertical lines. Adjacent to the bar chart, sketched tissues and cells provide a visualization of the cell types used in the particular study.

singletCode PyPI package



Availablility

The package is available on PyPI as singletCode .

How do I use it?

1. Installation

   It can be installed from PyPI using the following in the terminal:

   ```
   pip3 install singletCode
   from singletCode import check_sample_sheet, get_singlets
   ```

2. Preparing the input sample sheet.

   The input sample sheet is a .csv file that contains the information about cell ID (added while sequencing), lineage barcode, and sample name. Each row should be repeated n times where n is the number of UMIs associated with that barcode and cell ID combination. For creating the input, you can read the .csv file in as a pandas dataframe.

   ```
   import pandas as pd
   df = pd.read_csv("path/to/csv/file.csv")
   ```

You can check if the format and the column names are valid for running

How do I use it?

1. Installation

   It can be installed from PyPI using the following in the terminal:

```
pip3 install singletCode
from singletCode import check_sample_sheet, get_singlets
```

2. Preparing the input sample sheet.

   The input sample sheet is a .csv file that contains the information about cell ID (added while sequencing), lineage barcode, and sample name. Each row should be repeated n times where n is the number of UMIs associated with that barcode and cell ID combination. For creating the input, you can read the .csv file in as a pandas dataframe.

```
import pandas as pd
df = pd.read_csv("path/to/csv/file.csv")
```

   You can check if the format and the column names are valid for running singletCode using check_sample_sheet() function.

```
check_sample_sheet(df)
```

   It will either return an error with information about how to modify the sample sheet to make it a valid input or will print the message, “The sample sheet provided can be used as input to get_singlets to get a list of singlets identified”, in which case you can move to the next step.

3. Running get_singlets() to get an assignment of singlet status for each cell ID and barcode combination.

   This is the step where singlet identification is done using the singletCode framework. You can read more about the parameters for get_singlets() below. In this example, default values are used.

```
cellLabelList, stats = get_singlets(df, dataset_name = "Sample1")
```

   cellLabelList is a pandas dataframe that contains 5 rows: cellID, barcode, sample, nUMI and label. Label is whether the particular cell ID and barcode combination has been called a singlet or not.

   To get a dataframe containing just the singlets, you can run this:

```
singletList = cellLabelList[cellLabelList["label"] == "Singlet"]
```

   total cells, total number of singlets, number of singlets recovered from different categories of singlets (such as single-barcode singlets, muli-barcode singlets, dominant-UMI singlets), number of cells removed due to low barcode UMI counts for the barcode and number of indeterminate cells since singletCode can identify only truly singlet cells but not be certain if the other cells are truly not singlets.

   You can save the singlets list in .txt file and the list of cellIDs in each category of singlets by setting **save_all_singlet_categories** parameter to TRUE.

Detailed information about the parameters for the functions in the package

get_singlets

`get_singlets`(*sample_sheet, output_path=None, dataset_name=None, save_all_singlet_categories=False, save_plot_umi=False, umi_cutoff_method='ratio', umi_cutoff_ratio=7.5e-06, umi_cutoff_percentile=None, min_umi_cutoff=2, umi_diff_threshold=50, umi_dominant_threshold=10*) [source]

Function that inputs the sample sheet and other parameters and runs it through count_doublets to get a list of singlets in the sample. If a row is repeated, it is assumed to reflect the UMI associated with a barcode in this cell (identified by the cellID).

Parameters
- **sample_sheet** (*DataFrame*) – A dataframe that contains 3 columns: cellID, barcode, sample.
- **dataset_name** (*str*) – The name of the dataset being analysed. It will be in the name of all saved files and be a column in the singlet_stats sheet returned.
- **output_path** (*str, optional*) – The path to store any output files, including plots to show UMI distribution and what the umi_cutoff used is, csv files containing singlets of different categories. If None, then the list of singlets will

Detailed information about the parameters for the functions in the package

get_singlets

get_singlets(*sample_sheet*, *output_path=None*, *dataset_name=None*, *save_all_singlet_categories=False*, *save_plot_umi=False*, *umi_cutoff_method='ratio'*, *umi_cutoff_ratio=7.5e-06*, *umi_cutoff_percentile=None*, *min_umi_cutoff=2*, *umi_diff_threshold=50*, *umi_dominant_threshold=10*) [source]

Function that inputs the sample sheet and other parameters and runs it through count_doublets to get a list of singlets in the sample. If a row is repeated, it is assumed to reflect the UMI associated with a barcode in this cell (identified by the cellID).

Parameters

- **sample_sheet** (*DataFrame*) – A dataframe that contains 3 columns: cellID, barcode, sample.
- **dataset_name** (*str*) – The name of the dataset being analysed. It will be the suffix in all files saved.
- **output_path** (*str*, *optional*) – The path to store any output files, including plots to show UMI distribution and what the umi_cutoff used is, csv files containing singlets of different categories. If None, then the list of singlets will be returned but it won't contain information about what category of singlet each cell is. Defaults to None.
- **save_all_singlet_categories** (*bool*, *optional*) – If true, then singlets of each category are saved separately in csv files along with all singlets and all non-singlets. Defaults to False.
- **save_plot_umi** (*bool*, *optional*) – If true, then plots showing UMI distribution indicating the UMI cutoff used will be saved for each sample. Defaults to False.
- **umi_cutoff_method** (*str*, *optional*) – Specify if quality control for barcodes using UMI counts should be based on "ratio" or "percentile". Defaults to 'ratio'.
- **umi_cutoff_ratio** (*float*, *optional*) – The ratio used to determine the umi_cutoff if umi_cutoff_method is "ratio". Defaults to 3/4e5.
- **umi_cutoff_percentile** (*float*, *optional*) – If umi_cutoff_method is "percentile", then the umi_cutoff will be the minimum UMI count required to be in the top umi_cutoff_percentile'th percentile. There is no default and if umi_cutoff_method is set to "percentile", then manually set this parameter.
- **min_umi_cutoff** (*int*, *optional*) – This is the absolute minimum number of UMIs that need to be associated with a barcode for it to be considered a barcode. However, the actual umi_cutoff used will be the greater of min_umi_cutoff and the cutoff calculated using umi_cutoff_method. Defaults to 2.
- **umi_dominant_threshold** (*int*, *optional*) – The minimum UMI count to be associated with a barcode for it to be considered to be a potential dominant barcode in a cell. Default is 10.
- **umi_diff_threshold** (*int*, *optional*) – This is the minimum difference between UMI counts associated with a potential dominant barcode present within a cell and the median UMI count of all barcodes associated with the cell. If a cell has only one dominant barcode, it will be counted. Defaults to 50.

Return type tuple

Returns A 2-tuple containing:

- pandas.DataFrame: A dataframe which contains all unique cellID and barcode combinations in the data along with singlet assignment to each cell ID.
- pandas.DataFrame: A dataframe which contains the statistics for total singlets, different categories of singlets, and cells removed due to low UMI counts.

check_sample_sheet

check_sample_sheet(*data_frame*) [source]

Function to check if the dataframe can be used as input to get_singlets function. It checks if the three columns - cellID, barcode, sample needed are present and in same order. If this dataframe can be used for get_singlets function, then a statement will be printed to confirm that.

Parameters **sample_sheet** – A dataframe that contains your sample sheet.

Returns None

command line interface

Availability

The source code for the command line tool is available in our GitHub repository (repo). You can clone the repository to access the code locally.

```
git clone https://github.com/goyallab/singlercodetools/
```

Return type tuple

check_sample_sheet

`check_sample_sheet(data_frame)` [source]

Function to check if the dataframe can be used as input to get_singlets function. It checks if the required columns - cellID, barcode, sample needed are present and in same order. If this dataframe can be used for get_singlets function, then a statement will be printed to confirm that

Parameters **sample_sheet** – A dataframe that contains your sample sheet.

Returns None

command line interface

Availability

The source code for the command line tool is available in our GitHub repository (repo). You can clone the repository to download the code locally.

```
git clone https://github.com/GoyalLab/singletCodeTools/
```

Using the interface

Navigating to commandLineTools folder, you will find 3 files and you will need to run singletCodeCommandLine.py. There are 2 modules available, one to run singletCode and the other to create a sample sheet if you have used Watermelon barcoding techmology using the fastq sequenced files from MISEQ.

Detailed information about the modules in the command line interface

Description:

This script contains two modules: - Count Module: Generates the singlet files for the input data sheet. - Watermelon Module: Uses the MiSeq dial-out files to create the cell ID, barcode, and sample file. These outputs can then be used as input for the singlet code module.

Usage:

For the Count Module:

```
python singletCode.py count -i /path/to/input.txt -o /path/to/output
```

For the Watermelon Module:

```
python singletCode.py watermelon -i /path/to/fastq/files -o
path/to/save/csv/file -s path/sample/sheet -use10X False -input10X
path/to/barcodes/tsv
```

Options for Count Module:

- -i, –input_file: Specify the path to the input barcode file.
- -o, –output_prefix: Specify the path to the output prefix.
- -f, –force: Force overwrite if the output file already exists.
- -u, –cutoff: UMI cutoff ratio.
- -d, –umi_diff_threshold: Minimum difference in UMI between UMI count for dominant UMI and median UMI count.
- -m, –min_umi_good_data_cutoff: Minimum UMI count for a barcode to be considered.
- -g, –min_umi_filter_threshold: Minimum UMI filter threshold.

Options for Watermelon Module:

- -i, –inputFolder: Specify the path to the folder containing the fastq folders output from MiSeq.
- -s, –sampleSheet: Specify the path to the sample sheet in .csv format that contains sample name and sample number. This should match the names of the fastq files (e.g. Sample-1_S1_L001_R1_001.fastq.gz).
- -o, –outputFolder: Specify the path to save the output CSV file containing the barcode, cell ID information.
- -outputName: Specify the name of the output CSV file.
- –use10X: Specify if a 10X object is provided which has the same cells as those in fastq. If provided, cells in the fastq file will be filtered out if not present in the 10X object.
- -input10X: Path to the barcodes.tsv.gz or barcodes.tsv which contains cell IDs.

Authors:

- Ziyang Zhang (Charles)
- Keerthana M Arun

Vignette to use singletCode Command Line tool to analyse single-cell data with watermelon barcodes

The barcode region is assumed to be amplified using Illumina MiSeq.

All the data for this vignette and the files output from it can be downloaded here.

Parameters **umi_diff_threshold** (*int*, *optional*) – This is the minimum difference between UMI counts associated with a potential dominant barcode present within a cell and the median UMI count of all barcodes associated with the cell. If a cell has only one dominant barcode, it will be counted. Defaults to 50.

Returns A 2-tuple containing:

- pandas.DataFrame: A dataframe with information only on cell ID and barcode combinations in the data along with singlet assignment to each cell ID.
- pandas.DataFrame: A dataframe which contains the statistics for total singlets, different categories of singlets, and cells removed due to low UMI counts.

For the Watermelon Module:

```
python singletCodeCommandLine.py watermelon -i path/to/fastq/files -o
path/to/save/csv/file -s path/sample/sheet -use10X False -input10X
path/to/barcodes/tsv
```

- -i, –input_file: Specify the path to the input barcode file.
- -o, –output_prefix: Specify the path to the output prefix.
- -f, –force: Force overwrite if the output file already exists.
- -u, –cutoff: UMI cutoff ratio.
- -d, –umi_diff_threshold: Minimum difference in UMI between UMI count for dominant UMI and median UMI count.
- -m, –min_umi_good_data_cutoff: Minimum UMI count for a barcode to be considered.
- -g, –min_umi_filter_threshold: Minimum UMI filter threshold.

Options for Watermelon Module:

- -i, –inputFolder: Specify the path to the folder containing the fastq folders output from MiSeq.
- -s, –sampleSheet: Specify the path to the sample sheet in .csv format that contains sample name and sample number. This should match the names of the fastq files (e.g. sampleName_S1_L001_R1_001.fastq.gz).
- -o, –outputFolder: Specify the path to save the output CSV file containing the barcode, cell ID information.
- -outputName: Specify the name of the output CSV file.
- –use10X: Specify if a 10X object is provided which has the same cells as those in fastq. If provided, cells in the fastq file will be filtered out if not present in the 10X object.
- -input10X: Path to the barcodes.tsv.gz or barcodes.tsv which contains cell IDs.

Options for Count Module:

Authors:

- Ziyang Zhang (Charles)
- Keerthana M Arun

Vignette to use singletCode Command Line tool to analyse single-cell data with watermelon barcodes

The barcode region is assumed to be amplified using Illumina MiSeq.

All the data for this vignette and the files output from it can be downloaded here. It contains inputFiles and the outputFiles (it contains a test folder which has the expected output files). This vignette can be downloaded as a jupyter notebook from the singletCode Tools repo.

Installing singletCode Command line tool

To use the singletCode command line tool, clone the repository from GitHub. Let the Path to the folder you are running this command be **Path**. We can also install other packages needed to run the tool.

```
!git clone https://github.com/GoyalLab/singletCodeTools
Path = "path/to/singletCodeTools/repo"
%conda install scipy tqdm matplotlib biopython python-levenshtein pandas
```

Next step is to understand the samples present in the FASTQ files.

The sample fastq files are in the inputFolder. We can identify the sample name and number from the FASTQ file. For example, sampleName_S1_L001_R1_001.fastq.gz means that the sample name is sampleName and sample number is 1.Make sure that both read 1 and read 2 for each sample are present in the same folder (R1 and R2)

p/inputFiles/ 2. **outputFolder** will be p/outputFiles/ 3. **sampleSheet** will be p/inputFiles/sampleSheet.csv

Creating sample sheet for these two samples.

```
import subprocess
import pandas as pd
command = [
    'python',
    f'{Path}/commandLine/singletCodeCommandLine.py',
    'watermelon',
    '-i',  f'{p}/inputFiles',
    '-o',  f'{p}/outputFiles',
    '-s', f'{p}/inputFiles/sampleSheet.csv',
    '--outputName', 'watermelonBarcodeUmi.csv'
]
sampleSheet = pd.read_csv(f"{p}/inputFiles/sampleSheet.csv")
sampleSheet
```

	sampleName	sampleNumber
0	sampleName	1
1	otherSampleName	2

Now, to run the watermelon module of singletCodeTools, you need to run this command. If we are going by the folder structure of the zipped file and **p** is *path to the unzipped folder containing example files*, then 1. **inputFolder** will be

```
Arguments Received:
  command: watermelon
  inputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  sampleSheet:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputName: watermelonBarcodeUmi.csv
  use10X: False
```

p/inputFiles/ 2. **outputFolder** will be p/outputFiles/ 3. **sampleSheet** will be p/inputFiles/sampleSheet.csv

Creating sample sheet for these two samples.

```
import subprocess
import pandas as pd
command = [
    'python',
    f'{Path}/commandLine/singletCodeCommandLine.py',
    'watermelon',
    '-i', f'{p}/inputFiles/',
    '-o',  f'{p}/outputFiles',
    '-s', f'{p}/inputFiles/sampleSheet.csv',
    '--outputName', 'watermelonBarcodeUmi.csv'
]

sampleSheet = pd.read_csv(f"{p}/inputFiles/sampleSheet.csv")
result = subprocess.run(command)
sampleSheet
```

```
Arguments received:
  command: watermelon
  inputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  sampleSheet:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputName: watermelonBarcodeUmi.csv
  use10X: False
  input10X: None
All the inputs for the command are valid and will proceed with creating the
barcode sheet for all the samples in the sheet.
Filtered rows of dataframe: 940
Filtered rows of dataframe: 718
```

	sampleName	sampleNumber
0	sampleName	1
1	otherSampleName	2

Now, to run the watermelon module of singletCodeTools, you need to run this command. If we are going by the folder structure of the zipped file and **p** is *path to the unzipped folder containing example files*, then 1. **inputFolder** will be

```
result = subprocess.run([
    'python',
    f'{Path}/commandLine/singletCodeCommandLine.py',
    'watermelon',
    '-i', f'{p}/inputFiles/',
    '-o', f'{p}/outputFiles/',
    '-s', f'{p}/inputFiles/sampleSheet.csv',
    '--outputName', 'watermelonBarcodeUmiWith10X.csv',
    '--use10X',
    '--input10X', f'{p}/inputFiles/barcodes.tsv'
], capture_output=True, text=True)

# Check if the command was successful
if result.returncode == 0:
    print("Command executed successfully")
    print("Output:\n", result.stdout)
else:
    print("Command failed")
    print("Error:\n", result.stderr)
```

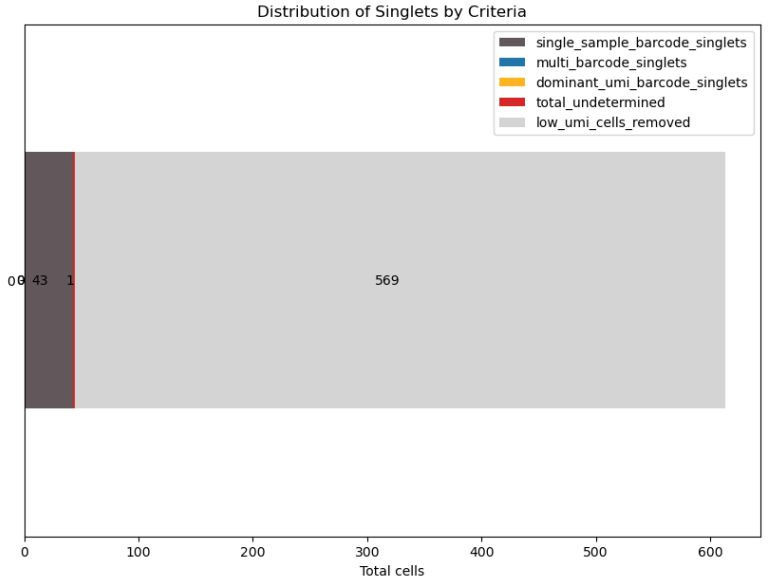
```
Command executed successfully
Output:
 Arguments received:
  command: watermelon
  inputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  sampleSheet:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputName: watermelonBarcodeUmiWith10X.csv
  use10X: True
  input10X:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
All the inputs for the command are valid and will proceed with creating the
barcode sheet for all the samples in the sheet.
Filtered rows of dataframe: 791
Filtered rows of dataframe: 629
```

```
Command executed successfully
Output:
 Arguments received:
  command: count
  input_file:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  out_prefix:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  umi_cutoff_ratio: 7.5e-06
  umi_diff_threshold: 50
  dominant_threshold: 10
```

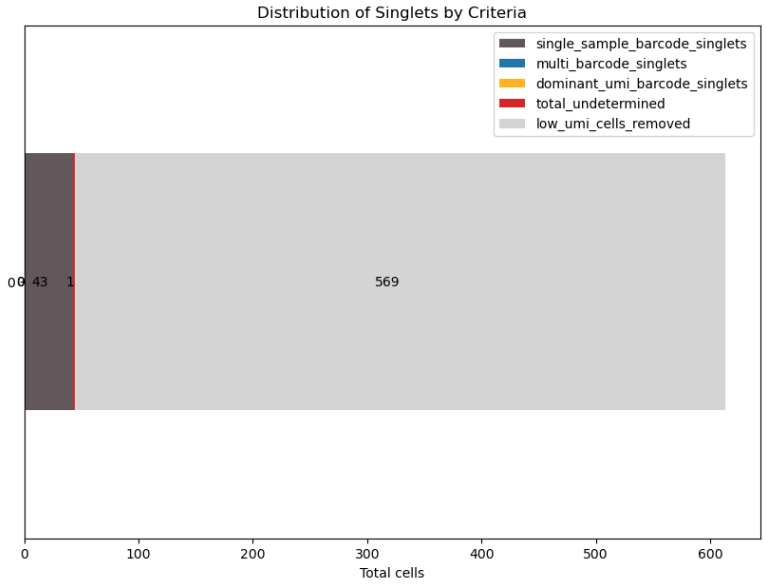
```
Command executed successfully
Output:
 Arguments received:
  command: watermelon
  inputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputFolder:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  sampleSheet:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  outputName: watermelonBarcodeUmiWith10X.csv
  umi10X: True
  inputs10X:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
All the inputs for the command are valid and will proceed with creating the
barcode sheet for all the samples in the sheet.
Filtered rows of dataframe: 791
Filtered rows of dataframe: 629
```

```
Command executed successfully
Output:
 Arguments received:
  command: count
  input_file:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  out_prefix:
/home/keerthana/Goyal_Lab/websiteToolData/thingsToAddToWebsite/watermelonVignet
  umi_cutoff_ratio: 7.5e-06
  umi_diff_threshold: 50
  dominant_threshold: 10
  min_umi_good_data_cutoff: 2
INFO: Raw data counts
sampleNum
sampleName         693
otherSampleName    524
Name: count, dtype: int64
INFO: Using raio based filtering.
Current Sample Adjusted UMI cutoff: 2
Total cells: 45
Sample sampleName singlet: 43
Total Singlets: 43
Total Multiplets: 1
All singlets identified are unique? True
Total Singlets: 43
Total Multiplets: 1
INFO: Using raio based filtering.
Current Sample Adjusted UMI cutoff: 2
Total cells: 22
Sample otherSampleName singlet: 22
Total Singlets: 22
Total Multiplets: 0
All singlets identified are unique? True
Total Singlets: 22
Total Multiplets: 0
All singlets identified are unique? True
```

```
import matplotlib.pyplot as plt

stats = pd.read_csv(f"
```



In the above plot, you see that the original data had 569 cells that were removed due to low barcode UMI count, 43 singlets with a single-barcode associated with them and 1 multiplet (singletCode could not determine if it was a singlet for sure.)



In the above plot, you see that the original data had 569 cells that were removed due to low barcode UMI count, 43 singlets with a single-barcode associated with them and 1 multiplet (singletCode could not determine if it was a singlet for sure.)

Looking at the scRNAseq data associated

Since this data has both scRNAseq and barcodes for the same cells, we can analyse them together

Installing and importing scanpy package to do this

```
#Install scanpy for further single-cell RNAseq analysis
# %conda install -c conda-forge scanpy python-igraph leidenalg
#Import scanpy
import scanpy as sc
```

In case there are version conflicts during this installation or while importing scanpy, we found *%conda update –all* to be an useful command that fixed the version conflict previously. Reading in the 10X h5ad object associated with the same watermelon data

```
adata = sc.read_h5ad(f"{p}/inputFiles/watermelonScRnaSeqData.h5ad")
adata
```

```
AnnData object with n_obs × n_vars = 1093 × 27264
```

Identifying the cells that were below the barcode UMI threshold and were filtered out during the cellranger step

Read in the output files to identify cells as being singlets, multiplets or being removed for low barcode UMI threshold

First reading in the cellID barcode-UMI sheet generated earlier with additional filter using scRNAseq data

```
lowUmiCells = cellidBarcodeUMI[~
cellidBarcodeUMI['cellID'].isin(otherSampleNameSinglets[0]) |

cellidBarcodeUMI['cellID'].isin(sampleNameMultiplets[0]))]
cellidBarcodeUMI =
pd.read_csv(f'{p}/outputFiles/watermelonBarcodeUmiWith10X.csv')
```

Annotating the cells in adata with these labels

Reading in all the singlets and multiplets identified in the two samples. There might not always be multiplets - check the stats file to see if there are any. In this example, there are no multiplets in otherSampleName

```
#Annotating the adata with these labels using the lists created
adata.obs.loc[adata.obs.index.isin(otherSampleNameSinglets[0]), 'singletStatus']
= 'singlet'
adata.obs.loc[adata.obs.index.isin(sampleNameMultiplets[0]),
'singletStatus'] = 'singlet'
adata.obs.loc[adata.obs.index.isin(lowUmiCells['cellID']), 'singletStatus']
= 'lowUmi'
otherSampleNameSinglets = pd.read_csv(f"
{p}/outputFiles/watermelon_otherSampleName_singlets_all.txt", header = None)
sampleNameMultiplets = pd.read_csv(f"
{p}/outputFiles/watermelon_sampleName_multiplets.txt", header = None)
```

Note that in this vignette we are not doing any actual QC - but in actual analysis

```
AnnData object with n_obs × n_vars = 1093 × 27264
```

Identifying the cells that were below the barcode UMI threshold and were filtered out using the cellranger pipeline

Read in the output files to identify cells as being singlets, multiplets or being removed for low barcode UMI threshold

First reading in the cellID barcode UMI sheet generated earlier with additional filter using scRNAseq data.

```
lowUmiCells = cellidBarcodeUMI[~
(cellidBarcodeUMI['cellID'].isin(otherSampleNameSinglets[0]) |

cellidBarcodeUMI['cellID'].isin(sampleNameMultiplets[0]))]
cellidBarcodeUMI =
pd.read_csv(f'{p}/outputFiles/watermelonBarcodeUmiWith10X.csv')
```

Annotating the cells in adata with these labels

Reading in all the singlets and multiplets identified in the two samples. There might not always be multiplets - check the stats file to see if there are any. In this example, there are no multiplets in otherSampleName.

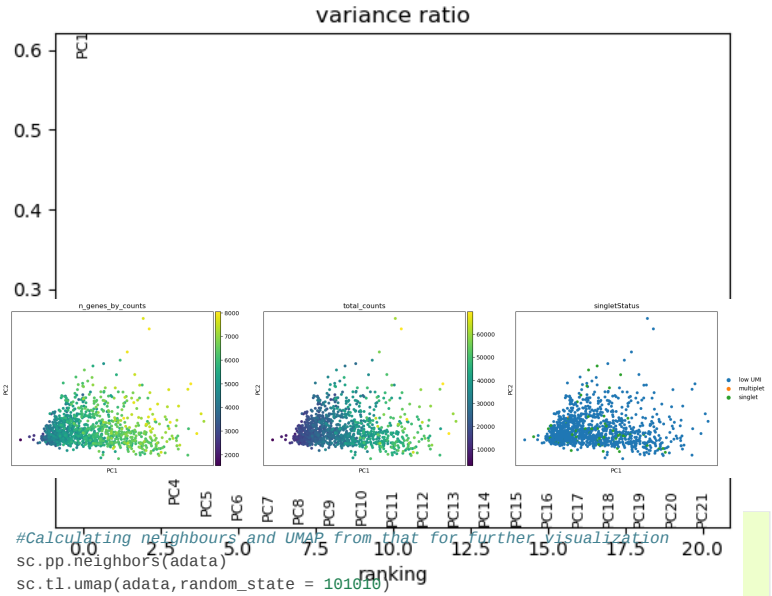
```
#Annotating the adata with these labels using the lists created
adata.obs.loc[adata.obs.index.isin(sampleNameSinglets[0]), 'singletStatus']
= 'singlet'
adata.obs.loc[adata.obs.index.isin(otherSampleNameSinglets[0]),
'singletStatus'] = 'singlet'
adata.obs.loc[adata.obs.index.isin(sampleNameMultiplets[0]),
'singletStatus'] = 'multiplet'
adata.obs.loc[adata.obs.index.isin(lowUmiCells['cellID']), 'singletStatus']
= 'lowUMI'
sampleNameSinglets = pd.read_csv(f"
{p}/outputFiles/watermelon_sampleName_singlets_all.txt", header = None)
otherSampleNameSinglets = pd.read_csv(f"
{p}/outputFiles/watermelon_otherSampleName_singlets_all.txt", header = None)
sampleNameMultiplets = pd.read_csv(f"
{p}/outputFiles/watermelon_sampleName_multiplets.txt", header = None)
```

Note that in this vignette we are not doing any actual QC - but in actual analysis, it would need to be done.

```
sc.pp.calculate_qc_metrics(adata, inplace=True)
```

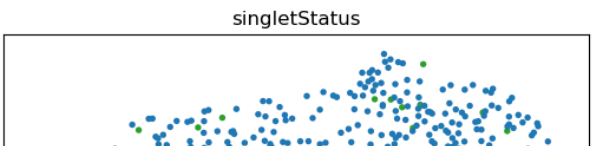
Calculating PCA and UMAP for visualization

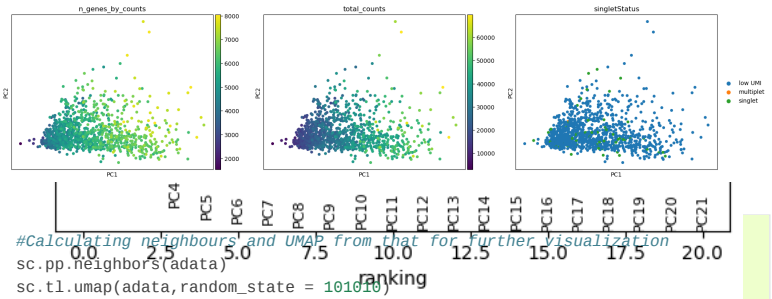
```
#Calculating PCA for the data and plotting variance ratio
sc.tl.pca(adata)
sc.pl.pca_variance_ratio(adata, n_pcs=20)
```




```
#Calculating neighbours and UMAP from that for further visualization
sc.pp.neighbors(adata)
sc.tl.umap(adata,random_state = 101010)
```

```
sc.pl.pca(
    adata,
    color = ['n_genes_by_counts', 'total_counts', 'singletStatus'],
    size = 100,
)
sc.pl.umap(
    adata,
    color=['singletStatus'],
    size=60
)
```

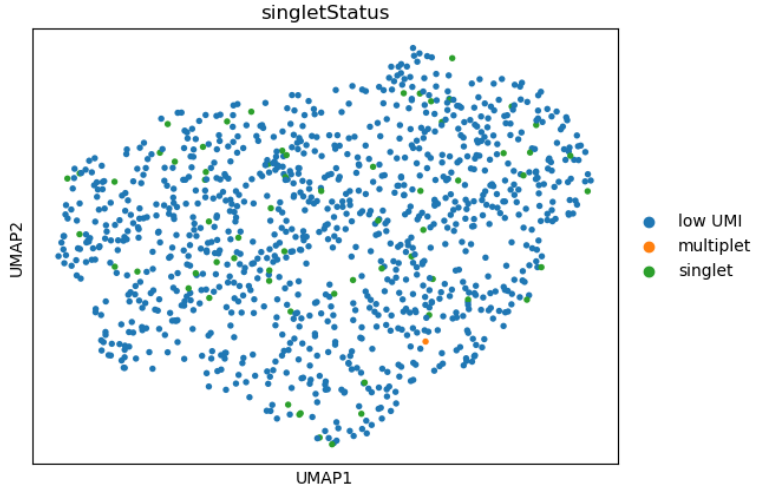





```
#Calculating neighbours and UMAP from that for further visualization
sc.pp.neighbors(adata)
sc.tl.umap(adata,random_state = 101010)
```

```
sc.pl.pca(
    adata,
    color = ['n_genes_by_counts', 'total_counts', 'singletStatus'],
    size = 100,
)
```

```
sc.pl.umap(
    adata,
    color=['singletStatus'],
    size=60
)
```



Saving the final adata

```
adata.write(f"{p}/outputFiles/watermelonScRNA.h5ad")
```

Vignette to use singletCode package

The input needed to run singletCode is a .csv file that contains the information about cell ID (added while sequencing), lineage barcode, and sample name. Each row should be repeated n times where n is the number of UMIs associated with that barcode and cell ID combination. You can download a sample input sheet here. It is a subset of data from Jiang Et al and details about it are described in the singletCode paper in detail. The folder also contains expected output files in the test folder within the outputFiles folder. This vignette can be downloaded as a jupyter notebook from the singletCode Tools repo.

Read in input sheet

```
#Read in the sheet
import pandas as pd
path = "path/to/downloaded/and/unzipped/folder"
pathToInputSheet = f"{path}/inputFiles/JiangEtAlSubset_InputSheet.csv"
df = pd.read_csv(pathToInputSheet)
```

Install singletCode package

```
!pip3 install singletCode
```

Check formatting of input sheet

```
check_sample_sheet(df)
```

Import necessary functions from it

```
from singletCode import check_sample_sheet, get_singlets
The sample sheet provided can be used as input to get_singlets to get a list
of singlets identified.
```

The sample sheet provided can be used as input to get_singlets to get a list of singlets identified.

Identify singlets from input sheet

Read in input sheet

here. It is a subset of data from Jiang Et al and details about it are described in the singletCode paper in detail. The folder also contains expected output files in the test folder within the outputFiles folder. This vignette can be downloaded as a jupyter notebook from the singletCode Tools repo.

```
#Reading within the sheet
import pandas as pd
path = "path/to/downloaded/and/unzipped/folder"
pathToInputSheet = f"{path}/inputFiles/JiangEtAlSubset_InputSheet.csv"
df = pd.read_csv(pathToInputSheet)
```

Install singletCode package

```
!pip3 install singletCode
```

Check formatting of input sheet

```
check_sample_sheet(df)
```

Import necessary functions from it

```
from singletCode import check_sample_sheet, get_singlets
The sample sheet provided can be used as input to get_singlets to get a list
of singlets identified.
```

The sample sheet provided can be used as input to get_singlets to get a list of singlets identified.

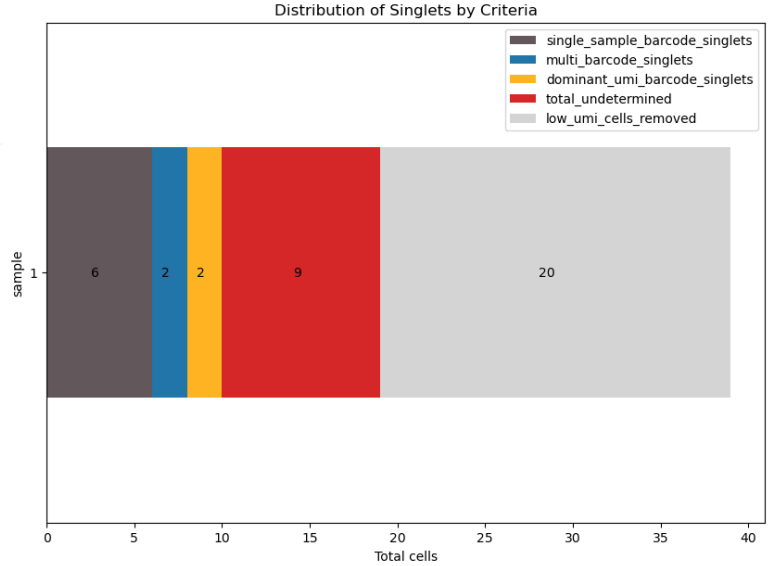
Identify singlets from input sheet

```
outputPath = "path/to/output/folder"
cellLabelList, stats = get_singlets(df, dataset_name= "JiangEtAlSubset",
save_all_singlet_categories = True, output_path=outputPath)
```

```
INFO: Raw data counts:
sample
1    1306
Name: count, dtype: int64
Total cells for sample 1: 39
INFO: Using ratio based filtering.
Current Sample Adjusted UMI cutoff: 2
```

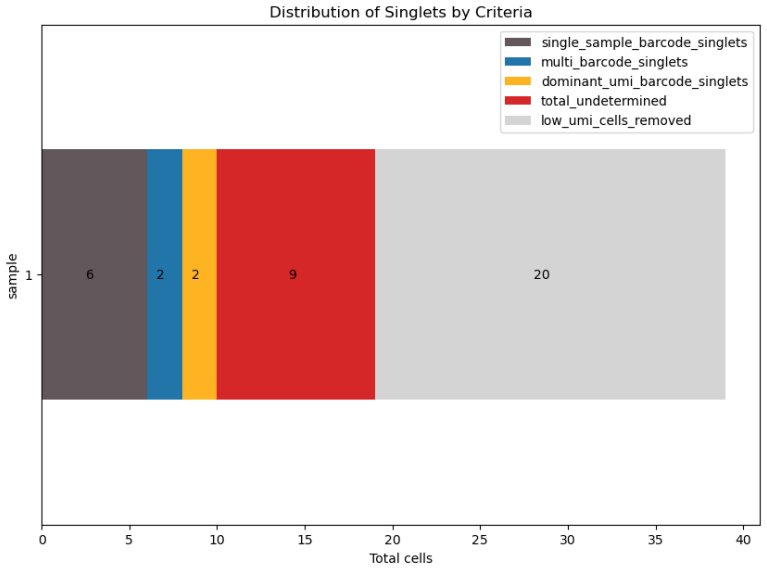
```
100%|██████████| 122/122 [00:00<00:00, 11475.27it/s]
```

```
All singlets identified with multiple barcodes are unique? True
Total Singlets: 10
Total Multiplets: 9
```




```
ax.set_xlabel('Total cells')
ax.set_title('Distribution of Singlets by Criteria')
plt.show()
```

The above plot shows that the data we had contained different kind of singlets: 6 single-barcode cells, 2 cells which had more than one barcode but with same combination being present in more than one cell, 2 cells that had one dominant barcode. The data also contained 9 cells which singletCode could not determine




```
  ax.set_xlabel('Total cells')
  ax.set_ylabel('sample')
  plt.show()
```

The above plot shows that the data we had contained different kind of singlets: 6 single-barcode cells, 2 cells which had more than one barcode but with same combination being present in more than one cell, 2 cells that had one dominant barcode. The data also contained 9 cells which singletCode could not determine as being truly singlets and 20 cells whose barcode UMI counts were below the set threshold.

Understanding the output files

To understand some of the files in the output, we can look at cell IDs and their data in the original input sheet

For the dominant_umi_singlets, there are two cell IDs. One of them is TGTAAGCGTCTCGCGA. If we look at that entry in the input sheet and count the number of UMI associated with each barcode, we see that one barcode has 99 UMI counts while the second highest UMI count is 7. So, the cell most likely has only one barcode associated with it and hence, a singlet.

```
import pandas as pd
df[df['cellID'] == 'TGTAAGCGTCTCGCGA'].groupby(['cellID', 'barcode',
'sample']).size().reset_index(name='count').sort_values('count',
ascending=False).reset_index(drop=True)
```

	cellID	barcode	sample	count
0	TGTAAGCGTCTCGCGA	ATTGTTGTTGCAGATGCAGTTGATGCTGATGAAGTTGTACAAGGTC...	1	99
1	TGTAAGCGTCTCGCGA	ATTCGACTTGATCTTCTAGAACATGGTGAACTAGCAGGTGCTGATC...	1	7
2	TGTAAGCGTCTCGCGA	ATACTAGCTCAAGCAGTACTACTACTTCGTCTTCATGCAGAACAAC...	1	6
3	TGTAAGCGTCTCGCGA	ATAGATGCACTTGGTGGTCGAGTTCTAGTTGTAGCTGATCGTCCAG...	1	6
4	TGTAAGCGTCTCGCGA	ATTCGACCAGAACCACATGCAGTTCAACGTGTTCGAGGTGTAGATG...	1	6
...	...	...	...	...
82	TGTAAGCGTCTCGCGA	ATAGGAGTAGTTGGTGATGGTCTACCAGAAGGTGAAGGTGGAGAAG...	1	1
83	TGTAAGCGTCTCGCGA	ATAGTACATGGTGGACCTGGACTTCGAGATGGAGCTCTTGTTCCTG...	1	1
84	TGTAAGCGTCTCGCGA	ATAGTACATGGTGGACCTGGACTTCGAGATGGAGCTCTTGTTCCTG...	1	1
85	TGTAAGCGTCTCGCGA	ATAGGAGTAGTTGGTGATGGTCTACCAGAAGGTGAAGGTGGAGAAG...	1	1
86	TGTAAGCGTCTCGCGA	GGTGCTGAACTTCTTGTTGTACTTCTAGTTGATGTTGGACGTCATC...	1	1

87 rows × 4 columns

Next, we can look at multi-barcode singlets. There are two cell IDs: AGGCTGCTCTTTCCGG and GAGGGATGTAACATCC. If we look at the barcodes with greater than 2 UMI counts, we see that they have the same combination. The only way this can occur is if a cell receives multiple barcode initially and then divides.

```
(df[df['cellID'] == 'AGGCTGCTCTTTCCGG']
  .groupby(['cellID', 'barcode', 'sample'])
  .size()
  .reset_index(name='count')
  .sort_values('count', ascending=False)
  .query('count >= 2')
  .reset_index(drop=True)
)
```

scRNAseq data

Further single-cell RNAseq analysis with both scRNAseq data and singlet information from singletCode output

Install and import scanpy for further single-cell RNAseq analysis

```
!pip scanpy[leiden]
```

```
#Import scanpy
import scanpy as sc

#Reading the scRNAseq input data in h5ad format
```

Reading the scRNAseq input data in h5ad format

	cellID	barcode	sample	count
3	TGTAAGCGTCTCGCGA	ATAGATGCACTTGGTGGTCGAGTTCTAGTTGTAGCTGATCGTCCAG...	1	6
4	TGTAAGCGTCTCGCGA	ATTCGACCAGAACCACATGCAGTTCAACGTGTTCGAGGTGTAGATG...	1	6
...	...	...	...	...
82	TGTAAGCGTCTCGCGA	ATAGGAGTAGTTGGTGATGGTCTACCAGAAGGTGAAGGTGGAGAAG...	1	13
83	TGTAAGCGTCTCGCGA	ATTCAACCGCCAACTGAACTTCTATCGACTACCTCTAGACACATC...	1	21
84	TGTAAGCGTCTCGCGA	ATAGTACATGGTGGACCTGGACTTCGAGATGGAGCTCTTGTTCCTG...	1	1
85	TGTAAGCGTCTCGCGA	ATAGGAGTAGTTGGTGATGGTCTACCAGAAGGTGAAGGTGGAGAAG...	1	1
86	TGTAAGCGTCTCGCGA	GGTGCTCAACTTCTTGTTGTACTTCTAGTTGATGTTGGACGTCATC...	1	1

87 rows × 4 columns

Next, we can look at multi-barcode singlets. There are two cell IDs: AGGCTGCTCTTTCCGG and GAGGGATGTAACATCC. If we look at the barcodes with greater than 2 UMI counts, we see that they have the same combination. The only way this can occur is if a cell receives multiple barcode initially and then divides.

```
(df[df['cellID'] == 'AGGCTGCTCTTTCCGG']
 .groupby(['cellID', 'barcode', 'sample'])
 .size()
 .reset_index(name='count')
 .sort_values('count', ascending=False)
 .query('count >= 2')
 .reset_index(drop=True)
)
```

Making copies of singletCode input/output to use them along with scRNAseq data. The -1 is added to cell IDs to match the cell IDs seen in 10x format data. **NOTE**: It may not be needed for your actual data.

```
singleCellDf = df.copy()
singleCellDf['cellID'] = singleCellDf['cellID'] + "-1"
singleCellDf = singleCellDf.drop_duplicates(subset = 'cellID')
cellLabelListSingleCell = cellLabelList.copy()
cellLabelListSingleCell['cellID'] = cellLabelListSingleCell['cellID'] + "-1"
cellLabelListSingleCell =
cellLabelListSingleCell.drop_duplicates(subset='cellID').reset_index(drop =
True)
```

scRNAseq data

Further single-cell RNAseq analysis with both scRNAseq data and singlet information from singletCode output

Install and import scanpy for further single-cell RNAseq analysis

```
!pip scanpy[leiden]
```

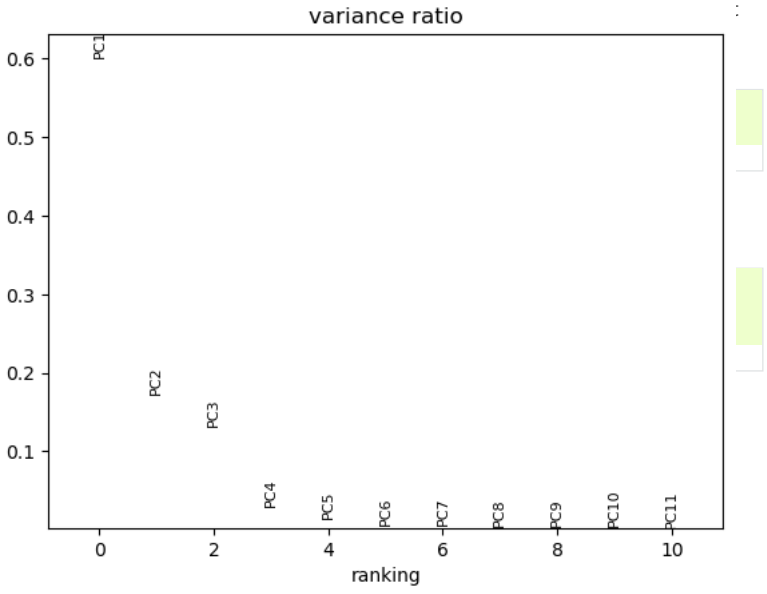
```
import scanpy as sc
```

Reading the scRNAseq input data in h5ad format

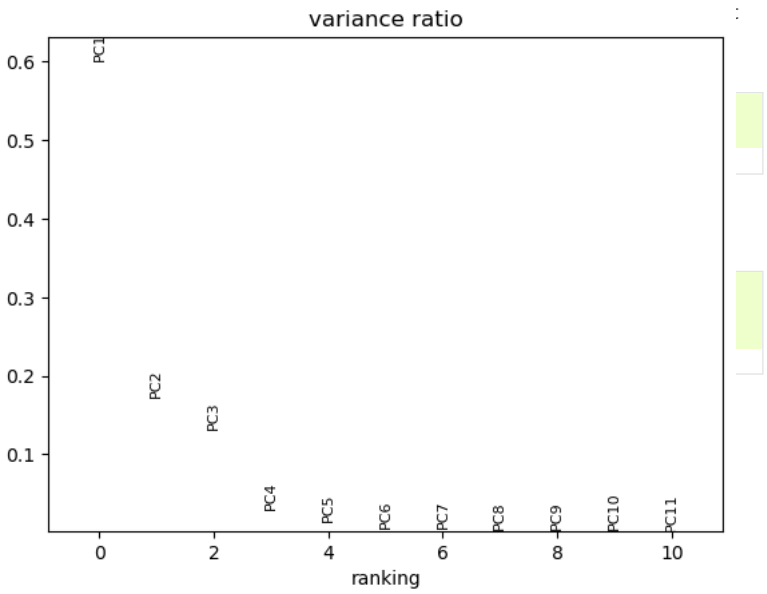
```
#Reading the scRNAseq data in h5ad format
adata = sc.read_h5ad(f"{path}/inputFiles/JiangEtAlSubset_scRNAseqData.h5ad")
adata
```

```
AnnData object with n_obs × n_vars = 39 × 36601
    var: 'gene_ids', 'feature_types'
```

Calculating total counts and genes identified per cell.



Identifying cells that were thresholded by singletCode as low UMI by identifying cells that were in the original list provided to singletCode but not labeled as either singlet or undetermined. Then creating a list of annotations of



Identifying cells that were thresholded by singletCode as low UMI by identifying cells that were in the original list provided to singletCode but not labeled as either singlet or undetermined. Then creating a list of annotations of singletStatus(singlet, multiplet, low UMI) for all cells

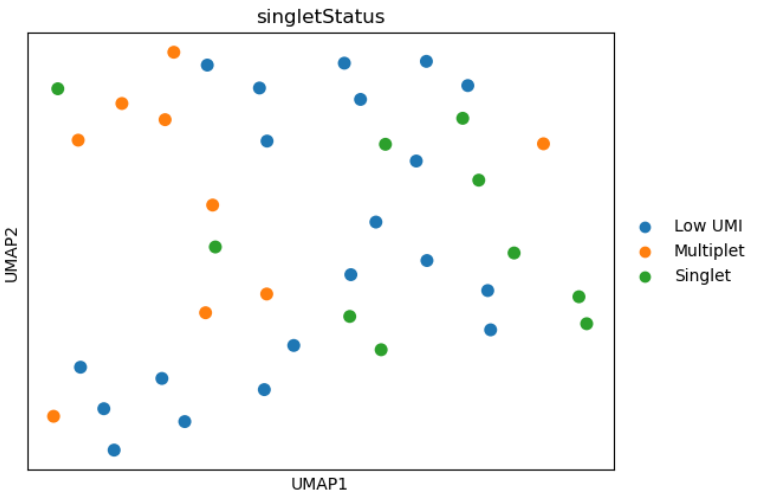
```
umiCutoff = pd.DataFrame(

singleCellDf.loc[~singleCellDf['cellID'].isin(cellLabelListSingleCell['cellID']
'cellID']
    .drop_duplicates()
    .reset_index(drop=True),
    columns=['cellID']
)
umiCutoff['label'] = "Low UMI"
```

```
cellIDLabels = cellLabelListSingleCell.drop(columns = ['barcode', 'sample',
'nUMI']).drop_duplicates().reset_index(drop = True)
```

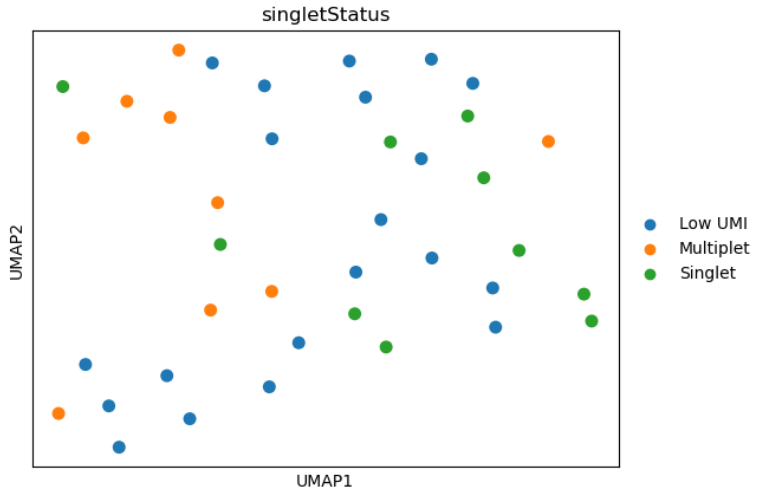
```
#Creating a list of cell IDs with annotation of whether singlet, multiplet
or low UMI.
labelID = pd.concat([umiCutoff, cellIDLabels]).reset_index(drop=True)
labelID = labelID.set_index(labelID['cellID']).drop(columns = ['cellID'])
#Adding the labels to cells in the adata to visualise it
adata.obs["singletStatus"] = labelID
```

Visualising the cells in PCA space

```
sc.pl.umap(
    adata,
sc.pl.pca(
    color=['singletStatus'],
    adata,
    # Setting a smaller point size to get prevent overlap
    color = ['n_genes_by_counts', 'total_counts', 'singletStatus'],
    size=250,
    size = 250
)
)
```



```
sc.pl.umap(
    adata,
sc.pl.pca(
    color=['singletStatus'],
    adata,
    # Setting a smaller point size to get prevent overlap
    color = ['n_genes_by_counts', 'total_counts', 'singletStatus'],
    size=250,
    size = 250
)
)
```



Saving the AnnData

```
adata.write(f"{outputPath}/JiangEtAlSubset.h5ad")
```

questions? contact us!

Do you have a doublet detection method that you would like use with singletCode? Do you have a question about how to use singletCode? Do you have a suggestion for how to improve singletCode? Please contact us!

For more information or questions, please contact Yogesh Goyal at yogesh.goyal@northwestern.edu

repositories

GitHub repository for analyses performed in our paper here.

GitHub repository for singletCode tools (command line interface and python package) here.