## Contents

1 Introduction 3
2 How to use OnClass 5
3 Installing OnClass 9
4 Dataset and pretrained models 11
5 Tutorial 15
6 References 19
Bibliography 21
OnClass is a python package for single-cell cell type annotation. It uses the Cell Ontology to capture the cell type similarity. These similarities enable OnClass to annotate cell types that are never seen in the training data.
OnClass is a python package for single-cell cell type annotation. It uses the Cell Ontology to capture the cell type similarity. These similarities enable OnClass to annotate cell types that are never seen in the training data.

OnClass package is still under development. A preprint of OnClass paper is on bioRxiv All datasets used in OnClass can be found in figshare. Currently, OnClass supports

1) annotate cell type
2) integrating different single-cell datasets based on the Cell Ontology
3) marker genes identification

OnClass is a joint work by Altman lab at stanford and czbiohub.

For questions about the software, please contact Sheng Wang at swang91@stanford.edu.

Our web server can be found at: https://onclass.readthedocs.io/.
The key idea of OnClass is using the Cell ontology to help cell type prediction. OnClass is the first tool that can classify cells into a new cell type that doesn’t exist in the training data. To achieve this, it has three steps:

### 2.1 1) Embedding Cell ontology

The first step of OnClass is to embed cell types into the low-dimensional space based on the Cell Ontology. OnClass can take different formats of cell type similarities, including hierarchical structure, directed acyclic graph, weighted network, unweighted network or just a few lines of cell type similarity. An example can be found here.

In our paper, we use the hierarchical ontology structure from the Cell Ontology. However, OnClass is very flexible and can take any forms of prior cell type similarity (i.e., label similarity) can be used. It could be a weighted network, where nodes are cell types and edge weights are cell type similarity (edge weights are not required to be normalized between 0 and 1). It could be an unweighted network where all edge weights are set to 1, which is used in our analysis. It could be just a few lines of cell type similarity, where each line is a tab-spitted three Column file in the form of “CL:000001 CL:000002 0.8”, representing cell type 1, cell type 2, and their similarity. By default, all edges are undirected since they are similarity. But if you have directed similarities, please email us and we are happy to modify the code to support it.

We provide a precomputed cell ontology embeddings based on the Cell Ontology in figshare. Please check the Tutorial section Embedding Cell Ontology for how to read and embed the Cell Ontology.

### 2.2 2) Read the gene expression data

The second step of OnClass is to read the gene expression data with training labels.

The gene expression data is used as training data. It includes a cell by gene matrix and a label for each cell. The label for each cell should be a cell ontology ID or cell ontology terms. If your training labels are not mapped to cell ontology ID, please use our natural language processing tool to map them to existing cell ontology terms. Our tool use a char-level LSTM siamese network and achieve 93.7% accuracy in mapping synonym of cell ontology terms.
and can deal with composition, misspelling, and abbreviations. For more information about this natural language processing tool, please see section Char-level LSTM for cell type term mapping.

OnClass is a highly flexible tool that can support various formats of gene expression data inputs, including scipy sparse matrix, numpy 2D array, tsv format of matrix, tsv format of “gene_name cell_name count”, scanpy AnnData, and sparse matrix format used by 10X Genomics. Some of these input implementations are adopted from the Scanorama project.

2.3 3) Predict cell types for new cells

The last step of OnClass is to predict cell types according to cell type embeddings and gene expression.

OnClass can predict cell types through any of the following four approaches:

a) Use the pretrained marker genes. OnClass found 20 marker genes for each cell type in the Cell Ontology. Marker genes are precomputed based on all FACS cells from Tabula Muris Senis.

b) Use the predicted score of any existing models and propagate to all cell types in the Cell Ontology.

c) Use the pretrained Bilinear Neural Network model. This model is trained on all FACS cells from Tabula Muris Senis and all cell type terms from the Cell Ontology.

d) Train from new data. Train a new model on a new gene expression data and predicted on the cell types for a set of new cells.

The time and memory complexity of a) and b) are very small (e.g., less than 1 minute on any personal laptop). The time and memory complexity of c) is moderate (less than 1 minute for 50K cells on GPU and less than 10 minutes for 50K on CPU.) The time and memory complexity of d) is large (about 1 hour for 50k cells on GPU and 4 hour for 50k cells on CPU). However, the expected performance is: d) > c) > b) > a). So please choose one of them according to your application.

2.4 Flowchart

Here is the flowchart of OnClass, describing these three steps:
2.4. Flowchart

Step 1: Embed the Cell Ontology

Step 2: Divide low-dimensional space of cell types

Step 3: Project single cells

Cell type annotation

Marker genes identification

1) CD4-
2) CD25-
...

Data integration

Unannotated cells
Annotated cells
Installing OnClass

OnClass can be substantially accelerated by using GPU (tensorflow). However, there is only required when you want to train your own model.

3.1 Install from Conda environment file (Recommended)

This is the recommended way to install OnClass as it can install the most updated version of OnClass

1) Only use CPU

```
git clone https://github.com/wangshenguiuc/OnClass.git
conda env create -f environment.yml --name env_name
conda activate env_name
```

2) Use GPU

```
git clone https://github.com/wangshenguiuc/OnClass.git
conda env create -f environment_gpu.yml --name env_name
conda activate env_name
```

3.2 PyPI

OnClass is available through the Python Package Index and thus can be installed using pip. Please use Python3.6. To install OnClass using pip, run:

1) Only use CPU

```
pip install OnClass
pip install tensorflow==1.14
```

2) Use GPU
pip install OnClass
pip install tensorflow-gpu==1.14

3.3 Development Version

The lastest version of OnClass is on GitHub

```bash
git clone https://github.com/wangshenguiuc/OnClass.git
```
Dataset and pretrained models

The datasets used by OnClass paper can be found at figshare.
We provided the following datasets.
Currently, we include:

### 4.1 1) 26-datasets

26-datasets obtained from Scanorama project.
4.2 2) cell_ontology

The Cell Ontology file used in our analysis, which is downloaded from OBO Foundry.

4.3 3) figures

Reproduced figures in the paper using OnClass.

4.4 4) marker_genes

Marker genes for 2331 cell types identified by OnClass and curated by experts.

4.5 5) pretrained model

Cell type embedding and pretrain prediction model.

4.6 6) raw data

Gene expression data for FACS cells and droplet cells used in our analysis.

For questions about the datasets, please contact Sheng Wang at swang91@stanford.edu.
Here, we provide an introduction of how to use OnClass. We are going to train a model on all FACS cells from Tabula Muris Senis (TMS) and then predict the cell types of all droplet cells in TMS. By using this example, you can see how OnClass embeds the Cell Ontology, reads gene expression data, uses the pretrained model, and makes the prediction on new cells.

5.1 Cell type annotation (Use the pretrained model)

The script AnnotateTMS.py for transferring cell type annotation is at our GitHub. We provide a pretrained model in our data repo, which was trained on all FACS cells in tabula muris senis.

Import OnClass and other libs as:

```python
from OnClass.utils import *
from OnClass.OnClassModel import OnClassModel
from OnClass.other_datasets_utils import my_assemble, data_names_all, load_names
```

Embed the cell ontology:

```python
OnClassModel = OnClassModel()
tp2emb, tp2i, i2tp = OnClassModel.EmbedCellTypes(dim=500, cell_type_network_file='../..\../OnClass_data/cell_ontology/cl.ontology', use_pretrain='../../onClass_data/pretrain/tp2emb_500')
```

Here, we used the pretrained cell type embedding file tp2emb_500, which is the 500-dimensional vectors of cell types from cl.ontology. All files are provided on figshare. Please download them and put them in the corresponding directory. At this step, we are not using gene expression or cell type annotations when embedding the Cell Ontology. If you want to generate your own embeddings, please set use_pretrain = None.

Read TMS h5ad gene expression data. This file is also on figshare which is the data used in our paper. cell_ontology_class_reannotated is the attribute of labels in the h5ad file:
Train the model

```python
OnClassModel.train(train_X, train_Y, tp2emb, train_genes, nhidden=[500], log_transform=True, use_pretrain='../../../OnClass_data/pretrain/BilinearNN_50019')
```

Here, we use the pretrained model BilinearNN_500 which can be downloaded from figshare. If you want to train your own model, please set use_pretrain = None.

Predict the labels of cells in droplet cells.

```python
test_data_file = '../../../OnClass_data/raw_data/tabula-muris-senis-droplet.h5ad'
test_X, test_genes, test_AnnData = read_data(feature_file=test_data_file, tp2i = tp2i, return_AnnData=True)
test_label = OnClassModel.predict(test_X, test_genes, log_transform=True, correct_batch=False)
```

```python
x = write_anndata_data(test_label, test_AnnData, i2tp, name_mapping_file='../../../OnClass_data/cell_ontology/cl.obo')  # output_file is optional
print(x.obs['OnClass_annotation_ontology_ID'])
print(x.obs['OnClass_annotation_ontology_name'])
```

### 5.2 Cell type annotation (Train from scratch)

The script TrainOnClassFromScratch.py showing an example of training from scratch is at our GitHub. To run this script, you need a training data (tabula-muris-senis-facs_cell_ontology.h5ad in this script) and a test data (tabula-muris-senis-droplet.h5ad in this script). The two datasets used in our script are in figshare.

Import OnClass and other libs as:

```python
from OnClass.utils import *
from OnClass.OnClassModel import OnClassModel
from OnClass.other_datasets_utils import my_assemble, data_names_all, load_names
```

Embed the cell ontology:

```python
OnClassModel = OnClassModel()
tp2emb, tp2i, i2tp = OnClassModel.EmbedCellTypes(dim=500, cell_type_network_file='../../../OnClass_data/cell_ontology/cl.ontology', use_pretrain='../../../OnClass_data/pretrain/tp2emb_500')
```

Here, we used the pretrained cell type embedding file tp2emb_500, which is the 500-dimensional vectors of cell types from cl.ontology. All files are provided on figshare. Please download them and put them in the corresponding directory. At this step, we are not using gene expression or cell type annotations when embedding the Cell Ontology. If you want to generate your own embeddings, please set use_pretrain = None.

Read the training data and the test dataset. cell_ontology_class_reannotated is the attribute of labels in the h5ad file:
5.3 Data Integration (integrate 26-datasets using OnClass)

A example script DataIntegration.py for transferring cell type annotation is at our GitHub

Load the pre-computed scoring matrix (see the above section for detail):

```python
test_Y_pred = np.load(OUTPUT_DIR + '26_datasets_predicted_score_matrix.npy')
datasets, genes_list, n_cells = load_names(data_names_all, verbose=False, log1p=True, DATA_DIR=DATA_DIR)
datasets, genes = merge_datasets(datasets, genes_list)
```

Integration based on our method:

```python
pca = PCA(n_components=50)
test_Y_pred_red = pca.fit_transform(test_Y_pred[:, :nseen])
```

Please check DataIntegration.py for how to obtain the UMAP plots.

5.4 Marker genes identification

A example script FindMarkerGenes.py for transferring cell type annotation is at our GitHub

To find maker genes, we first train on all FACS cells and then generate the scoring matrix for all FACS cells:

```python
train_X, train_Y_str, genes_list = read_data(filename=data_file, return_genes=True)
tms_genes_list = [x.upper() for x in list(genes_list.values())[0]]
```

(continues on next page)
train_Y = MapLabel2CL(train_Y_str, l2i)

## train and predict
OnClass_obj = OnClassPred()
OnClass_obj.train(train_X, train_Y, Y_emb, max_iter=20, nhidden=[100])
test_Y_pred = OnClass_obj.predict(test_X)

np.save(OUTPUT_DIR + 'FACS-predicted_score_matrix.npy', test_Y_pred)

Differential expression analysis:

```python
ncell = np.shape(test_Y_pred)[0]
co2name, name2co = get_ontology_name()
tp2genes = read_type2genes(g2i)
thres = np.array(range(1,1000))
    topk = 50
in_tms_ranks = []
not_tms_ranks = []
    n_in_tms = 0
for tp in tp2genes:
    ci = l2i[tp]
    k_bot_cells = np.argsort(test_Y_pred[:,ci])[:topk]
    k_top_cells = np.argsort(test_Y_pred[:,ci])[ncell-topk:]
    pv = scipy.stats.ttest_ind(train_X[k_top_cells,:], train_X[k_bot_cells,:], axis=0)
    top_mean = np.mean(train_X[k_top_cells,:],axis=0)
    bot_mean = np.mean(train_X[k_bot_cells,:],axis=0)
    for g in range(ngene):
        if top_mean[0,g] < bot_mean[0,g]:
            pv[g] = 1.
    pv_sort = list(np.argsort(pv))
```

Here, pv_sort is the rank list of marker genes for each cell type.

Please check FindMarkerGenes.py for how to marker genes. Please check Marker_genes_based_prediction_droplet.py and Marker_genes_based_prediction_26_datasets.py for how to use these marker genes to predict cell types for cells in TMS droplets and 26-datasets.
CHAPTER 6

References