Abhi Adduri - SC-Interview Task Data Analysis

The following notebook contains code that you can run to reproduce any of the results shown. For the most part, I will use this notebook as a markdown document to analyze the data and show plots easily. My repository also contains standalone scripts with usage guides, which is how I generated my results.

The final comparison across methods as reported below:

Method	Features	Micro AUROC	Micro AUPR	Macro AUPR	Micro F1	Macro F1
LR	Raw	0.892	0.684	0.603	0.643	0.571
RF	Raw	0.960	0.767	0.633	0.679	0.489
NN	Raw	0.979	0.852	0.713	0.756	0.644
LR	HVG	0.975	0.831	0.670	0.779	0.669
RF	HVG	0.971	0.810	0.669	0.722	0.571
NN	HVG	**0.989**	**0.920**	**0.783**	**0.837**	**0.700**
LR	scGPT	0.973	0.819	0.663	0.765	0.656
RF	scGPT	0.972	0.819	0.675	0.727	0.567
NN	scGPT	0.988	0.918	0.780	0.832	**0.700**

A general overview of this notebook:

1. First we will visualize and inspect the data.
2. We will fit baseline classifiers to the data, and a simple neural network, and compute the above metrics.
3. We observe a large gap in the metrics on the training and test sets, so we try alternate featurizations, and see large improvements using the highly-variable genes (HVG), and a foundation model (scGPT).
4. Ideas for future improvements.

All results were separately validated on a cluster with ten different random seeds. The results shown here are for a single train / validation / test split that stratifies the data such that every class is present in each split.

Setup

If you want to run these scripts on a standalone server and not a colab notebook, you may need to create a conda environment to make sure flash attention installs ok:

```
conda create -n scgpt python=3.9
conda activate scgpt
conda install pip
conda install -c nvidia cuda-toolkit
```

Imports and load data

```
# Install package requirements if you want to run in this notebook.
! pip install packaging wandb torch numpy scanpy matplotlib gdown tqdm --quiet
! pip install scgpt "flash-attn<1.0.5" --quiet # This will take a while...

# Clone my fork of the interview repository and chdir into it
! git clone https://github.com/abhinadduri/SC-interview.git
%cd SC-interview

# Make folders to store files for metric visualization down the line
! mkdir plots
! mkdir embeddings

# Download the requisite data
! gdown 1nsmQHdWek4YzIfKs9xUnLBHxKBWu8hUJ -O cells.npy

# Needed to generate embeddings with scGPT foundation model
! gdown https://drive.google.com/drive/folders/1oWh_-ZRdhtoGQ2Fw24HP41FgLoomVo-y -O scgpt --folder
! wget https://github.com/bowang-lab/scGPT/files/13243634/gene_info.csv -P scgpt

# Disable logging to W&B for this colab notebook
```

```
%env WANDB_MODE=offline
%env WANDB_SILENT=true
```

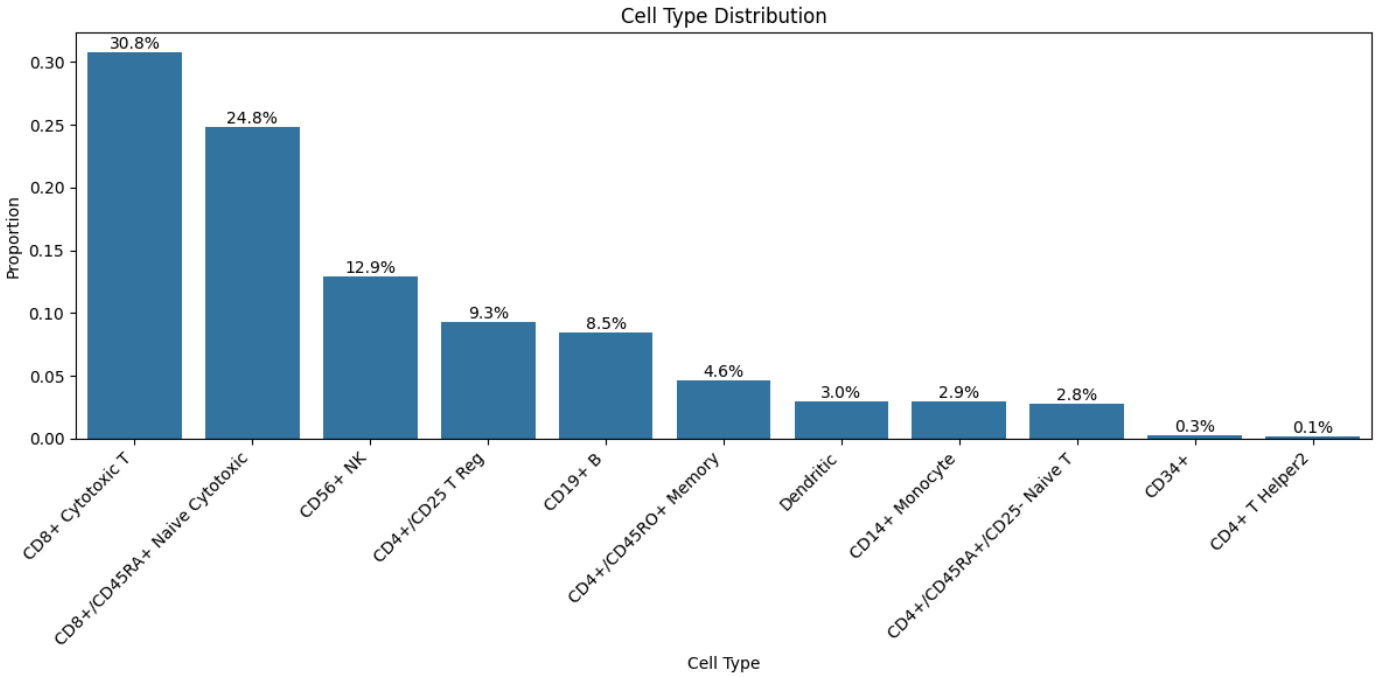
Show hidden output

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scanpy as sc
import seaborn as sns
from IPython.display import Image


# Create an AnnData object for our data
cells = np.load('cells.npy', allow_pickle=True).item()
adata = sc.AnnData(X=cells['UMI'].toarray(), obs={'cell_type': cells['classes']}, var={'gene_id': cells['gene_ids']})

# Basic preprocessing following scanpy suggestions
sc.pp.normalize_total(adata, target_sum=1e4)
sc.pp.log1p(adata)
```

Data Inspection and Visualization

We first want to visually inspect the data to see how separable the cell type clusters are, and also how imbalanced the classes are.

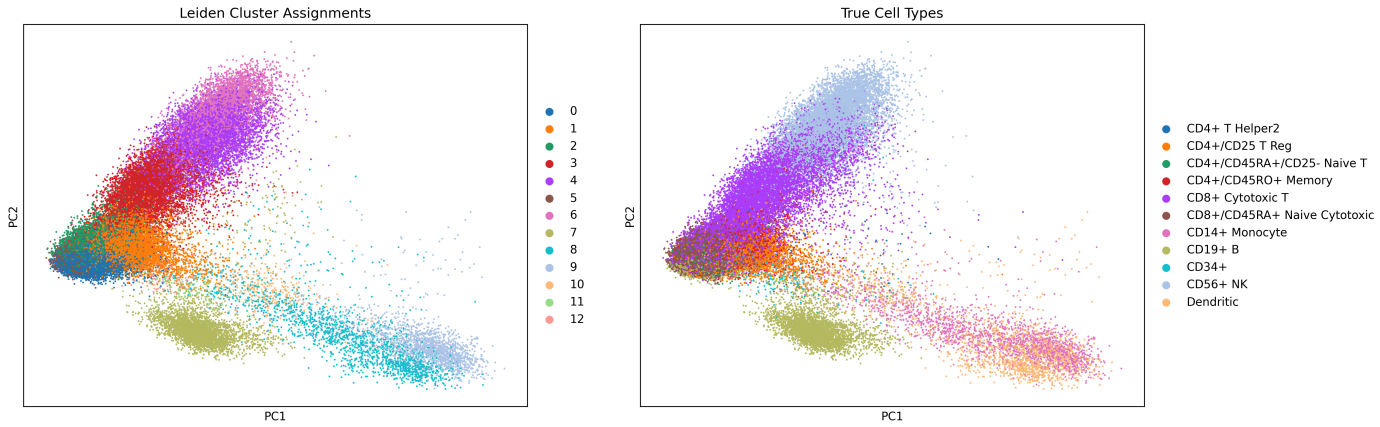
```
# Calculate the proportion of each cell type
cell_type_counts = adata.obs['cell_type'].value_counts()
cell_type_proportions = cell_type_counts / len(adata)

plt.figure(figsize=(12, 6))
sns.barplot(x=cell_type_proportions.index, y=cell_type_proportions.values)

plt.title('Cell Type Distribution')
plt.xlabel('Cell Type')
plt.ylabel('Proportion')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Add percentage labels on top of each bar
for i, v in enumerate(cell_type_proportions):
    plt.text(i, v, f'{v:.1%}', ha='center', va='bottom')

plt.show()
```



We see that the distribution of labels is very long tailed, and we can expect that classification of CD34+ and CD4+ T Helper2 might be difficult.

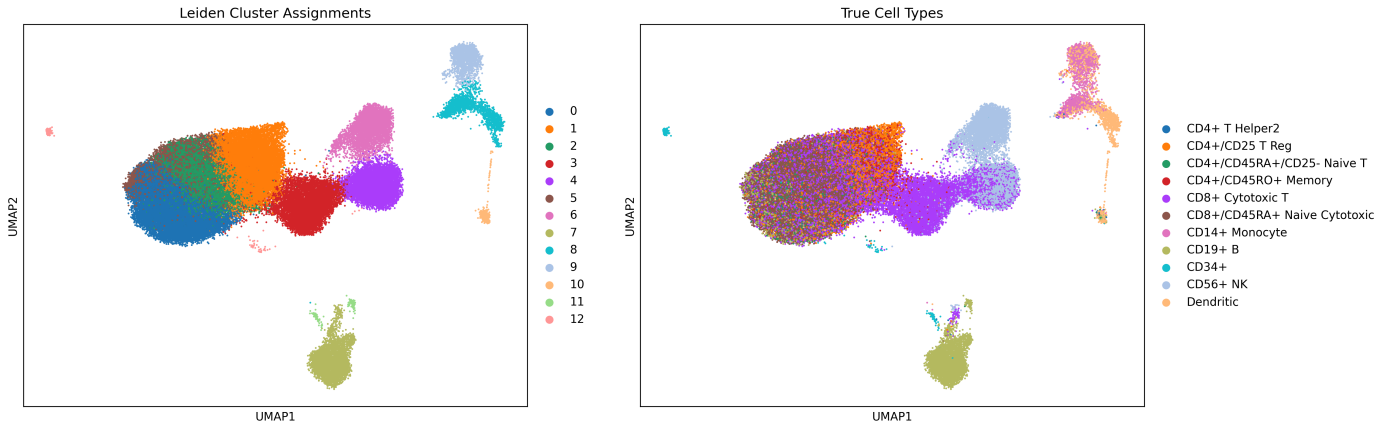
Let's also pre-compute several common transforms that scanpy offers to give us more insight into the data. Since most of the gene expression data seems sparse, we can start by computing the set of highly-variable genes.

```
# Compute common transforms using scanpy
sc.pp.highly_variable_genes(adata)
sc.pp.neighbors(adata, n_neighbors=10, n_pcs=40)

sc.tl.pca(adata, svd_solver='arpack')
sc.tl.umap(adata)
sc.tl.leiden(adata)

# Create confusion matrix
cluster_cell_type = pd.crosstab(adata.obs['leiden'], adata.obs['cell_type'])


with plt.rc_context({"figure.figsize": (8, 6), "figure.dpi": (300)}):
    sc.pl.pca(adata, color=['leiden', 'cell_type'], title=['Leiden Cluster Assignments', 'True Cell Types'], size=10)
```



From here we see there is quite a large overlap with CD56+ NK and CD8+ Cytotoxic T. There is also a blob of similar looking points consisting of CD8+ Cytotoxic T, CD8+/CD45RA + Naive Cytotoxic, CD19+ B, and CD4+/CD25T Reg, suggesting these classes may be difficult to discriminate.

```
with plt.rc_context({"figure.figsize": (8, 6), "figure.dpi": (300)}):
    sc.pl.umap(adata, color=['leiden', 'cell_type'], title=['Leiden Cluster Assignments', 'True Cell Types'], size=10)
```



Lastly let's take a look at a confusion matrix of the clusters vs the true cell type annotations.

```
confusion_matrix = pd.crosstab(adata.obs['leiden'], adata.obs['cell_type'], normalize='index')
plt.figure(figsize=(12, 10))
sns.heatmap(confusion_matrix, annot=True, cmap='YlGnBu', fmt='.2f')
plt.title('Leiden Clusters vs Known Cell Types')
plt.tight_layout()
plt.show()
```

This shows that a few cell types appear across a lot of clusters. Furthermore, cells like CD4+ T Helper2 are co-located with Dendritic cells in PCA space, and CD4+/CD45A+/CD25- Naive T are co-located with CD8+/CD45A+ naive Cytotoxic cells, which will likely make their discrimination difficult.

Baseline Models

Let's start by computing some baselines on the raw data. This will be expensive but can guide our analysis. We will train a logistic regression model, a random forest classifier, and a simple neural network on the normalized data. For these baselines we will use the balanced class weight option to address the cell type imbalance in our dataset.

```
! python train.py --method lr --verbose
```

```
Provided featurization has (65943, 16769) shape for X and (65943,) shape for y

Results for the test set:
              precision    recall  f1-score   support
CD14+ Monocyte  0.7772151898734178  0.850415512465374  0.8121693121693122      361.0
```

```
CD19+ B 0.7467411545623837      0.7059859154929577      0.72579185520362        1136.0
CD34+   0.9565217391304348      0.6875  0.8     32.0
CD4+ T Helper2  0.0     0.0     0.0     18.0
CD4+/CD25 T Reg 0.48839071257005606     0.5012325390304027      0.4947283049472831      1217.0
CD4+/CD45RA+/CD25- Naive T      0.30930930930930933     0.2853185595567867      0.29682997118155624     361.0
CD4+/CD45RO+ Memory     0.31647940074906367     0.2779605263157895      0.29597197898423816     608.0
CD56+ NK        0.8445984979780474      0.8450867052023121      0.8448425310603872      1730.0
CD8+ Cytotoxic T        0.6223404255319149      0.6526800873150619      0.6371492837693855      4123.0
CD8+/CD45RA+ Naive Cytotoxic    0.6499841621792841      0.6376631448104413      0.6437647058823529      3218.0
Dendritic       0.7701149425287356      0.6961038961038961      0.7312414733969985      385.0
accuracy        0.6434149670179695      0.6434149670179695      0.6434149670179695      0.6434149670179695
macro avg       0.5892450485829679      0.5581769896630019      0.5711354015086486      13189.0
weighted avg    0.6424400050822505      0.6434149670179695      0.6425195591513487      13189.0

Results for the test set:
        {'micro_auroc': 0.8924760681277552, 'micro_aupr': 0.6840640465973885, 'macro_aupr': 0.6025529531902228, 'micro_f1':
```

```
! python train.py --method rf --verbose
```

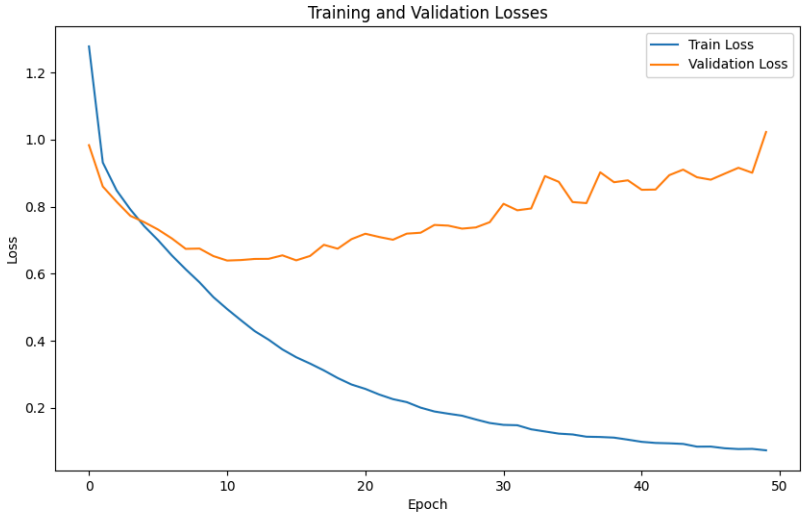
```
Provided featurization has (65943, 16769) shape for X and (65943,) shape for y

Results for the test set:
                precision       recall  f1-score        support
CD14+ Monocyte  0.7017241379310345      0.9667458432304038      0.8131868131868132      421.0
CD19+ B 0.9840213049267643      0.662780269058296       0.7920685959271169      1115.0
CD34+   1.0     0.3888888888888889      0.56    36.0
CD4+ T Helper2  0.0     0.0     0.0     18.0
CD4+/CD25 T Reg 0.7032640949554896      0.18824463860206514     0.29699248120300753     1259.0
CD4+/CD45RA+/CD25- Naive T      0.0     0.0     0.0     382.0
CD4+/CD45RO+ Memory     0.0     0.0     0.0     580.0
CD56+ NK        0.8544627629334849      0.8946428571428572      0.8740913056120966      1680.0
CD8+ Cytotoxic T        0.6398616515348032      0.7235394769005133      0.6791327291499368      4091.0
CD8+/CD45RA+ Naive Cytotoxic    0.5900061437640794      0.8994692475803934      0.7125896611427157      3203.0
Dendritic       0.8781512605042017      0.5173267326732673      0.6510903426791277      404.0
accuracy        0.6785957995299113      0.6785957995299113      0.6785957995299113      0.6785957995299113
macro avg       0.5774083051408961      0.4765125412796987      0.48901381171825586     13189.0
weighted avg    0.6529496821964097      0.6785957995299113      0.6377930273794957      13189.0

Results for the test set:
        {'micro_auroc': 0.9602348269934852, 'micro_aupr': 0.7665258064254556, 'macro_aupr': 0.6333428587835703, 'micro_f1':
```

Now that we have baselines, let's see if a neural network approach can improve our results.

```
! python train.py --method nn --verbose --loss-plot plots/raw_nn_loss_50.png
Image('plots/raw_nn_loss_50.png', width=720, height=432)
```

```
Provided featurization has (65943, 16709) shape for X and (65943,) shape for y
wandb: Tracking run with wandb version 0.17.5
wandb: W&B syncing is set to `offline` in this directory.
wandb: Run `wandb online` or set WANDB_MODE=online to enable cloud syncing.
Training: 100% 50/50 [01:06<00:00,  1.33s/epoch, Train Loss=0.0737, Val Loss=1.0228]
Results for the train set:
                precision       recall  f1-score        support
CD14+ Monocyte  0.9993654822335025      1.0     0.999682640431609       1575.0
CD19+ B 0.9993364299933643      0.9997787121044479      0.9995575221238938      4519.0
CD34+   0.9806451612903225      1.0     0.990228013029316       152.0
CD4+ T Helper2  1.0     0.44    0.6111111111111112      75.0
CD4+/CD25 T Reg 0.9987883683360258      0.9983851433185305      0.9985867151221482      4954.0
CD4+/CD45RA+/CD25- Naive T      0.9960079840319361      0.995345744680851       0.9956767542401064      1504.0
CD4+/CD45RO+ Memory     0.9927095990279465      0.9983706720977596      0.9955320877335501      2455.0
CD56+ NK        0.9991309385863267      0.9992756772417789      0.9992033026725574      6903.0
CD8+ Cytotoxic T        0.9991930477963997      0.9786600194552529      0.9888199520855089      16448.0
CD8+/CD45RA+ Naive Cytotoxic    0.9732510288065843      0.9993963175369756      0.9861504095309009      13252.0
Dendritic       0.9962073324905183      1.0     0.9981000633312223      1576.0
accuracy        0.9920244135322861      0.9920244135322861      0.9920244135322861      0.9920244135322861
macro avg       0.9940577611448117      0.9462920260395996      0.9602407792192658      53413.0
weighted avg    0.9922009470629758      0.9920244135322861      0.9918832155114706      53413.0

        {'train_loss': 0.030774051323533058, 'micro_auroc': 0.9999723477314464, 'micro_aupr': 0.9997267153373384, 'macro_
Results for the val set:
                precision       recall  f1-score        support
CD14+ Monocyte  0.8     0.8228571428571428      0.811267605633803       175.0
CD19+ B 0.8616352201257862      0.8187250996015937      0.839632277834525       502.0
CD34+   1.0     0.6470588235294118      0.7857142857142858      17.0
CD4+ T Helper2  0.0     0.0     0.0     8.0
CD4+/CD25 T Reg 0.6270566727605119      0.6236363636363637      0.6253418413855971      550.0
CD4+/CD45RA+/CD25- Naive T      0.46774193548387094     0.3473053892215569      0.39862542955326463     167.0
CD4+/CD45RO+ Memory     0.4595744680851064      0.3956043956043956      0.4251968503937008      273.0
CD56+ NK        0.9459084604715673      0.8891786179921773      0.9166666666666667      767.0
CD8+ Cytotoxic T        0.8509433962264151      0.7401531728665208      0.7916910473961382      1828.0
CD8+/CD45RA+ Naive Cytotoxic    0.7128396377197656      0.9083503054989817      0.7988059701492536      1473.0
Dendritic       0.7803468208092486      0.7714285714285715      0.7758620689655172      175.0
accuracy        0.7721988205560236      0.7721988205560236      0.7721988205560236      0.7721988205560236
macro avg       0.6823678737892975      0.6331179892942468      0.6517094585175228      5935.0
weighted avg    0.7760081855132062      0.7721988205560236      0.7693552150030722      5935.0

        {'val_loss': 1.0227501392364502, 'micro_auroc': 0.9743836890662605, 'micro_aupr': 0.8544431203186056, 'macro_aupr
Results for the test set:
                precision       recall  f1-score        support
CD14+ Monocyte  0.7605633802816901      0.8350515463917526      0.7960687960687961      194.0
CD19+ B 0.8317025440313112      0.7616487455197133      0.7951356407857811      558.0
CD34+   1.0     0.8947368421052632      0.9444444444444444      19.0
CD4+ T Helper2  0.0     0.0     0.0     9.0
CD4+/CD25 T Reg 0.6585365853658537      0.6176470588235294      0.6374367622259697      612.0
CD4+/CD45RA+/CD25- Naive T      0.518796992481203       0.3709677419354839      0.43260188087774293     186.0
CD4+/CD45RO+ Memory     0.4562043795620438      0.41254125412541254     0.43327556325823224     303.0
CD56+ NK        0.9245049504950495      0.8767605633802817      0.9000000000000001      852.0
CD8+ Cytotoxic T        0.8297752808988764      0.7272279665189562      0.7751246392023091      2031.0
CD8+/CD45RA+ Naive Cytotoxic    0.7103513770180437      0.9144254278728606      0.7995724211651524      1636.0
Dendritic       0.776536312849162       0.7128205128205128      0.7433155080213903      195.0
accuracy        0.7634571645185747      0.7634571645185747      0.7634571645185747      0.7634571645185747
macro avg       0.6788156184530213      0.6476206963176151      0.6597250596408926      6595.0
weighted avg    0.7664747718470555      0.7634571645185747      0.7599773115607741      6595.0

        {'test_loss': 1.048636794090271, 'micro_auroc': 0.9733719838529544, 'micro_aupr': 0.849138604292797, 'macro_aupr'
wandb:
wandb: Run history:
wandb:       epoch ▁▁▁▂▂▂▂▂▃▃▃▃▃▄▄▄▄▄▅▅▅▅▅▆▆▆▆▆▇▇▇▇▇███
wandb:  macro_aupr █▁
wandb:    macro_f1 █▁
wandb:  micro_aupr █▁
wandb: micro_auroc █▁
wandb:    micro_f1 █▁
wandb:   test_loss ▁
wandb:  train_loss █▆▅▅▄▄▃▃▂▂▂▂▂▂▂▂▂▂▂▁▁▁▁▁▁▁▁▁▁▁▁
wandb:    val_loss ▇▅▄▃▂▁▁▁▁▁▁▂▂▂▂▃▃▄▄▅▅▄▆▅▅▆▆▆▆▆▆█
wandb:
wandb: Run summary:
wandb:       epoch 50
wandb:  macro_aupr 0.72433
wandb:    macro_f1 0.65973
wandb:  micro_aupr 0.84914
wandb: micro_auroc 0.97337
wandb:    micro_f1 0.76346
wandb:   test_loss 1.04864
wandb:  train_loss 0.03077
wandb:    val_loss 1.02275
wandb:
wandb: You can sync this run to the cloud by running:
wandb: wandb sync /content/SC-interview/wandb/offline-run-20240805_024534-vd1h7heg
wandb: Find logs at: ./wandb/offline-run-20240805_024534-vd1h7heg/logs
```

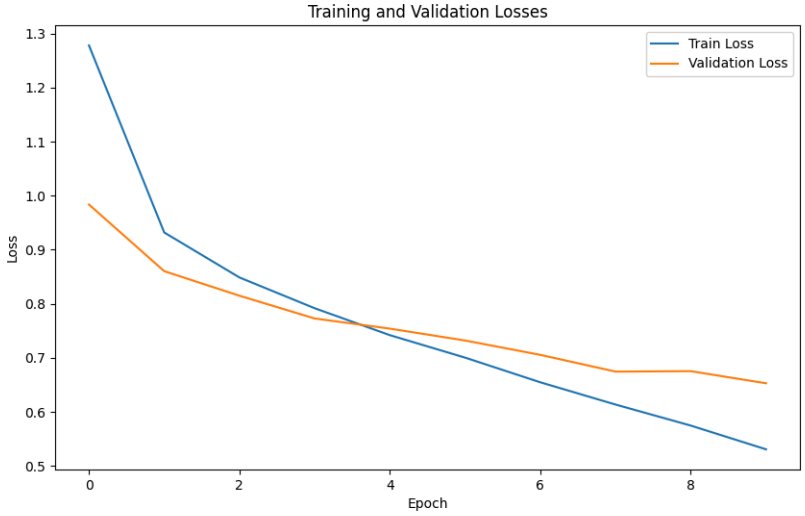
```
wandb: Find logs at: ./wandb/offline-run-20240805_024534-vd1h7neg/logs
wandb: WARNING The new W&B backend becomes opt-out in version 0.18.0; try it out with `wandb.require("core")`! See https:/
```



We overfit very heavily! Let's try 10 epochs instead (early stopping).

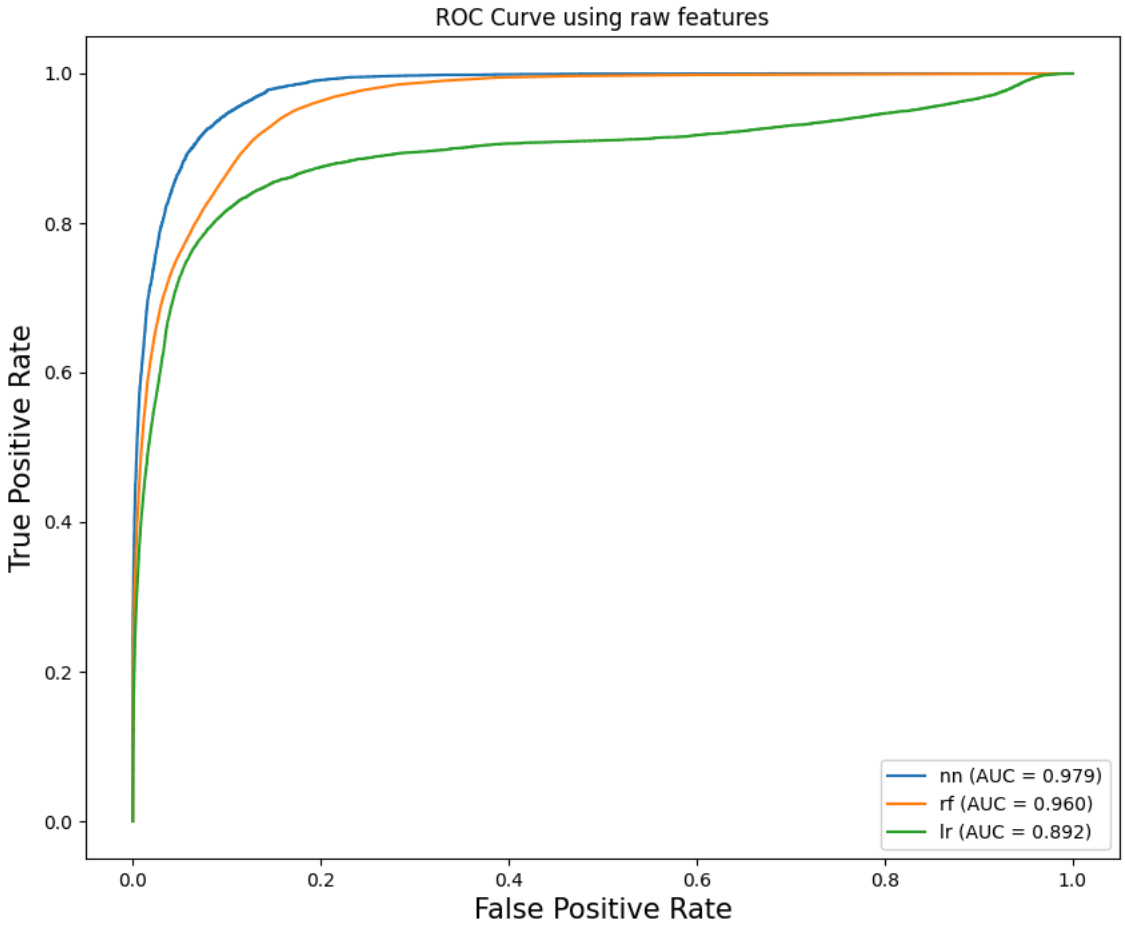
```
! python train.py --method nn --epochs 10 --loss-plot plots/raw_nn_loss_10.png
Image('plots/raw_nn_loss_10.png', width=720, height=432)
```

```
Provided featurization has (65943, 16769) shape for X and (65943,) shape for y
Training: 100% 10/10 [00:13<00:00,  1.33s/epoch, Train Loss=0.5307, Val Loss=0.6529]
Results for the train set:
         {'train_loss': 0.4206043779850006, 'micro_auroc': 0.9929624882242312, 'micro_aupr': 0.9444357038880338, 'macro_aupr
Results for the val set:
         {'val_loss': 0.6529132723808289, 'micro_auroc': 0.9787925426322368, 'micro_aupr': 0.8547440510131012, 'macro_aupr':
Results for the test set:
         {'test_loss': 0.6578068733215332, 'micro_auroc': 0.9785791554127263, 'micro_aupr': 0.8519300630658467, 'macro_aupr
wandb: WARNING The new W&B backend becomes opt-out in version 0.18.0; try it out with `wandb.require("core")`! See https://
```



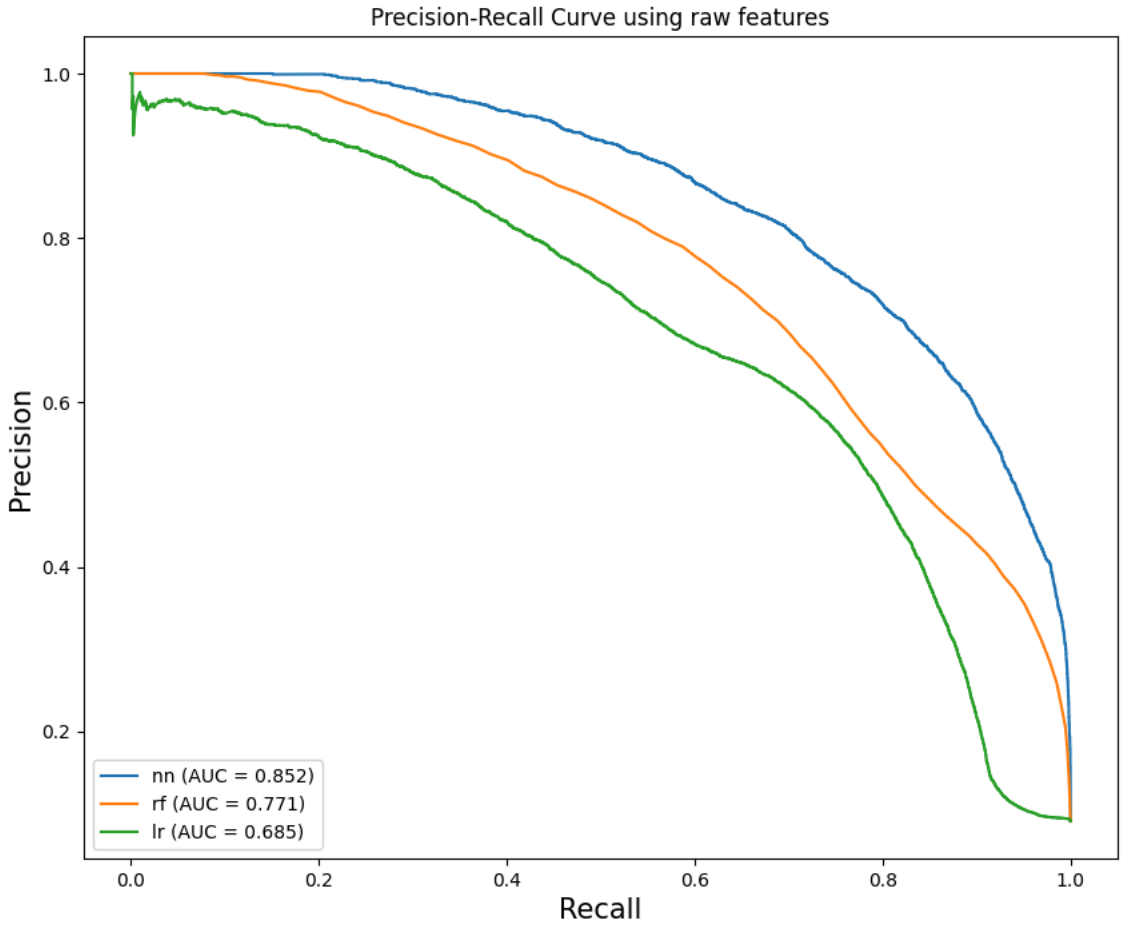
Let's visualize ROC and PR curves for the three classifiers.

```
%run utils.py # this gives us a plot_curves helper function
methods = ['nn', 'rf', 'lr']
plot_curves(methods, 'roc', title="ROC Curve using raw features")
plot_curves(methods, 'prc', title="Precision-Recall Curve using raw features")
Image('plots/roc_comparison.png')
```




```
<Figure size 640x480 with 0 Axes>
```

```
Image('plots/prc_comparison.png')
```



Addressing overfitting by removing non-variable genes

Our neural network above has much higher AUPR on the training data than on the test data, meaning the generalization gap is large. Since we know that only a subset of our genes are highly variable, e.g. explain the variance in our dataset, let's try training on only those genes.

```
! python embed.py --featurizer hvg --out-file hvg_embeddings.h5ad
```

Show hidden output

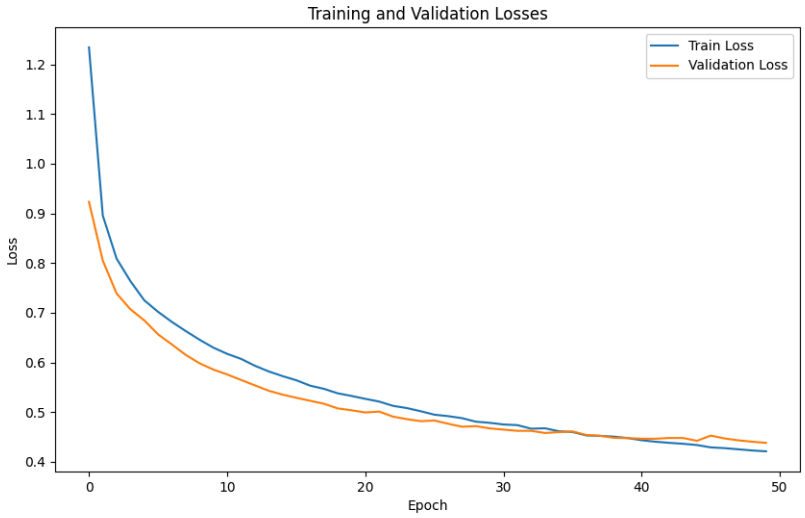
We can re-run benchmarks and compute new ROC and PR curves, using the hvg embeddings as features.

```
! python train.py --method lr --data hvg_embeddings.h5ad
! python train.py --method rf --data hvg_embeddings.h5ad
```

```
Provided featurization has (65943, 1800) shape for X and (65943,) shape for y
Results for the test set:
Results for the test set:
         {'micro_auroc': 0.9749110044425643, 'micro_aupr': 0.831029580736889, 'macro_aupr': 0.6704593496318331, 'micro_f1':
Provided featurization has (65943, 1800) shape for X and (65943,) shape for y
Results for the test set:
Results for the test set:
         {'micro_auroc': 0.9706814464745444, 'micro_aupr': 0.8104784583858283, 'macro_aupr': 0.6685477865256043, 'micro_f1':
```

```
! python train.py --method nn --data hvg_embeddings.h5ad --loss-plot plots/hvg_nn_loss_50.png
Image('plots/hvg_nn_loss_50.png', width=720, height=432)
```

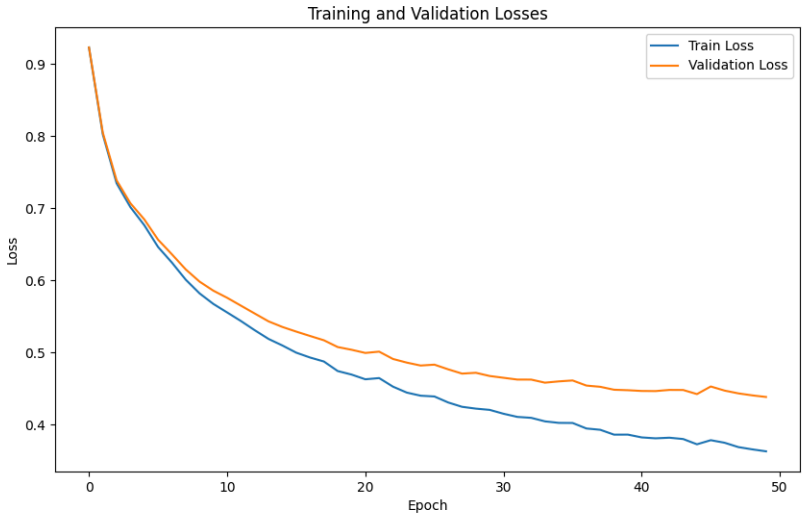
```
Provided featurization has (65943, 1800) shape for X and (65943,) shape for y
Training: 100% 50/50 [01:03<00:00,  1.26s/epoch, Train Loss=0.4212, Val Loss=0.4383]
Results for the train set:
         {'train_loss': 0.36292052268981934, 'micro_auroc': 0.9931762614434989, 'micro_aupr': 0.9469621935059652, 'macro_aup
Results for the val set:
         {'val_loss': 0.4383322596549988, 'micro_auroc': 0.9900036636718054, 'micro_aupr': 0.92484008024682, 'macro_aupr': 0
Results for the test set:
         {'test_loss': 0.45920440554618835, 'micro_auroc': 0.9890710586109243, 'micro_aupr': 0.9196806324115885, 'macro_aupr
wandb: WARNING The new W&B backend becomes opt-out in version 0.18.0; try it out with `wandb.require("core")`! See https://w
```



The validation curve is likely lower than the training curve for two reasons. First, we use dropout in evaluation but not training, which increases training loss. Second, the eval loss is computed after a whole epoch, whereas the training loss is aggregated per batch in the epoch. To verify nothing strange is going on, let's use the --recompute-train-loss flag.

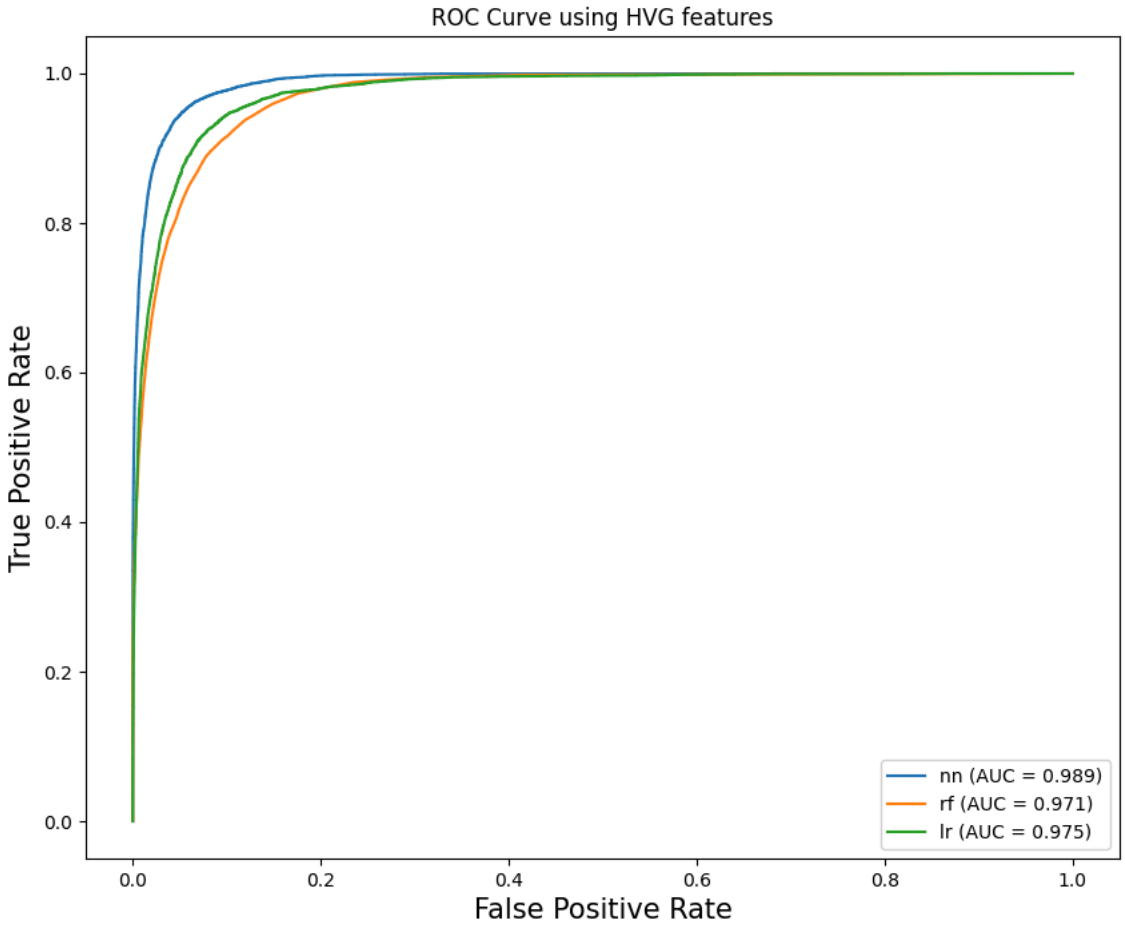
```
! python train.py --method nn --data hvg_embeddings.h5ad --loss-plot plots/hvg_nn_loss_50_corrected.png --recompute-train-loss
Image('plots/hvg_nn_loss_50_corrected.png', width=720, height=432)
```

```
Provided featurization has (65943, 1800) shape for X and (65943,) shape for y
Training: 100% 50/50 [01:02<00:00,  1.25s/epoch, Train Loss=0.3629, Val Loss=0.4383]
Results for the train set:
         {'train_loss': 0.36292052268981934, 'micro_auroc': 0.9931762614434989, 'micro_aupr': 0.9469621935059652, 'macro_aup
Results for the val set:
         {'val_loss': 0.4383322596549988, 'micro_auroc': 0.9900036636718054, 'micro_aupr': 0.92484008024682, 'macro_aupr': 0
Results for the test set:
         {'test_loss': 0.45920440554618835, 'micro_auroc': 0.9890710586109243, 'micro_aupr': 0.9196806324115885, 'macro_aupr
wandb: WARNING The new W&B backend becomes opt-out in version 0.18.0; try it out with `wandb.require("core")`! See https://w
```



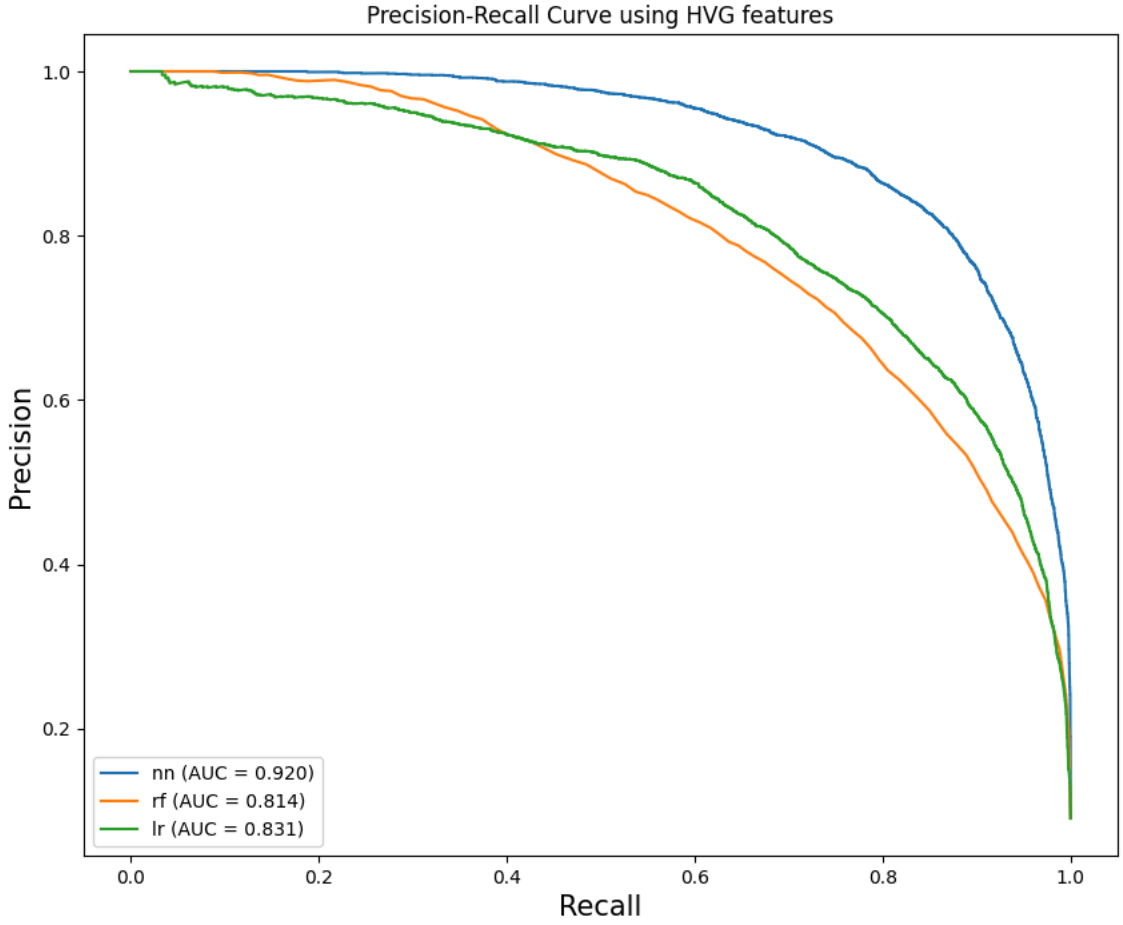
Things make sense again now 🙂 . Let's recompute ROC and PR curves.

```
%run utils.py # this gives us a plot_curves helper function
methods = ['nn', 'rf', 'lr']
plot_curves(methods, 'roc', title="ROC Curve using HVG features")
plot_curves(methods, 'prc', title="Precision-Recall Curve using HVG features")
Image('plots/roc_comparison.png')
```




```
<Figure size 640x480 with 0 Axes>
```

```
Image('plots/prc_comparison.png')
```



All methods are greatly improved by considering only the highly variable genes! Removing the sparse genes likely helps the learning algorithms focus on what's important.

Using scGPT foundation model embeddings

Lastly, let's try using a foundation model developed for single-cell data. Here we use scGPT to compute gene embeddings and train on those.

```
! python embed.py --featurizer scgpt --out-file scgpt_embeddings.h5ad
```

```
/usr/local/lib/python3.10/dist-packages/scgpt/model/model.py:21: UserWarning: flash_attn is not installed
  warnings.warn("flash_attn is not installed")
/usr/local/lib/python3.10/dist-packages/scgpt/model/multiomic_model.py:19: UserWarning: flash_attn is not installed
  warnings.warn("flash_attn is not installed")
Computed highly variable genes
/content/SC-interview/embed.py:82: ImplicitModificationWarning: Trying to modify attribute `.var` of view, initializing view
  adata.var[gene_col] = adata.var['gene_id'].apply(lambda id: ensembl_df.loc[id, 'feature_name'] if id in ensembl_df.index e
Converted ENSEMBL IDs to gene symbols
scGPT - INFO - match 1717/1800 genes in vocabulary of size 60697.
/usr/local/lib/python3.10/dist-packages/scgpt/model/model.py:77: UserWarning: flash-attn is not installed, using pytorch tra
  warnings.warn(
Embedding cells: 100% 1031/1031 [00:36<00:00, 28.40it/s]
/usr/local/lib/python3.10/dist-packages/scgpt/tasks/cell_emb.py:279: ImplicitModificationWarning: Setting element `.obsm['X_
  adata.obsm["X_scGPT"] = cell_embeddings
```

```
! python train.py --method lr --data scgpt_embeddings.h5ad
! python train.py --method rf --data scgpt_embeddings.h5ad
```

```
Provided featurization has (65943, 1717) shape for X and (65943,) shape for y
Results for the test set:
Results for the test set:
         {'micro_auroc': 0.9725809073775076, 'micro_aupr': 0.819081701601465, 'macro_aupr': 0.6630551038859934, 'micro_f1':
Provided featurization has (65943, 1717) shape for X and (65943,) shape for y
Results for the test set:
Results for the test set:
         {'micro_auroc': 0.9717497150470669, 'micro_aupr': 0.8190467663337506, 'macro_aupr': 0.6752539992124774, 'micro_f1':
```

This time we can visualize the embeddings from our trained model.

```
! python train.py --method nn --data scgpt_embeddings.h5ad --loss-plot scgpt_nn_loss.png --epochs 100 --output-latents embeddin
Image('scgpt_nn_loss.png', width=720, height=432)
```

```
Provided featurization has (65945, 1717) shape for X and (65945,) shape for y
Training: 100% 100/100 [02:06<00:00,  1.27s/epoch, Train Loss=0.3719, Val Loss=0.4500]
Results for the train set:
                precision       recall  f1-score        support
CD14+ Monocyte  0.9165628891656289      0.9346031746031747      0.9254951273184533      1575.0
CD19+ B 0.9508335688047471      0.7446337685328612      0.8351948374286423      4519.0
CD34+   0.9931972789115646      0.9605263157894737      0.9765886287625417      152.0
CD4+ T Helper2  1.0     0.10666666666666667     0.1927710843373494      75.0
CD4+/CD25 T Reg 0.758418463866818       0.809245054501413       0.7830078125000001      4954.0
CD4+/CD45RA+/CD25- Naive T      0.748641304347826       0.3663563829787234      0.49196428571428574     1504.0
CD4+/CD45RO+ Memory     0.7305835892935498      0.6782077393075356      0.7034220532319392      2455.0
CD56+ NK        0.9851301115241635      0.9213385484571925      0.9521670783741298      6903.0
CD8+ Cytotoxic T        0.8797674418604651      0.9199902723735408      0.8994293865905849      16448.0
CD8+/CD45RA+ Naive Cytotoxic    0.8353841946951053      0.9221249622698461      0.8766140602582496      13252.0
Dendritic       0.9096858638743456      0.881979695431472       0.8956185567010309      1576.0
accuracy        0.8671671690412447      0.8671671690412447      0.8671671690412447      0.8671671690412447
macro avg       0.8825640642131103      0.7496065982647182      0.7756611737470188      53413.0
weighted avg    0.8690404592673661      0.8671671690412447      0.8637530936822347      53413.0

        {'train_loss': 0.3718605935573578, 'micro_auroc': 0.9929983671745608, 'micro_aupr': 0.9448230079459095, 'macro_au
Results for the val set:
                precision       recall  f1-score        support
CD14+ Monocyte  0.8430232558139535      0.8285714285714286      0.8357348703170029      175.0
CD19+ B 0.9656084656084656      0.7270916334661355      0.8295454545454546      502.0
CD34+   0.8666666666666667      0.7647058823529411      0.8125  17.0
CD4+ T Helper2  0.0     0.0     0.0     8.0
CD4+/CD25 T Reg 0.6909090909090909      0.76    0.7238095238095238      550.0
CD4+/CD45RA+/CD25- Naive T      0.7     0.3772455089820359      0.490272373540856       167.0
CD4+/CD45RO+ Memory     0.6738197424892703      0.575091575091575       0.6205533596837944      273.0
CD56+ NK        0.9709944751381215      0.9165580182529335      0.9429912810194501      767.0
CD8+ Cytotoxic T        0.8614900314795383      0.8982494529540481      0.8794858061060525      1828.0
CD8+/CD45RA+ Naive Cytotoxic    0.8080931943592887      0.8947725729803123      0.8492268041237113      1473.0
Dendritic       0.8011049723756906      0.8285714285714286      0.8146067415730337      175.0
accuracy        0.8372367312552653      0.8372367312552653      0.8372367312552653      0.8372367312552653
macro avg       0.743791808621826       0.6882597728384399      0.7089751104289891      5935.0
weighted avg    0.8387399878632366      0.8372367312552653      0.8340895266235585      5935.0

        {'val_loss': 0.450032502412796, 'micro_auroc': 0.9895628605029636, 'micro_aupr': 0.9210691674169793, 'macro_aupr'
Results for the test set:
                precision       recall  f1-score        support
CD14+ Monocyte  0.8137254901960784      0.8556701030927835      0.8341708542713568      194.0
CD19+ B 0.9056179775280899      0.7222222222222222      0.8035892323030907      558.0
CD34+   1.0     0.9473684210526315      0.972972972972973       19.0
CD4+ T Helper2  0.0     0.0     0.0     9.0
CD4+/CD25 T Reg 0.684971098265896       0.7745098039215687      0.7269938650306748      612.0
CD4+/CD45RA+/CD25- Naive T      0.5942028985507246      0.22043010752688172     0.32156862745098036     186.0
CD4+/CD45RO+ Memory     0.6538461538461539      0.6171617161716172      0.634974533106961       303.0
CD56+ NK        0.9696202531645569      0.8990610328638498      0.9330085261875761      852.0
CD8+ Cytotoxic T        0.8485280151946819      0.879862136878385       0.8639110466521633      2031.0
CD8+/CD45RA+ Naive Cytotoxic    0.8082267926625903      0.8887530562347188      0.8465793304221252      1636.0
Dendritic       0.8162162162162162      0.7743589743589744      0.7947368421052632      195.0
accuracy        0.8259287338893101      0.8259287338893101      0.8259287338893101      0.8259287338893101
macro avg       0.7359049905113625      0.6890361431203302      0.7029550755002876      6595.0
weighted avg    0.825009107196061       0.8259287338893101      0.8211299145291585      6595.0

        {'test_loss': 0.4752683937549591, 'micro_auroc': 0.9883713199686623, 'micro_aupr': 0.9135836844279941, 'macro_aup
wandb: WARNING The new W&B backend becomes opt-out in version 0.18.0; try it out with `wandb.require("core")`! See https:/
```