University of Barcelona
Open University of Catalonia

Master's thesis
Biostatistics and Bioinformatics

Prediction of single-cell RNA-seq gene expression under small molecule drug perturbations with scGEN

By

Aitor Moruno Cuenca

Advisor:

Izaskun Mallona

12 october of 2023
Academic year 2023-2024

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Prediction of single-cell RNA-seq gene expression under small molecule drug perturbations with scGEN*
Nombre del autor:	*Aitor Moruno Cuenca*
Nombre del consultor/a:	*Dr. Izaskun Mallona*
Nombre del PRA:	*Dr. David Merino*
Fecha de entrega (mm/aaaa):	*12/2023*
Titulación o programa:	Master en Bioestadística y Bioinformática
Área del Trabajo Final:	*Análisis de datos ómicos*
ºIdioma del trabajo:	*Inglés*
Palabras clave	*scRNA-seq, single-cell perturbation, scGEN*

Resumen del Trabajo

El análisis de perturbación en single-cell puede ayudar a desengranar las sutilezas moleculares de las células cuando se les aplica un estímulo, como por ejemplo compuestos químicos. Estos comportamientos moleculares a los estímulos celulares son de vital importancia para entender los mecanismos de progresión de enfermedades, respuestas a fármacos y muchos otros mecanismos biológicos.

Los datos experimentales usados consistieron en células PBMCs (Peripherial Blood Mononuclear Cells) extraídas de tres donantes para las que se secuenció scRNA-seq (Single-cell RNA sequencing) aplicando más de 144 compuestos químicos presentes en Library of Integrated Network-Based Cellular Signatures (LINCS).

En esta tesis se generan datos de expresión génica RNA-seq in silico para células reguladoras T y dos compuestos químicos, Cabozantinib y Resminostat,, y se validan con los datos de expresión génica observados. Los datos son generados mediante scGEN, un método reciente considerado el gold-standard en problemas de perturbación de single-cell.

Los métodos de validación, regresión lineal y sesgo respecto recta identidad, muestran que las predicciones de expresión génica para ambos compuestos son muy acertados, siendo $R^2 = 0.97$ y $R^2 = 0.93$ para Cabozantinib y Resminostat, respectivamente. Para ambos compuestos, se observó un ligero sesgo negativo en las predicciones para genes con un número alto de counts normalizados.

Abstract

Single-cell perturbation analysis can help unravel the molecular intricacies of cells when subjected to a stimulus, such as chemical compounds. These molecular responses to cellular stimuli are of vital importance in understanding disease progression mechanisms, drug responses, and many other biological processes.

The experimental data used consisted of PBMCs (Peripheral Blood Mononuclear Cells) extracted from three donors, for which scRNA-seq (Single-cell RNA sequencing) was performed, applying more than 144 chemical compounds from the Library of Integrated Network-Based Cellular Signatures (LINCS).

In this thesis in silico RNA-seq gene expression data are generated for regulatory T cells and two chemical compounds, Cabozantinib and Resminostat, and validated against observed gene expression data. The data are generated using scGEN, a recent method considered the gold standard in single-cell perturbation problems.

The validation methods, linear regression and bias towards the identity line, demonstrate that the gene expression predictions for both compounds are highly accurate, with $R^2 = 0.97$ and $R^2 = 0.93$ for Cabozantinib and Resminostat, respectively. For both compounds, a slight negative bias was observed in predictions for genes with a high number of normalized counts.

Contents

List of Figures

Introduction

0.1 Context and justification of the thesis

In the last decade, the field of molecular biology has witnessed a paradigm shift in its way to conduct research thanks to omics analysis. Such transformative shift due to lower sequencing costs, transition from bulk measurements to single-cell analysis and unprecented volume of data, is opening new horizons on our understanding of cellular heterogeneity.

The complexity of human biology cannot be measured by traditional bulk sequencing approaches, in part due to the function and interplay of approximately 37 trillion cells in the human body. However, advances in single-cell technologies are providing unmatched insight into the function of cells and tissues under multiple omic frames: genomics, transcriptomics, proteomics, epigenomics and metabolomics. Yet the use of single-cell technologies for compound screening requires mapping a causation between a chemical compound and the molecular impact on the cell state. Such experiments are not always available due to their high intesive labor and cost, even not all cells are suitable fo high-throughput transcriptomic screening.

Under a more general framework, single-cell perturbation analysis, a key research area in molecular biology, can help to decipher the subtle molecular nuances of cellular responses to stimuli (also called perturbation) such as: chemical compounds, genetic alterations, environmental factors, and so on. Traditional bulk-RNA measurements mask the variability inherent in cellular responses, as they average gene expression indistringuishably of its cell type. However, by accounting cellular gene expression variability, single-cell perturbation analysis helps to capture the diversity of cellular gene expression behaviors induced by stimuli. These cellular behaviors are crucial in understanding disease progression, disease biology, drug response and any other fundamental biological mechanisms. Hence, if single-cell perturbation analysis accurately predicts chemical perturbation in new cell types, it could reduce development time of new treatments.

0.2 General description

For this thesis, we will use an already generated single-cell perturbational dataset of human peripheral blood mononuclear cells (PBMCs). Such dataset contains scRNA-seq measurements after 24h of induction of a chemical compound, the original design considered a total 144 compounds from the Library of Integrated Network-Based Cellular Signatures (LINCS) selected on diverse transcriptional signatures observed in CD34+ hematopoietic stem cells. The experiment was repeated in three healthy human donors and ene expression data is obtained by scRNA-seq using the 10x Multiome assay.

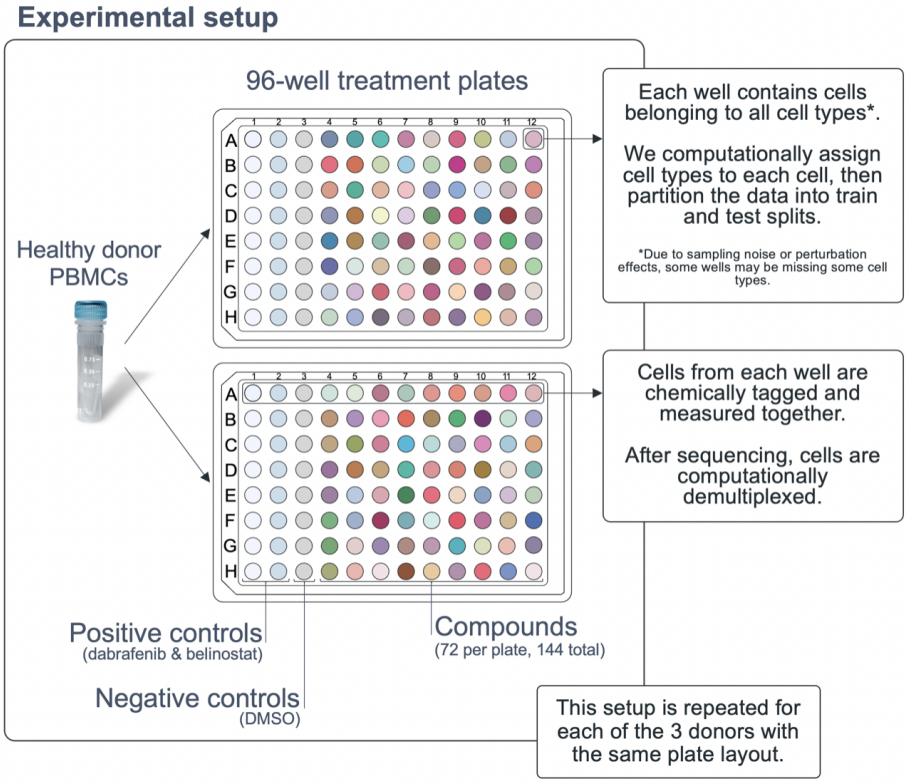

Figure 1: Experimental design of the PBMCs dataset

Source: Extracted from [1]

The PBMCs from donors were plated on 96-well plates. The first two columns shown in the picture above are the positive controls (dabrafenib and belinostat) and the next one belongs to the negative controls (DMSO). The reason to select positive and negative controls in the experimental design is its impact on gene expression, as positive controls have a large impact on transcription. Each of the wells contain PBMCs, that are of different types including: T cells (CD4+, CD8+, regulatory), NK cells, Myeloid (macrophages, monocytes) and B cells.

We will predict gene expression values for T-regulatory cells for a majority of compounds, that is compounds not used in training set. Only compounds for which we will not predict RNA-seq measured for T-regulatory cells will be used for training, this will mimic a scientific context where we might want to make predictions into new cell types (not measured) while taking only 10% of measurements in such cell type.

There are several challenges associated with the data of this experiment:

- High dimensionality. There are 18211 genes to consider in the analysis, this means the feature space is very high-dimensional
- Cell type specificity. Each cell type is completely unique from a transcriptional point of view, hence different cell types may transcriptionally respond different to the same compound, there is an inherent variability on the transcriptional changed induced by the compound for each cell type.

To tackle the single-cell perturbation problem we will use scGEN, an autoencoder-based method for single-cell perturbation analysis. The reason to choose this method is that its an autoencoder specifically designed for single-cell expression data and its able to capture cell-to-cell variability. However, its limited by the quantity and quality of training data, as much of the autoencoder approaches in single-cell perturbation.

scGEN is a computational tool developed for the analysis of single-cell RNA sequencing (scRNA-seq) data in the context of single-cell perturbation analysis. It utilizes deep generative models, specifically variational autoencoders, to accurately model cellular responses to perturbations. scGEN can predict the response of cells under stimuli, even for phenomena that are absent from the training data, across different cell types, studies, and species [1]. It captures features that distinguish responding from non-responding genes and cells, enabling experimental design and in silico screening of perturbation response in the context of disease and drug treatment.

0.3 Objectives

Main objectives:

- Predict how small molecules change gene expression in non measured cell types.
- Understand the general framework (experimental design, dataset, code) needed to perfom a single-cell perturbation analysis

Specific objectives:

- Predict gene expression data of T-regulatory cells on compounds not used for training data and compare the performance with the groundtruth.
- Use scGEN as a generative method to predict gene expression.
- Write all the code necessary, in Python, to perform the analysis with scGEN on the experimental data.

0.4 Approach and methodology

When it comes to single-cell perturbation analysis, there are two possible methodological approaches: machine learning based and autoencoder-based methods, the latter is based on deep learning.

In machine learning methods, such as random forest and XGBoost, utilize algorithms to learn patterns from the input data and make out-of-bag predictions from the model. These methods are effective when the input data is well-structured and features are explicitly defined. Unfortunately, that is not the case in single-cell analysis and they may struggle with capturing the complex relationships and heterogeneity present in single cell.

On the other hand, autoencoder-based methods, such as Dr.VAE, scGEN and chemCPA, use variational autoencoders (VAEs) to model cellular responses to perturbations. These methods encode the input data into a lower-dimensional latent space and then decode it to reconstruct the oriignal data. Autoencoders have the advantage because they can capture non-linear relationships and inherent variability in the data, two appealing requirements for single-cell type data.

Among the autoencoder-based methods, scGEN is considered the best tool for single-cell perturbation analysis up to now. Firstly, scGEN utilizes deep generative models to capture biological varaibility and heterogeneity in single-cell RNA sequencing data. It can also accuately impute missing values, making it useful for downstream analysis. Additionally, scGEN can accurately model cellular responses to perturbations, even for those not present in the training set. Its ability to predict perturbation responses across diferent cell types and species sets, being purposedly designed for single-cell RNA sequence, makes it the most powerful tool in single-cell perturbation analysis.

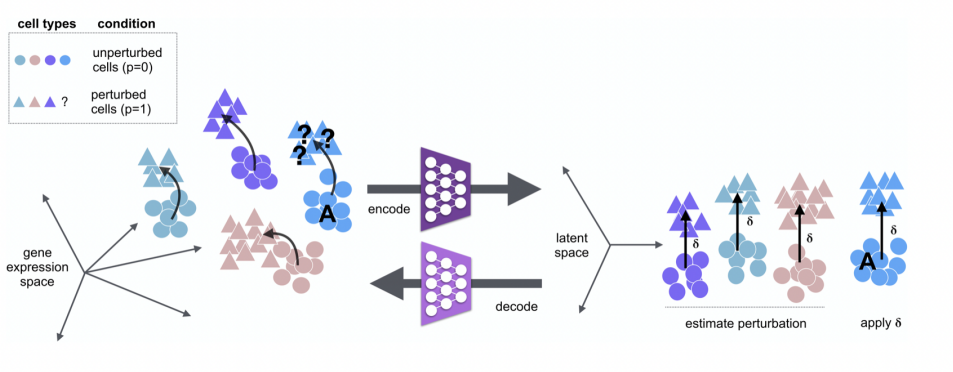

Figure 2: General framework of scGEN method

Source: Extracted from [2]

scGEN assumes we have several cell types for which we have, at least, one condition with two states: perturbed and unperturbed. For some cell types we will have RNA gene expression data measured under both conditions, meanwhile for other cell types only one of the two conditions will be measured. An encoder network will learn the representation of the data on a low dimensional space called latent space, there the decoder will infer the gene expression of the non observed perturbation cells based on the latent space representation. Note that in our case the condition is the small molecule with multiple perturbed states, one for each chemical compound. However, the reasoning with multiple perturbed stated remains unchanged.

0.5 Planification of the project

The tasks for this project are: literature review, data preprocessing, write Python code for scGEN analysis, scGEN model training, validation of the predictions with the groundtruth.

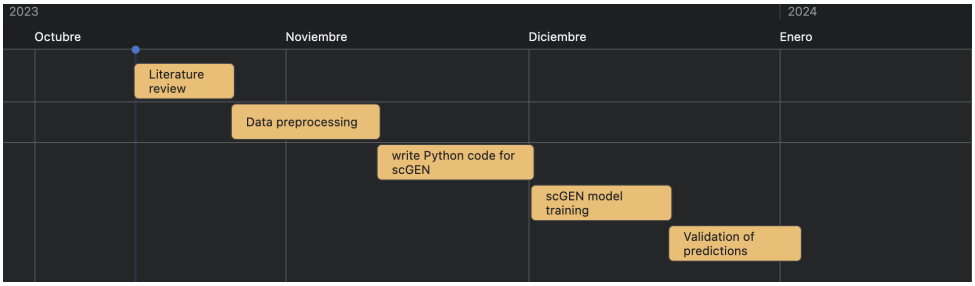

Figure 3: Gantt chart of projec tasks

Literature review will help us write the theoretical framework of scGEN, to make sure every detail of the methodology is well understood and the general problem well-posed. Later on we will start by processing all the data needed for the analysis: dataset conversion to the right format, creating training/test/validation sets, filtering variables when needed... Once this task is completed, we will start writing the Python code for scGEN analysis, and then train the model. Last but not least, we will validate the predictions of the model with the groundtruth, those observations not included in the training set.

Some risks can be elucidated for this project:

- As scGEN is based on a Variational Autoencoder, it is possible the model does not converge, in such case different hyperparameters will be tested.
- The complexity of the method and the time-constraints for the development of the thesis are an inherent risk.
- Problems with the datasets for training: difficulty converting the datasets to the right data structure in Python.

0.6 Expected outputs

The products obtained from this thesis are:

- Manuscript of the thesis and slides for the oral presentation of it.
- Datasets processed in the right format to be analyed
- Pipeline, in Python, to perform the scGEN analyses on the experimental data of this thesis
- Graphics for the correlation structure validation of the gene expression prediction of T-regulatory cells under the perturbation of chemical compounds

Chapter 1

Development

1.1 Theoretical framework

Throughout this section we follow [2] and extend or detail when required.

1.1.1 Variational autoencoders

A variational autoencoder is a type of neural network consisting of an encoder and a decoder, similar to classical autoencoders. Contrary to classical autoencoders, VAEs are able to generate new data points not present in the training set.

Mathematically, VAEs maximizes the likelihood of each sample x_i in the training set under a generative probabilistic process:

$$P(x_i|\theta) = \int P(x_i|z_i;\theta)P(z_i|\theta)dz_i$$

where θ is a model parameter, in our case represented by a neural network, and z_i is the latent variable. VAEs sample latent variables z_i, that is non-observable variables, that are likely to generate an observed variable x_i and using those to compute the probability $P(x_i|\theta)$.

Unfortunately, the true posterior distribution $P(z_i|x_i,\theta)$ is usually very high-dimensional and without an analytic closed form. However, variational inference allow us to use a so-called variational distribution $Q(z_i|x_i,\phi)$, which is modeled by a neural network with parameter ϕ, called the inference network (autoencoder).

To assess the approximation of $Q(z_i|x_i,\phi)$ to $P(z_i|x_i,\theta)$, the Kullback-Leibler (KL) divergence is the most used pseudo-distance between probability distributions.

$$KL(Q(z_i|x_i,\phi)||P(z_i|x_i,\theta)) = E_{Q(z_i|x_i,\phi)}[Q(z_i|x_i,\phi) - P(z_i|x_i,\theta)]$$

The distribution used for $Q(z_i|x_i)$ will be a multivariate Gaussian, i.e $Q(z_i|x_i) = N(z_i;\mu_\phi(x_i),\Sigma_\phi(x_i))$ where $\mu_\phi(x_i)$ and $\Sigma_\phi(x_i)$ are implemented with the encoder neural network and $\Sigma_\phi(x_i)$ is constrained to be a diagonal matrix, that is assuming independece as correlation is 0.

All in all, a Monte Carlo sampling from $Q(z_i|x_i,\phi)$ is performed and with stochastic gradient descent the following loss function is optimized:

$$\ell(x_i) = \frac{1}{L}\sum_{l=1}^{L} logP(x_i|z_{i,l},\theta) - \alpha KL[Q(z_{i,l}|x_i,\phi)||P(z_{i,l}|\theta)]$$

This last expression is also-known as ELBO. Where α is the learning rate parameter and is adjusted in the hyperparameter tuning. When convergence is attained the VAEs has converged and the optimal approximation distribution $Q(z_i|x_i,\phi)$ of $P(z_i|\theta)$ is found.

1.1.2 scGEN: combining variational autoencoders and latent space vector arithmetics

Let every cell i with expression profile x_i be characterized by a variable p_i, which is the discrete attribute across the whole manifold, one can think of such as perturbation, species or batch. It is assumed only two conditions define the manifold 0 (unperturbed) and 1 (perturbed), although this can be extended to multiple conditions. Consider the conditional distribution of the expression profile based on a low-dimensional latent variable z_i and a variable p_i, i.e $P(x_i|z_i, p_i)$.

VAE is used to model $P(x_i|z_i, p_i)$ in its dependence on z_i, while vector arithmetics is used to model the dependence on p_i.

Consider a typical extrapolation problem. Assume cell type A exists in the training data only in the unperturbed state $p_A = 0$. Using the observed $z_{A,p=0}$ as the starting point, we predict the latent representation of the perturbed cell (P=1) of cell type A using $\widehat{z}_{A,p=1} = z_{A,p=0} + \delta$, where δ is the difference vector of means between cells in the training set in condition 0 and 1.

To estimate δ scGEN does the following:

- Extract all cells for each condition existent in the training set.
- For each cell type, upsample the cell type sizes to be equal to the maximum cell type size for that condition.
- To remove the population size bias, randomly downsample the condition with a higher sample size to match the sample size of the other condiiton.
- Estimate $\widehat{\delta} = \overline{z}_{p=0} - \overline{z}_{p=1}$, where z_p denotes the latent representation of cells in condition p.

1.2 Data preprocessing

In this section one will find the data preprocessing working pipeline, challenges and solutions faced during the preprocessing of the that is used data throughout the thesis.

Two input datasets have been used in this section:

- `adata_train.parquet`: this dataset is a parquet file containing the single-cell gene expression data with counts and normalized counts of the experiment. The dimension of the matrix is 416442312 $\times$ 4, each row being a combination of cell and gene. Note that this matrix is not a count matrix, hence we will need to transform it into a count matrix containing the normalized counts. This is the main file we will use to train the scGEN algorithm.

	obs_id	gene	count	normalized_count
0	000006a87ba75b72	AATF	1	5.567933
1	000006a87ba75b72	ABHD12	1	5.567933
2	000006a87ba75b72	ABHD3	1	5.567933
3	000006a87ba75b72	AC004687.1	1	5.567933
4	000006a87ba75b72	AC009779.2	1	5.567933
...	...	...	...	...
416442307	ffffe67500d95d8d	ZSWIM7	1	4.205181
416442308	ffffe67500d95d8d	ZSWIM8	1	4.205181
416442309	ffffe67500d95d8d	ZUP1	1	4.205181
416442310	ffffe67500d95d8d	ZXDB	1	4.205181
416442311	ffffe67500d95d8d	ZYX	4	5.580224

[416442312 rows x 4 columns]

Figure 1.1: Adata_train.parquet dataset view

- `adata_meta.csv`: this dataset is a csv file containing the metadata of each cell. There are multiple features: plate name, well, row, cell type, compound name, dose of the compound.

	obs_id	library_id	plate_name	well	row	col	cell_id	\
0	000006a87ba75b72	library_4	plate_4	F7	F	7	PBMC	
1	0000233976e3cd37	library_0	plate_3	D4	D	4	PBMC	
2	0001533c5e876362	library_2	plate_0	B11	B	11	PBMC	
3	00022f989630d14b	library_35	plate_2	E6	E	6	PBMC	
4	0002560bd38ce03e	library_22	plate_4	B6	B	6	PBMC	
...	...	...	...	...	..	...	...	
240085	ffff28f274e983df	library_27	plate_0	G12	G	12	PBMC	
240086	ffff32893af5befb	library_31	plate_4	E7	E	7	PBMC	
240087	ffff6c3e0a7b05ad	library_38	plate_1	C5	C	5	PBMC	
240088	ffff8e571c7e8cb0	library_28	plate_5	B1	B	1	PBMC	
240089	ffffe67500d95d8d	library_9	plate_3	E1	E	1	PBMC	

	donor_id	cell_type	sm_lincs_id	sm_name	\
0	donor_2	T cells CD4+	LSM-4944	MLN 2238	
1	donor_1	T cells CD4+	LSM-46203	BMS-265246	
2	donor_0	T regulatory cells	LSM-45663	Resminostat	
3	donor_0	T cells CD4+	LSM-43216	FK 866	
4	donor_2	T cells CD4+	LSM-1099	Nilotinib	
...	...	...	...	...	
240085	donor_0	NK cells	LSM-3349	Mometasone Furoate	
240086	donor_2	T cells CD4+	LSM-2287	Midostaurin	
240087	donor_2	NK cells	LSM-45786	BAY 87-2243	
240088	donor_1	B cells	LSM-43181	Belinostat	
240089	donor_1	Myeloid cells	LSM-43181	Belinostat	

	SMILES	dose_uM	\
0	CC(C)C[C@H](NC(=O)CNC(=O)c1cc(Cl)ccc1Cl)B(O)O	1.0	
1	CCCCOc1c(C(=O)c2c(F)cc(C)cc2F)cnc2[nH]ncc12	1.0	
2	CN(C)Cc1ccc(S(=O)(=O)n2ccc(/C=C/C(=O)NO)c2)cc1	1.0	
3	O=C(/C=C/c1cccnc1)NCCCCC1CCN(C(=O)c2ccccc2)CC1	1.0	
4	Cc1cn(-c2cc(NC(=O)c3ccc(C)c(Nc4nccc(-c5cccnc5)...	1.0	

Figure 1.2: Adata_meta.csv dataset view

The first step in the preprocessing of the data will be to transform this datasets into an `AnnData` object. See below a representation of such object.

An AnnData object in Python is a fundamental data structure used in the analysis of single-cell RNA sequencing (scRNA-seq) data. It is a core component of the AnnData package, which is commonly used in the field of single-cell analysis. An AnnData object is designed to store, manipulate, and analyze high-dimensional and heterogeneous single-cell data efficiently.

Such object is a container that holds all the data and metadata associated with single-cell experiments. It is particularly well-suited for representing scRNA-seq data. Here's a description of each AnnData components:

- **.X**: data matrix containing the gene expression data. This matrix can be either a numpy array or a scipy sparse matrix, usually a scipy sparse matrix is the most convenient as it stores the count matrix in an efficient way which works best for large experiments such as our case. This component has been derived based on adata_train.parquet.

- **.obs**: these are the annotations of observations contained in component .X. This is also known as cell metadata, as each row of .X matrix is linked to a cell. This component has been derived based on adata_meta.csv.

- **.var**: these are the annotations of the columns contained in component .X. This is also known as gene metadata, as each column of .X matrix is linked to a gene (in transcriptomics analysis). In this case, we do not have any gene annotations, nor are relevant for this project, the .var component will be based on the gene identifiers to map .X matrix into a scipy sparsed format matrix.

- **.uns**: the unstructured annotations usually contain the experiment metadata. In our case this component is not relevant nor we do have this information.

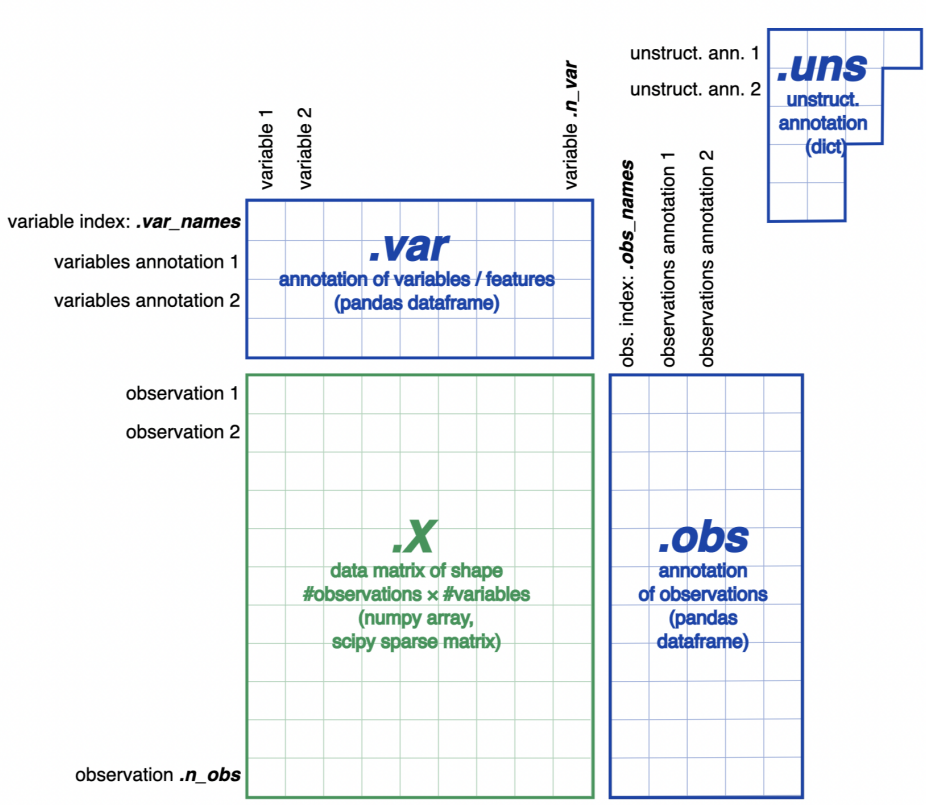

Figure 1.3: AnnData object in Python3

Source: Extracted from [5]

To work with scRNA-seq first we transform all our data into the former AnnData object. First, one has to transform adata_train.parquet file into a sparsed scipy matrix, the result is the following:

```
(240077, 21231)
  (0, 32)       1.0246189
  (0, 52)       1.0246189
  (0, 111)      1.0246189
  (0, 126)      1.0246189
  (0, 152)      1.5199653
  (0, 180)      1.0246189
  (0, 184)      1.0246189
  (0, 234)      1.0246189
  (0, 243)      1.0246189
  (0, 267)      4.093329
  (0, 308)      1.0246189
  (0, 314)      1.0246189
```

Figure 1.4: adata_train.parquet to sparsed scipy matrix

The first column contains a tuple (i, j) where i is the cell identifier and j the gene identifier, the second column is the raw count value. The last matrix will define our $AnnData.X$ component.

Once our AnnData object is ready, we can start inspecting the data. First, we will visualize how many compound replicates each cell has.

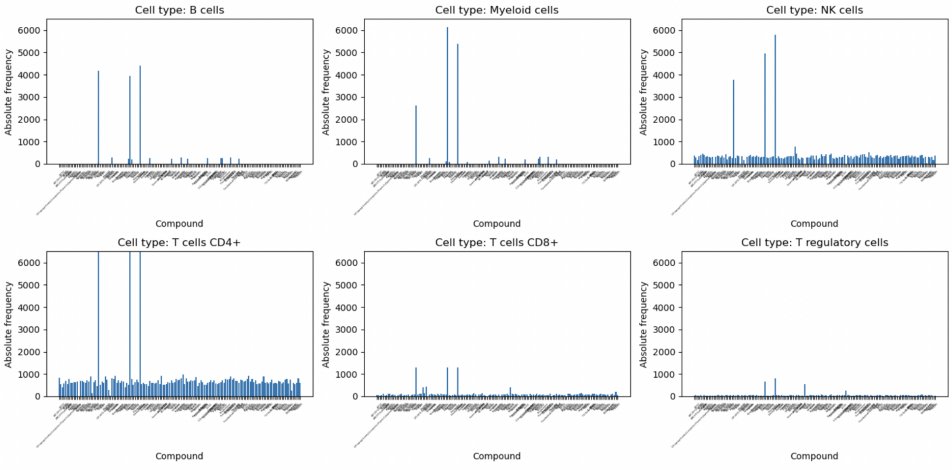

Figure 1.5: Number of replicates by compound for each cell

Replicates of each compound per cell differ. T cells CD4+ and NK cells show stable and high frequency for all compounds. However, B cells, Myeloid cells, T cells CD8+ and T regulatory cells, have less replicates though comparable between each cell type. Note that all cell types exhibit a high amount of replicates for three compounds, those are the control compounds (positive and negative) of: Dabrafenib, Belinostat and Dimethyl Sulfoxide, this is beneficial as we will need the negative control Dimethyl Sulfoxide to act as unperturbed state to predict the gene expression of other compounds for T regulatory cells.

The next step is on filtering and normalizing gene expression data. scGEN works best when using normalized counts and filtered genes/cells, hence we will need to perform such transformations on AnnData.X component. First, we filter by cells, removing cells that have at less 200 genes expressed, those cells express low variability and will not be of any much value, likewise we remove all genes that are expressed in less than 3 cells as those will not be of value either. Lastly, counts are normalized with scipy function `sc.pp.normalize_total` and a log transformationn is applied after with `sc.pp.log1p`.

The last step is on creating the training and validation sets. The training set will consist on the AnnData object built but removing compounds Resminostat and Cabozantinib for T regulatory cells, these will be the active compounds for which we will predict the gene expression for T regulatory cells. On the other hand, the validation set will consist on the AnnData object built containing the excluded data of the training set: expression for compounds Cabozantinib and Resminostat for T regulatory cells.

1.3 scGEN model training

In this section we will train scGEN on our training data and obtain prediction of gene expression for T-regulatory cells in compounds Cabozantinib and Resminostat.

To fix ideas, the main aim is to use a dataset with the RNA-seq profile of all cells and compounds, excluding compounds Cabozantinib and Resminostat for T regulatory cells, to estimate the latent space via VAE. Next, with the same dataset, we will predict the gene expression of T regulatory cells for compounds Cabozantinib and Resminostat based on the negative control compound Dimethyl Sulfoxide (DSMO).

scGEN needs a starting point, the unperturbed state, for which basing the prediction of the perturbed state. Since we have more than 70 compounds one has to select which is the compound that will act as an unperturbed state. Clearly, the most reasonable compound to use is one which is a negative control, like Dimethyl Sulfoxide, because the expression profile of cells under such compound will be their natural gene expression profile without any perturbation, since this compound does not alter the cell expression in any way. For this reason, Dimethyl Sulfoxide will be the unperturbed state, while the perturbed states will be either Cabozantinib or Resminostat (depending on the compound that we are predicting).

A github repository is available for scGEN with all functions ready-to-use in a Python3 library format,

see [6]. The first step is to download scgen library and import it in Python3 with `import scgen`, by that we upload all necessary functions for this section. Most of VAE hyperparameters are already set by default, like number of autoencoder layers or decoder layers. Fins below the instance to train the VAE.

```
# Model training
model.train(
    max_epochs=100,
    batch_size=32,
    early_stopping=True,
    early_stopping_patience=25
)
```

Figure 1.6: VAE train function

The training parameters set for the VAE are the following:

- `max_epochs`: the epoch is the number of times the dataset will go through the VAE to be trained. In our cased we have selected 100 epoch, that is the dataset will go through the VAE for, at maximum, 100 times.
- `batch_size`: most of the times we can not throw all our training data into the model at once, since the model could collapse. Otherwise, what we usually do is batch-training, so we train the model with a small subset of the data - that is the batch - and update the gradient for each batch. In our case, the batch size will be of 32.
- `early_stopping`: if the loss function does not improve substantially from one iteration to the next, we can presume a local/global minima is being attained, when that happens one needs many iterations to have a slight improvement in the loss function value, for such reason if this happens the training procedure will stop.
- `early_stopping_patience`: based on the previous criteria, if the loss function does not improve for at least 25 iterations, the algorithm will stop and an optimal value for the loss function has been attained.

```
Epoch 30/100:  30%|| | 30/100 [41:08<1:35:59, 82.28s/it, v_num=1, train_loss_step
Monitored metric elbo_validation did not improve in the last 25 records. Best score: 2690.823. Signaling
stop.
```

Figure 1.7: VAE trained

The attained convergence with 30 epochs, 41 minutes to train with each iteration lasting for 82.28 seconds. The loss function is the ELBO metric, as shown in the theoretical framework, attains its minimum at a value of 2690.823. With this we estimate the latent space representation of the data, such representation is very high-dimensional so it is not easily visualized. We use UMAP, a dimensionality reduction technique, to visualize the latent space representation for each cell type of our training data.

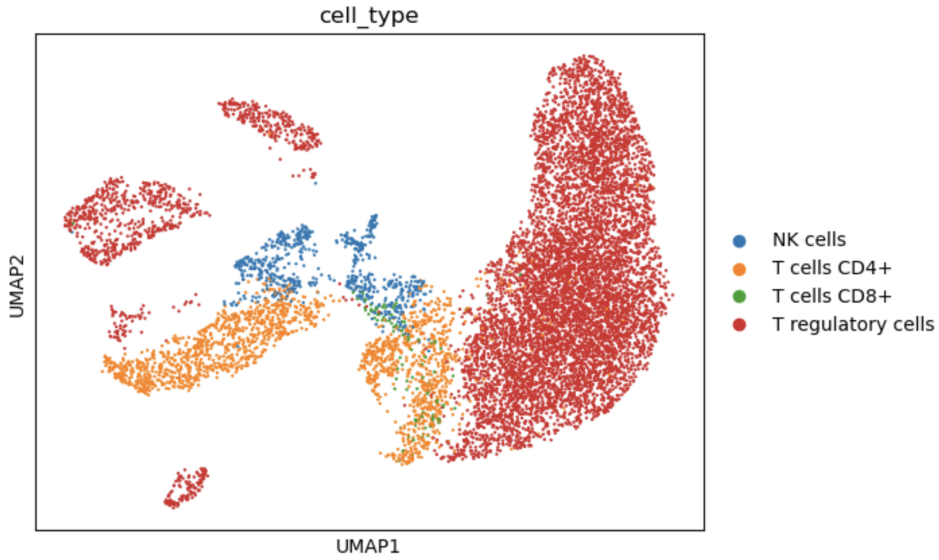

Figure 1.8: UMAP representation of the latent space by cell type

In the training set we have four different cell types: NK cells, T cells CD4+, T cells CD8,+ and T regulatory cells, notice that B and Myeloid cells were removed from the analysis in the cell/gene filtering step. From the UMAP, T regulatory cells and T CD4+ cells seem to be well differentiated in their expression profile, while more similarity exists in their expression profile between T CD8+ cells and NK cells, or T CD8+ cells and T CD4+ cells. there seems to be less variability for NK cells and T CD4+ cells, which is reasonable since we have observed a high number of replicates per compound for such cells. On the contrary, T regulatory cells appear to have more variability in their expression profile, which is also in line with the small amount of replicates per compound observed in the data preprocessing section. This UMAP is indistinguishable of each compound.

1.4 Prediction of compound gene expression and validation

We will predict gene expression profile for T regulatory cells for two active compounds: Resminostat and Cabozantinib. As stated in the last section, the unperturbed state will be gene expression profile under compound Dimethyl Sulfoxide, as such compound does not alter the gene expression profile of the cell and thus the measured expression is the natural state of the cell. For the prediction, we will use the method `model.predict`, where `model.` is the VAE trained in the last section.

```
# Predictions for Resminostat compound based on control Dimethyl Sulfoxide
pred, delta = model.predict(
    ctrl_key='Dimethyl Sulfoxide',
    stim_key='Resminostat',
    celltype_to_predict= 'T regulatory cells')

pred.obs['condition'] = 'Pred'

# Predictions for Cabozantinib based on control Dimethyl Sulfoxide
pred, delta = model.predict(
    ctrl_key='Dimethyl Sulfoxide',
    stim_key='Cabozantinib',
    celltype_to_predict= 'T regulatory cells'
)
pred.obs['condition'] = 'Pred'
```

Figure 1.9: Code instance to predict Cabozantinib and Resminostat gene expression

The method `model.predict` returns two objects: pred, delta. Pred will be an AnnData object that contains the gene expression estimation of the perturbation (Resminostat/Cabozantinib) along with the metadata, while delta will be the estimation of the mean gene expression difference between the perturbed and unperturbed state for all the cells considered in the analysis. Note that, in the two instances the control is set as 'Dimethyl Sulfoxide' as that is our unperturbed state, whereas the perturbed state is `stim_key = Resminostat` or `stim_key = Cabozantinib`. The prediction is based on all T-regulatory cells in the training set with a compound of Dimethyl Sulfoxide, for such instances a prediction of their gene expression will be estimated for each compound Resminostat and Cabozantinib.

`Pred.X` contains the estimation of the perturbation with the model scGEN, what the `.pred` instance does is predict the perturbed gene expression based on the VAE trained and the estimation of δ, so this is not an out-of-bag prediction of the VAE model but a combination of the latent space representation and the high-dimensional vector arithmetics with δ. To validate this prediction, some rearragentments with the data has to be performed such: concatenating in a single AnnData object the pred object with the AnnData validation object. The validation dataset consists on the groundtruth gene expression for T regulatory cells in compounds Resminostat and Cabozantinib.

First, we will visualize in a PCA the gene expression profile by each predicted compound.

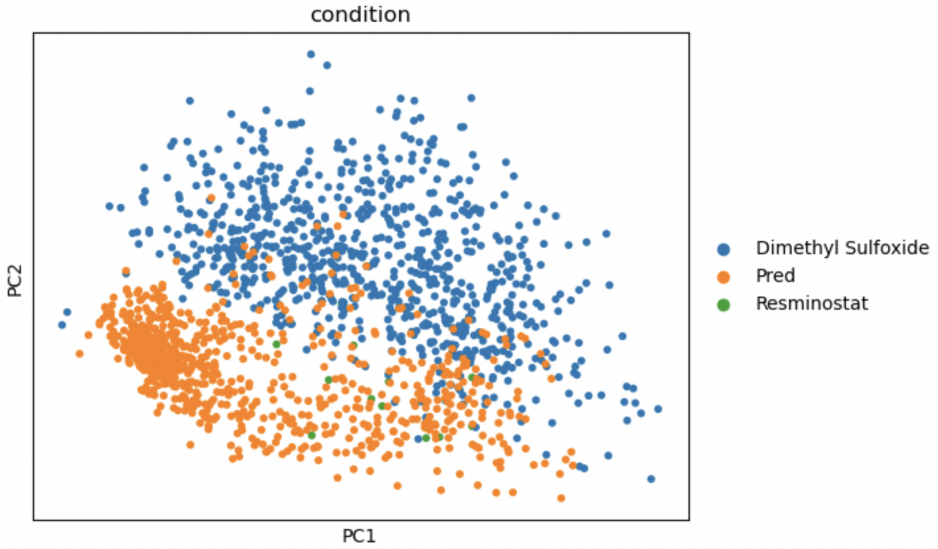

Figure 1.10: PCA of gene expression by Resminostat, Prediction of Resminostat and Dimethyl Sulfoxide

The second principal component is able to distinguish between the negative control Dimethyl Sulfoxide with the prediction of Resminostat, there seems to be a clear distinction between them which is a good sign as we know the cell expression profile under these two compounds has to be different. On the other hand, the groundtruth expression profile Resminostat (green) is quite on par with the prediction of Resminostat, as it could be even clustered together visually. All in all, the expression profile predicted seems reasonable and on par with what would be expected. Note that the groundtruth is reduced compared to the number of predictions in Resminostat, we will compare both distributions and make sure the is an agreement between the prediction and the groundtruth.

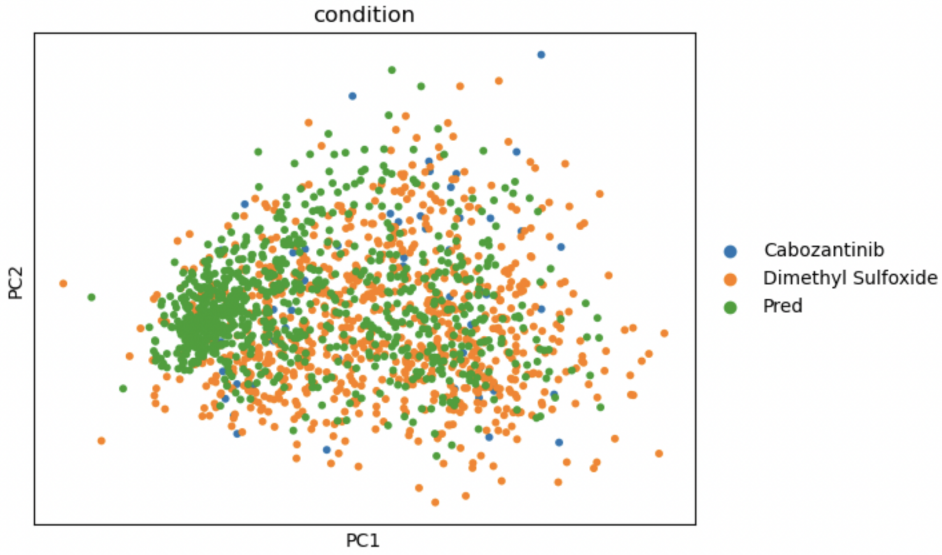

Figure 1.11: PCA of gene expression by Cabozantinib, Prediction of Cabozantinib and Dimethyl Sulfoxide

Unlike the Resminostat PCA, Cabozantinib PCA more blurry differences in gene expression profile between Dimethyl Sulfoxide and predicted Cabozantinib. The data seems, overall, to have more variability in all its conditions: Cabozantinib, Predicted Cabozantinib, Dymethil Sulfoxide. No clear clustering can be inferred from the principal components. Bear in mind, that PCA is a dimensionality reduction technique that only accounts for linear dependencies, unlike UMAP which accounts for non linear dependencies in the high-dimensional space, thus it is possible that based on a linear dimensionality reduction it is not possible to distinguish between the three conditions due to non-linear dependencies between the conditions.

To validate the results, we will follow a similar approach as in other references ([2], [3]), by correlating the prediction vs the groundtruth. Bear in mind, that correlation of the prediction with the groundtruth is a necessary condition for the prediction to be aligned with the groundtruth but its not a sufficient condition.

- r_{xy} pearson correlation will measure how the prediction and the groundtruth variability align, the higher the correlation is to 1 the better the prediction of the model. This is a necessary condition for the prediction to be well aligned.
- $y = x$ even though correlation is necessary it is not sufficient for a prediction to be well aligned. For instance, a correlation between a set of points $\{y_i\}_i$ and $\{x_i\}_i$ can be high but that does not mean $x_i = y_i$. If the adjusted regression line between the prediction and the groundtruth is not alike the identity $y = x$, it will mean the prediction is biased.

All in all, to assess for the validity of the prediction a close look into the variability and biais of the prediction will be deemed. To fix ideas, we have generated predictions for T regulatory cells and compounds Cabozantinib and Reminostat, each cell has hundreds or thousands of genes for which their gene expression has been predicted, we will compare the gene expression of each gene by each compound. A self-made function has been programed to this task, a linear regression is estimated and the correlation coefficient of the gene expression for all genes.

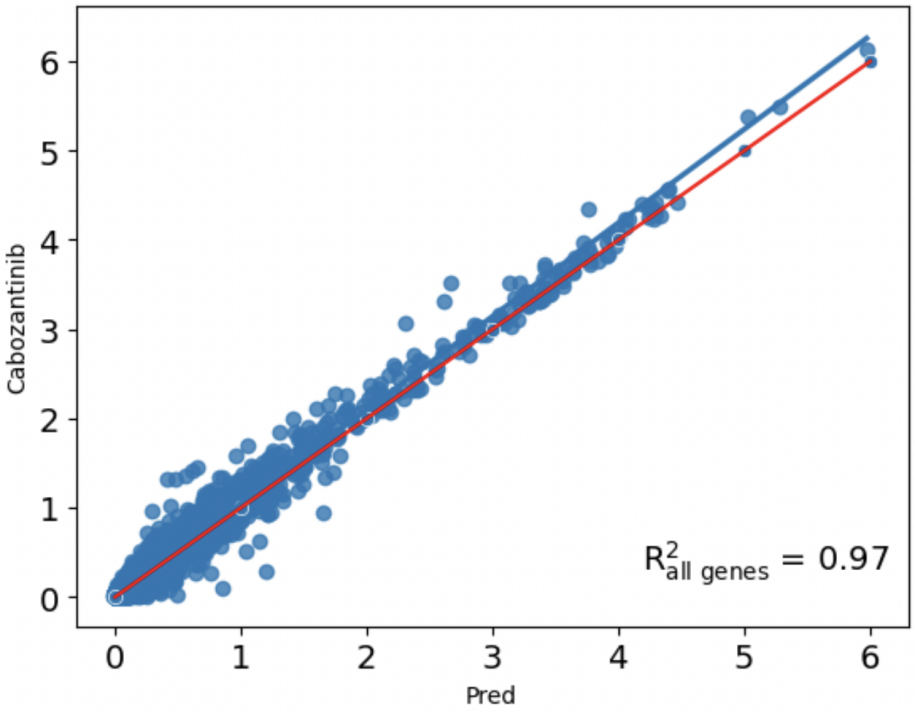

Figure 1.12: Validation plot for Cabozantinib prediction

X axis contains the values of the predicted gene expression of Cabozantinib for each gene, while Y axis contains the groundtruth of Cabozantinib gene expression for each gene. The red line indicates the y=x identity line. The curve fit slightly differs from the identity line y=x, this indicates the prediction has some bias and that this bias is on underestimating the gene expression. We know the bias produces, on average, underestimation of the gene expression as the fit curve lies above y=x axis. On the other hand, the pearson correlation coefficient of the gene expression for all genes is very high $R^2 = 0.97$, indicating that scGEN is able to correctly capture the variability of the expression profile for Cabozantinib. Overall, the predictions are very much alike the groundtruth and the quality of the prediction for this compound is not much different from the groundtruth.

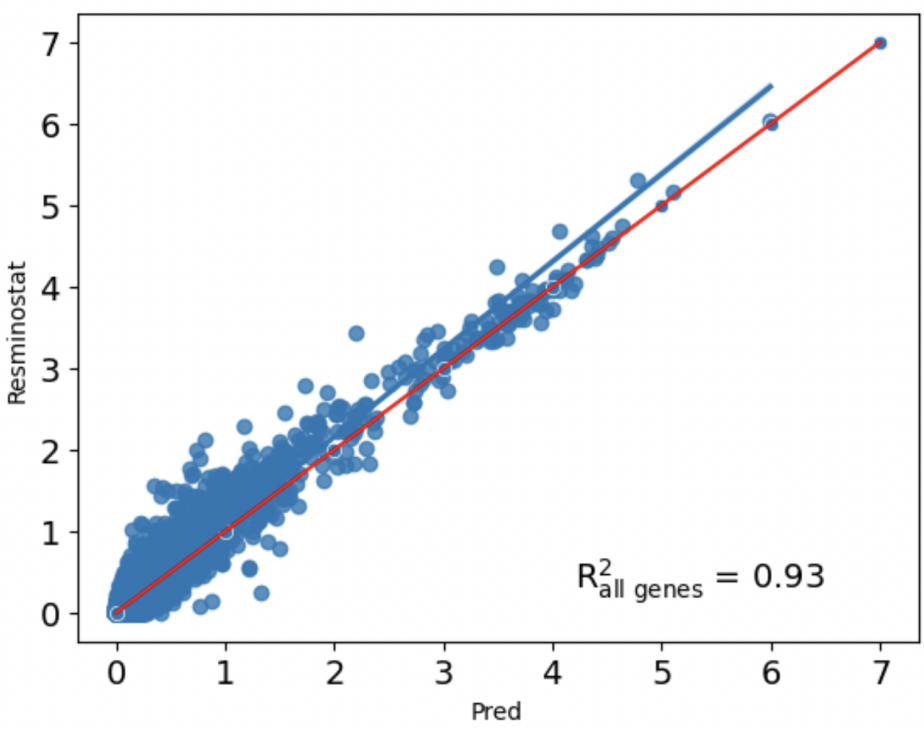

Figure 1.13: Validation plot for Resminostat prediction

While the bias is present alike with Cabozantinib, the bias seems to be higher as the gene expression increases, that is when gene expression is low the bias is less present contrary to higher gene expression profiles beyond 4 normalized counts. The pearson correlation coefficient is lower than Cabozantinib and $R^2 = 0.93$, although it is still a high correlation coefficient for the prediction. Even though there is bias present, the predictions are of high quality and could be trusted for other scientific tasks. In both cases, Cabozantinib and Resminostat, on average there is an underestimation of the gene expression profile that increases with the number of normalized counts, that is those genes with true high normalized counts can expect to be predicted with lower normalized counts with scGEN.

Chapter 2

Conclusion and further research

How cells molecularly react to stimuli is a key question in molecular biology, be its molecular reaction a: transcription, metabolism and gene regulation profile. In this thesis, we have focused on how cells transcriptomically react (molecular reaction) to chemical compounds (stimuli) by means of synthetic data generation under such perturbations. We aimed at predicting the gene expression profile of compounds Cabozantinib and Resminostat for T regulatory cells, based on scGEN and the expression profile of T regulatory cells under their unperturbed transcriptomic profile - that is Dymethil Sulfoxide compound -.

A pipeline in Python3 for this analysis has been created and it is accessible in a github respository https://github.com/amoruno/Master-s-thesis. We trained scGEN, the gold-standard method for single-cell perturbation analysis, with data from three donors of PBMCs scRNA-seq measured for more than 140 chemical compounds, we removed from the training data all compounds of Cabozantinib and Resminostat for T regulatory cells. Notice that by removing such piece of data, the training data does not contain any RNA-seq measurement from what we want to predict, thus being able to generate data in silico on non observed cases in the training set.

In silico RNA-seq of Cabozantinib and Resminostat for T regulatory cells was generated from scGEN. Combining VAEs and latent space vector arithmetics, scGEN is able to generate de novo gene expression data on compounds and cells not present in the training set.

A validation of the results, by means of linear regression and bias towards the identity line, was performed compared to the groundtruth. Validation tools demonstrate that the gene expression prediction for both compounds are highly accurate, with pearson correlation coefficient of $R^2 = 0.97$ and $R^2 = 0.93$ for Cabozantinib and Resminostat, respectively. For both compounds, a slight negative bias was observed in predictions for genes with a high number of normalized counts, this bias was lesser for Resminostat than Cabozantinib. These results are in line with the predictive power of scGEN shown in literature. Further, it is more strongly validated since the number of replicates between cell types is highly unbalanced in our experiment, with our cell target T regulatory cells having much less replicates per compound than other cell types like NK cells or

We hypothesize that the negative bias observed for genes with high normalized counts, could be due to the small number of genes with high normalized counts which could influence the fit of the regression curve. We acknowledge that, for the sake of resources, more validation tools should be explored to assess for the validation of the prediction: correlation structure, sparsity and dimensionality.

scGEN has proven a strong computational tool to generate in silico scRNA-seq, while reusage of biological data is increasingly becoming of fundamental importance in research. An application on scGEN about in silico scRNA-seq generation between species ([2]), is shown as a promising tool on generating in silico scRNA-seq from one animal specie to another. One of the current trends in preclinical research is on reducing the burden on animal drug assessment, by reducing animal testing and accerelating preclinical research. Thus we envision a further research line on validating scGEN for in silico scRNA-seq data generation between animal species a promising research line with potential in radically changing preclinical research, making drug development faster and more efficient.

Bibliography

[1] Daniel Burkhardt, Andrew Benz, Richard Lieberman, Scott Gigante, Ashley Chow, Ryan Holbrook, Robrecht Cannoodt, Malte Luecken. (2023). *Open Problems Single Cell Perturbations*. Kaggle. https://kaggle.com/competitions/open-problems-single-cell-perturbations

[2] Mohammad Lotfollahi, F. Alexander Wolf and Fabian J. Theis. (2019). *scGen predicts single-cell perturbation responses*. Nature Methods. https://www.nature.com/articles/s41592-019-0494-8

[3] Leon Hetzel, Simon Boehm, Niki Kilbertus, Stephan Günnemann, Mohammad Lotfollahi, Fabian J Theis. (2023). *Predicting Cellular Responses to Novel Drug Perturbations at a Single-Cell Resolution*. NeurIPS. https://openreview.net/forum?id=vRrFVHxFiXJ

[4] Omar Kana, Rance Nault, David Filipovic, Daniel Marri, Tim Zacharewski and Sudin Bhattacharya. (2023). *Generative modeling of single-cell gene expression for dose-dependent chemical perturbations*. Patterns (N Y). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10436058/

[5] Vladimir Kiselev, Tallulah Andrews, Jennifer Westoby, Davis McCarthy, Maren Büttner, Jimmy Lee, Krzysztof Polanski, Sebastian Y. Müller, Elo Madissoon, Stephane Ballereau, Maria Do Nascimento Lopes Primo, Rocio Martinez Nunez and Martin Hemberg. *Analysis of single cell RNA-seq data*. https://chanzuckerberg.github.io/scRNA-python-workshop/preprocessing/00-tabula-muris.html

[6] Mohammad Lotfollahi. *scgen-reproducibility*. https://github.com/theislab/scgen-reproducibility/tree/master

[7] Isaac Virshup, Gökcen Eraslan, et al. *Scanpy – Single-Cell Analysis in Python*. https://scanpy.readthedocs.io/en/stable/

[8] Priya Ranganathan, C.S. Pramesh and Rakesh aggarwal. *Common pitfalls in statistical analysis*. Perspectives of Clinical Research. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5654219/

[9] Van Rossum, G., Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

Appendix

Python code

```
import pandas as pd
import anndata as ad
import numpy as np

# Read gene expression data in parquet format
df_url = "/Users/aitormc/adata_train.parquet"
df = pd.read_parquet(df_url)

from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix

# Sample data with categorical labels
row_labels = df['obs_id'].values
col_labels = df['gene'].values
values = df['count']

# Create mappings from categorical labels to numerical indices
row_label_to_index = {label: index for index, label in enumerate(set(
    row_labels))}
col_label_to_index = {label: index for index, label in enumerate(set(
    col_labels))}

# Map categorical labels to numerical indices and create triplets
row_indices = [row_label_to_index[label] for label in row_labels]
col_indices = [col_label_to_index[label] for label in col_labels]

# Create a sparse matrix from the triplets
sparse_matrix = coo_matrix((values, (row_indices, col_indices)))

sparse_matrix = sparse_matrix.tocsr()

# Export to npz
from scipy.sparse import coo_matrix, save_npz

filename = "rna_seq_counts_sparse_matrix.npz"
save_npz(filename, sparse_matrix)

# Export observation id to csv
sr = pd.Series(row_label_to_index)
sr.to_csv('rna_seq_obs_id.csv')

# Export gene id to csv
sr = pd.Series(col_label_to_index)
sr.to_csv('rna_seq_gene_symbols.csv')
```

```
# Load sparse matrix
from scipy.sparse import load_npz
fn = '/Users/aitormc/rna_seq_counts_sparse_matrix.npz'
X = load_npz(fn)

# Create an AnnData object with the sparse matrix
import scanpy as sc

adata = sc.AnnData(X=X)

# Load metadata of obs id
import pandas as pd

fn = '/Users/aitormc/rna_seq_obs_id_symbol.csv'
df_obs = pd.read_csv(fn,index_col = 0)
df_obs.columns = ['index cell']
df_obs['obs_id'] = df_obs.index
print(df_obs)

# Load metadata of gene id
fn = '/Users/aitormc/rna_seq_gene_symbol.csv'
df_var = pd.read_csv(fn,index_col = 0)
print(df_var.shape)
df_var.columns = ['index gene']
df_var

# Load metadata of obs
fn = "/Users/aitormc/adata_obs_meta.csv"
adata_metadata = pd.read_csv(fn)
print(adata_metadata)

# Merge metadata with obs id
df_obs_ = df_obs.merge(adata_metadata, on='obs_id', how='left')

# Add obs and col metadata to AnnData
adata.obs = df_obs_
adata.var = df_var

# Filtering of genes/cells and normalization counts
print(adata.shape)
sc.pp.filter_cells(adata, min_genes=200)
sc.pp.filter_genes(adata, min_cells=3)
print(adata.shape)

sc.pp.normalize_total(adata, target_sum=1e4)
sc.pp.log1p(adata)
print(adata.X)

import pandas as pd
import matplotlib.pyplot as plt

occurrences = adata.obs.groupby(['cell_type', 'sm_name']).size().unstack
    ().fillna(0)

# Calculate the number of rows and columns for the grid
num_rows = 3
num_columns = 3
```

```
# Get the list of unique cells
unique_cells = occurrences.index.unique()

# Create a grid of subplots with the desired number of rows and columns
fig, axs = plt.subplots(num_rows, num_columns, figsize=(15, 10))

# Iterate through cells and generate a bar plot for each one
for i, cell in enumerate(unique_cells):
    row = i // num_columns
    col = i % num_columns

    ax = axs[row, col]
    ax.bar(occurrences.columns, occurrences.loc[cell])
    ax.set_title(f'Cell type: {cell}')
    ax.set_xlabel('Compound')
    ax.set_ylabel('Absolute frequency')

    # Adjust the font size of the X-axis
    ax.tick_params(axis='x', labelsize=2)

    ax.set_xticklabels(occurrences.columns, rotation=45)
    ax.set_ylim(0, 6500)

# Remove unused subplots if there are fewer than 9 cells
for i in range(len(unique_cells), num_rows * num_columns):
    fig.delaxes(axs[i // num_columns, i % num_columns])

plt.tight_layout()
plt.show()

# Create training and validation datasets

# Validation
is_t_regulatory = adata.obs["cell_type"] == "T regulatory cells"
is_resminostat_or_cabozantinib = adata.obs["sm_name"].isin(["Resminostat
   ", "Cabozantinib"])

validation = adata[is_t_regulatory & is_resminostat_or_cabozantinib]

# Training
# Remove T regulatory cells for compounds Resminostat and
training = adata[~((adata.obs["cell_type"] != "T regulatory cells") &
                   (adata.obs["sm_name"] != "Resminostat") & (adata.obs
                      ["sm_name"] != "Cabozantinib"))]

train_new = training.copy()

# Install scGEN and associated requirements
import sys
#if branch is stable, will install via pypi, else will install from
   source
branch = "stable"
IN_COLAB = "google.colab" in sys.modules

if IN_COLAB and branch == "stable":
    !pip install --quiet scgen[tutorials]
elif IN_COLAB and branch != "stable":
    !pip install --quiet --upgrade jsonschema
```

```
    !pip install --quiet git+https://github.com/theislab/scgen@$branch#
       egg=scgen[tutorials]

# Install git
!apt-get install -y git

# Clone scgen repo
!git clone https://github.com/theislab/scgen.git

pip install scgen[tutorials]

pip install git+https://github.com/theislab/scgen.git

import logging
import scanpy as sc
import scgen

scgen.SCGEN.setup_anndata(train_new, batch_key="sm_name", labels_key="
   cell_type")

# Model save
model = scgen.SCGEN(train_new)
model.save("/Users/aitormc/model_perturbation_prediction.pt", overwrite=
   True)

# Model training
model.train(
    max_epochs=100,
    batch_size=32,
    early_stopping=True,
    early_stopping_patience=25
)

# Generate the latent space
latent_X = model.get_latent_representation()
latent_adata = sc.AnnData(X=latent_X, obs=train_new.obs.copy())

# Representation of the latent space
sc.pp.neighbors(latent_adata, use_rep = 'X')
sc.tl.umap(latent_adata)
sc.pl.umap(latent_adata, color=['cell_type'], wspace=0.4, frameon=True)

# Predictions for Resminostat compound based on control Dimethyl
   Sulfoxide
pred, delta = model.predict(
    ctrl_key='Dimethyl Sulfoxide',
    stim_key='Resminostat',
    celltype_to_predict= 'T regulatory cells')

pred.obs['condition'] = 'Pred'

# PCA
sc.tl.pca(eval_adata)
sc.pl.pca(eval_adata, color="condition", frameon=True)

validation_copy = validation.copy()
stim_adata = validation_copy[((validation_copy.obs['cell_type'] == 'T
   regulatory cells') & (validation_copy.obs['sm_name'] == 'Resminostat'
```

```
    ))]
ctrl_adata = train_new[((train_new.obs['cell_type'] == 'T regulatory
    cells') & (train_new.obs['sm_name'] == 'Dimethyl Sulfoxide'))]
stim_adata.obs['condition'] = 'Resminostat'
ctrl_adata.obs['condition'] = 'Dimethyl Sulfoxide'

# Remove model training variables created in AnnData
del ctrl_adata.obs['_scvi_batch']
del ctrl_adata.obs['_scvi_labels']
del pred.obs['_scvi_batch']
del pred.obs['_scvi_labels']

import anndata as ad
from scipy.sparse import csr_matrix
pred.X = csr_matrix(pred.X)
eval_adata = ad.concat([stim_adata, ctrl_adata, pred])

# Validation of prediction Resminostat
axis_keys = {
    'x' : 'Pred',
    'y' : 'Resminostat'
}

from scipy import stats, sparse
import numpy as np

stim = eval_adata[eval_adata.obs['condition'] == axis_keys['x']]
ctrl = eval_adata[eval_adata.obs['condition'] == axis_keys['y']]

x = np.average(stim.X.A, axis=0)
y = np.average(ctrl.X.A, axis = 0)

m, b, r_value, p_value, std_err = stats.linregress(x,y)
import seaborn as sns
df = pd.DataFrame({axis_keys["x"]: x, axis_keys["y"]: y})
ax = sns.regplot(x=axis_keys["x"], y=axis_keys["y"], data=df,
    scatter_kws={'rasterized' : True})
ax.tick_params(labelsize=14)

x_coeff = 0.3
y_coeff = 0.8
ax.text(max(x) - max(x) * x_coeff, max(y) - (y_coeff + 0.15) * max(y), r
    '$\mathrm{R^2_{\mathrm{\mathsf{all\ genes}}}}$ = ' + f"{r_value **
    2:.2f}", fontsize=14)

# Sample data
x = list(range(8))
y = list(range(8))

# Create a scatter plot
sns.scatterplot(x=x, y=y)

# Add a diagonal line
plt.plot([min(x), max(x)], [min(y), max(y)], color='red')

# Show the plot
plt.show()
```

```
# Predictions for Cabozantinib based on control Dimethyl Sulfoxide
pred, delta = model.predict(
    ctrl_key='Dimethyl Sulfoxide',
    stim_key='Cabozantinib',
    celltype_to_predict= 'T regulatory cells'
)
pred.obs['condition'] = 'Pred'

validation_copy = validation.copy()
stim_adata = validation_copy[((validation_copy.obs['cell_type'] == 'T
   regulatory cells') & (validation_copy.obs['sm_name'] == 'Cabozantinib
   '))]
ctrl_adata = train_new[((train_new.obs['cell_type'] == 'T regulatory
   cells') & (train_new.obs['sm_name'] == 'Dimethyl Sulfoxide'))]
stim_adata.obs['condition'] = 'Cabozantinib'
ctrl_adata.obs['condition'] = 'Dimethyl Sulfoxide'

# Remove model training variables created in AnnData
del ctrl_adata.obs['_scvi_batch']
del ctrl_adata.obs['_scvi_labels']
del pred.obs['_scvi_batch']
del pred.obs['_scvi_labels']

import anndata as ad
from scipy.sparse import csr_matrix
pred.X = csr_matrix(pred.X)
eval_adata = ad.concat([stim_adata, ctrl_adata, pred])

# Validation of predictions of Cabozantinib
axis_keys = {
    'x' : 'Pred',
    'y' : 'Cabozantinib'
}

from scipy import stats, sparse

stim = eval_adata[eval_adata.obs['condition'] == axis_keys['x']]
ctrl = eval_adata[eval_adata.obs['condition'] == axis_keys['y']]

x = np.average(stim.X.A, axis=0)
y = np.average(ctrl.X.A, axis = 0)

m, b, r_value, p_value, std_err = stats.linregress(x,y)
import seaborn as sns
df = pd.DataFrame({axis_keys["x"]: x, axis_keys["y"]: y})
ax = sns.regplot(x=axis_keys["x"], y=axis_keys["y"], data=df,
   scatter_kws={'rasterized' : True})
ax.tick_params(labelsize=14)

x_coeff = 0.3
y_coeff = 0.8
ax.text(max(x) - max(x) * x_coeff, max(y) - (y_coeff + 0.15) * max(y), r
   '$\mathrm{R^2_{\mathrm{\mathsf{all\ genes}}}}$ = ' + f"{r_value **
   2:.2f}", fontsize=14)

# Sample data
x = list(range(7))
y = list(range(7))
```

```
# Create a scatter plot
sns.scatterplot(x=x, y=y)

# Add a diagonal line
plt.plot([min(x), max(x)], [min(y), max(y)], color='red')

# Show the plot
plt.show()

# PCA
sc.tl.pca(eval_adata)
sc.pl.pca(eval_adata, color="condition", frameon=True)
```