scRNA denoising research

2024

1 Gene Distribution

All of the content below can be seen in mat_distribution.ipynb

1.1 Mean and Variance

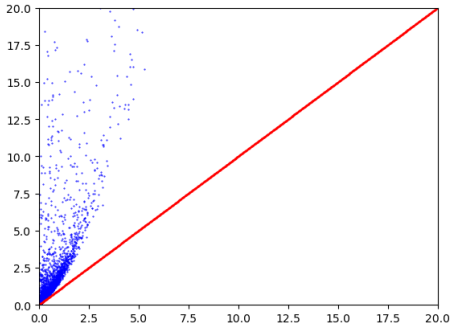


Figure 1: mean and variance distribution of gene

It's a graph showing the mean (x-axis) and variance (y-axis) of every genes. Attention that the red line is the μ and σ^2 of Poisson Distribution.

As it shows in this graph, we can find that our μ and σ^2 don't fit the Poisson Distribution. Also, the gap between our real condition and the ideal condition of Poisson Distribution will increase wider as the μ grows.

These features fit the condition of Negative Binomial Distribution more as it's μ and σ^2 fit the quadratic function. Also, we can consider the gene amount is like a process we're testing to find the experiment amount while the real gene amount is the success amount and p is the probability that the gene will be expressed. So given the matrix full of the success amount of gene, we can use the NB distribution to fit the condition.

1.2 Some Other Suspection

We can get some other suspection about the Poisson distribution, such as zero inflated Poisson distribution, because if we add some excessive zeros (from our original data),we can also get the variance higher than the mean.

But as the graph shows, our ZIP (zero inflated Poisson distribution) will show a μ and σ^2 curve that is convex function at first and become concave function then but our real condition and NB distribution condition actually have the concave function.

The graph below shows the poisson distribution with 1/6 data = 0 and can't detect gene amount less than 5.

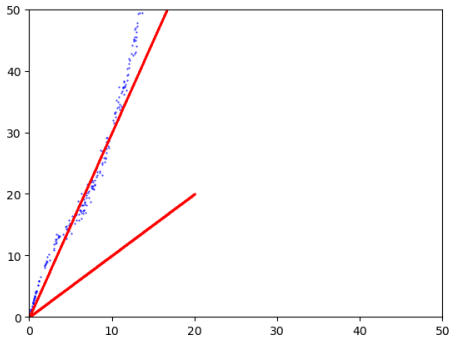


Figure 2: Zero Inflated Poisson

1.3 Ideal NB Distribution

Here's an example about the ideal NB distribution

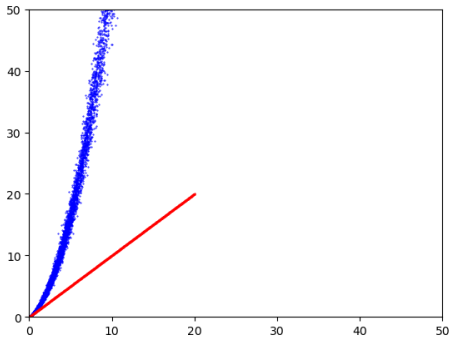


Figure 3: Ideal NB distribution

1.4 Distribution of Single Gene

We use the distribution of gene1 to describe this part.

The condition of most genes just have similar condition with gene1. They oftenly have many 0s in the column, and have less than 0.5 percent data lying in the region larger or equal to 3.

t

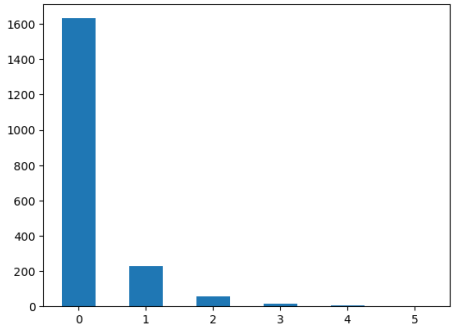


Figure 4: Gene 1 Distribution

Actually, most of gene has the sum less than 0.1 * their length (around 60 %), so most genes having more than 90 % 0s in their probability model.

However, there still exists around 4% genes having total expressing amount more than 1937 (cell amount). I temporarily set these genes as the high expression gene. I deem that these gene will be more important part to determine the features of different cell

The following graph show a high expression gene distribution (red) with its standard NB distribution (black).

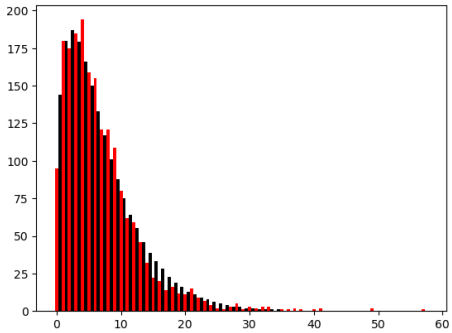


Figure 5: high expression gene and corresponding NB(Gene 13869)

Here's a more obvious graph to see the whole matrix. We can find that there're some genes that have really high expression amount

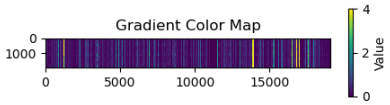


Figure 6: inDrop1 distribution

1.5 Zero-Inflated Negative Distribution

From the experience about the ZIP, also due to the condition in reality that we have an enourmous amount of 0s in our gene distribution. Maybe applying our model to ZINB (Zero Inflated Negative Binomial) model can be more proper. The following graph shows the fit result of Gene13 using the ZINB model and NB model.

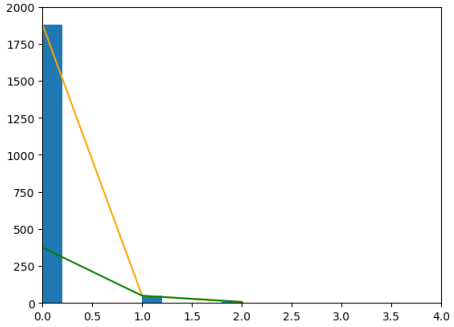


Figure 7: ZINB fit result (Gene13)

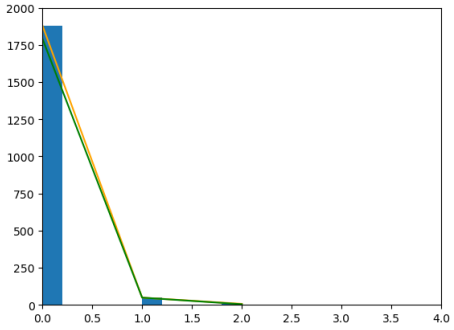


Figure 8: NB fit result (Gene13)

Because the 0s occupy too much proportion of the data and we don't need to concern about the probability fitting result of 0s (ZINB has 2 part: NB and inflated zeros, the inflated zeros can make sure the ZINB has the correct amount of 0), we choose to enlarge the part that larger than 0.

The following 2 pictures is the enlarged version

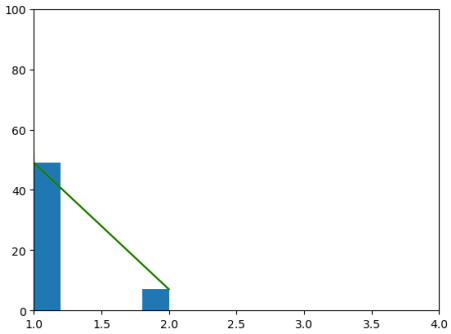


Figure 9: fit result of ZINB larger than 0 (Gene 13)

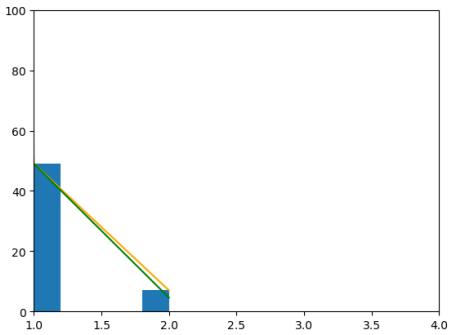


Figure 10: fit result of NB larger than 0 (Gene 13)

The orange line means the real condition and the green line is the fit result.

We can find that the ZINB model fit the real condition better (you may not see the orange line in Figure9 because the 2 lines just overlap together). It's a ZINB distribution with zero inflation rate 0.79.

1.5.1 Zero Inflation Rate Calculation

The zero inflation rate calculated with $get\_zeroinflated()$ function in my github page (). I use the number non-zero to get the parameter of ZINB distribution. To be specific, for a ZINB distribution, The appearance of number != 1 will follow the NB distribution. Here we choose the amount of number 1 and 2

here to get our NB distribution. And then we can get zero inflation rate via the difference between the zero appearance rate of NB distribution and real distribution.
However, in this project, we have the scRNA data to be analyzed, which is very discrete, so actually most gene only has 0, 1 and 2 amount of gene expression (mostly zero). So we only choose the appearance amount of number 1 and 2 to estimate the maximum likelihood parameter for the Zero-Inflated Negative Binomial distribution.

1.5.2 Function Implementation of ZINB

ZINB has 2 parts of distribution:

1. z probablity full of zero
2. $(1-z)$ probability NB distribution NB(r,p)

r,p parameter for the negative binomial distribution:

$$r = \frac{\mu^2}{\sigma^2 - \mu} \qquad p = \frac{r}{r+\mu}$$

We can calculate μ and σ from sample data after throwing z (zero inflation rate) zeroes.

And we can get the probability of appearance of 1,2 from the pmf of NB distribution:

$$P(X = k) = \binom{k+r-1}{r-1} p^r (1-p)^k$$

So the probability of x=1 ($p1$) and x=2 ($p2$) is:

$$P(X = 1) = r * p^r (1-p)$$

$$P(X = 2) = ((r+1) * (r-1)/2) * p^r (1-p)^2$$

we can use the probability of x=1 and x=2 in the real data ($p1'$ and $p2'$) to calculate $p1'/p2'$. Try to find the best z, which will make $p1'/p2'$ closest to $p1/p2$.

In the project, I try z with gap 0.01, which is also the precision.

1.5.3 Zero-inflation Rate Result

There're 19093 genes in total and for 13695 genes, there's at least 1 cell having this gene expression more than 1 (the other 5398 genes is shown as -1 in the first graph). Detail file is uploaded on Github()

The zero-inflation rate data is shown below:

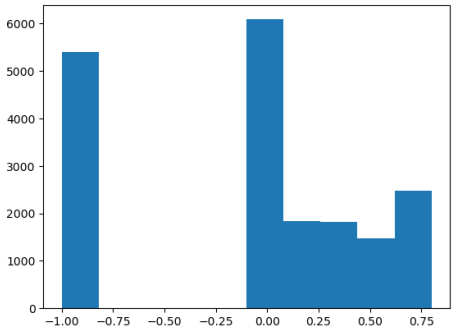


Figure 11: Zero-inflation Rate distribution in total

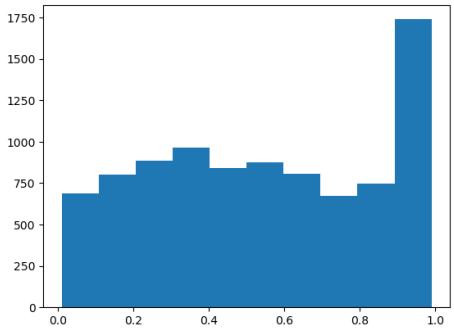


Figure 12: Zero-inflation Rate distribution larger than 0

From the graph, we can see that most genes expressing as the NB distribution more, and for the potential ZINB distribution genes here, the zero-inflation rate is distributed very average.

1.6 Conclusion of Gene Distribution

If we look along the column, most genes with various expression amount will behave as the negative binomial distribution and there're also some genes more likely to be zero-inflated negative binomial distribution.
However, given that the zero inflation rate isn't concentrated and the genes likely to be ZINB distribution don't occupy most of the total genes. So, In the further analysis of our scRNA data, It's better to choose negative binomial distribution as more proper probability model.
(temporary conclusion)

2 Lower Dimension

2.1 PCA

PCA is a linear lower dimension technology. Though it's not so fit for our scRNA data because the dimension is too high and it is probably the non-linear construction in high-dimension space, we can use PCA to do the primary processing of data.(See in tSNE)
Using SVD to calculate PCA will increase the speed much, especially for the large matrix data.

```
Define function pca_svd(matrix, n_components):

    get the centered_matrix using mean along col

    Perform SVD on centered_matrix to get U, S, and Vt

    # Select the first n_components columns of U and rows of Vt
    U_reduced,S_reduced,Vt_reduced = Select first n_components
    columns of U,S,Vt

    # Calculate the reduced matrix
    reduced_matrix = Dot Product of U_reduced and diagonal
    matrix of S_reduced
    return reduced_matrix
```

There might be some difference if you skip the first processing, but both methods have some right. In this scRNA project, choosing the centered matrix will get a better behavior in the lower dimension visualization.

2.2 tSNE

Different from the PCA method, tSNE is a nonlinear manifold analysis. It uses the comparative distance relations in original dimensions to rebuild the position of cell in lower dimension instead of using the distance as a comparative parameter directly.
This method will be better for the high dimension data compared with PCA method because in the high dimension space. For tSNE, it firstly uses the Gaussian distribution in high dimension data to get the conditional probability matrix. And then it uses student-t distribution to reproduce the the similar conditional probability in the lower dimension matrix. So that the lower dimension matrix we've got will have the similar conditional probability between each points(rows) compared with high dimension matrix.
tSNE calculated the conditional probability of point i and j in high dimension data using the following function(σ is a bandwitdth parameter to control the

range of neighbors):

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}$$

It assumes the Gaussian distribution for high dimension data but it uses the student-t kernel function assumption in lower dimension because it has a heavier tail and better robustness compared with Gaussian kernel. There'll be the curse of dimensionality problem for the period transferring high dimension data to low dimension data, that is much data point will be gathered near the bound and the distance between points will be tended to same, so we need to choose distribution with a heavier tail to make the result more reliable, and t-student distribution is good choice.
Here's the conditional probability in low dimension based on student-t distribution:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

In fact, we'll use the PCA to lower the dimension of matrix firstly before the tSNE processing. It can :

1. reduce the calculation of tSNE algorithm
2. help to denoise the data firstly because PCA can wipe out some noise data (they usually don't occupy the principle component)

In my scRNA project, tSNE performs best in the visualization.(See in the visualization.ipynb)

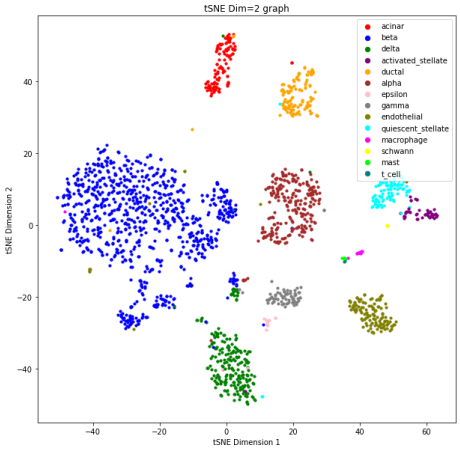
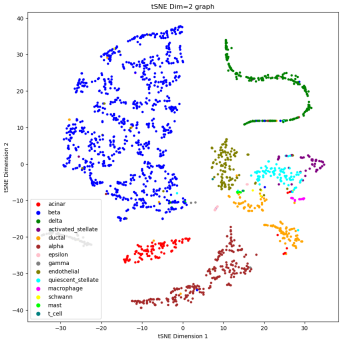
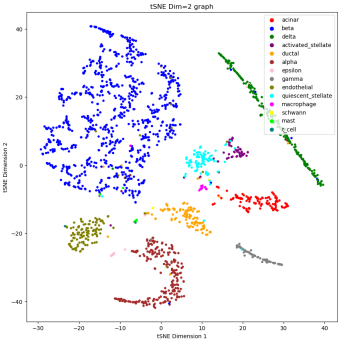


Figure 13: tSNE 2-D Visualization Result (Standard)

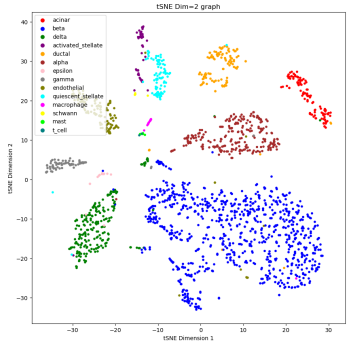
There were some interesting phenomenon having having happened as I apply the tSNE to the scRNA matrix.
(1.Every visualization results shown below having been applied with best perplexity choice.
2.Every colored points below are in real conditions
3.cp10k means normalize the matrix m to let $\Sigma(m[i]) = 10k, \forall i$)



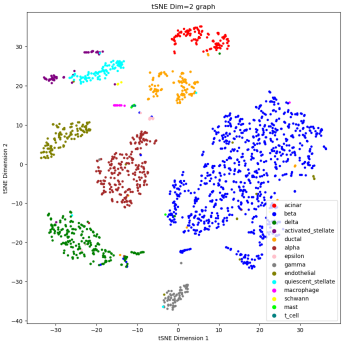
(a) original data tSNE result



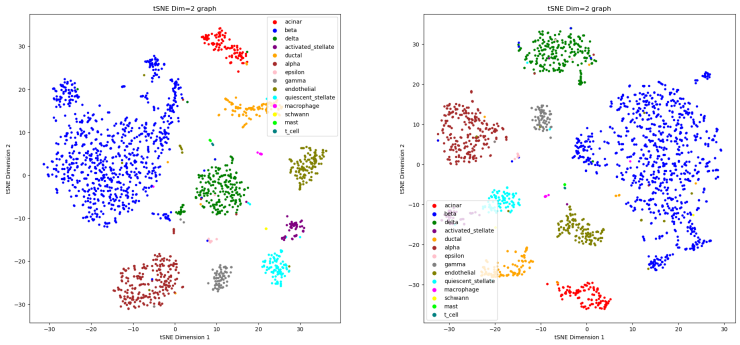
(b) cp10k data tSNE result



(c) log data tSNE result



(d) sqrt data tSNE result

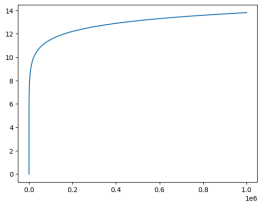


(e) log cp10k data tSNE result (f) sqrt cp10k data tSNE result

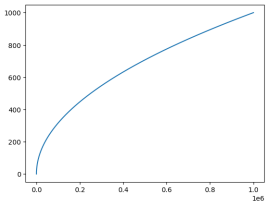
Figure 14: tSNE results for the different types of scRNA data

From the (a) and (b) graphs above, we can find that scRNA data doesn't show its relations inside clearly if taking no actions or just normalize the row sum to a fixed number. These 2 graphs can't clearly classify the cells correctly because some cells are separated into several parts, like *beta* and *ductal*. Some different kinds of cells are mixed up, like *quiescent_stellate*, *ductal* and *activated_stellate*.
From the (c) (d) and (b) graphs above, we can find that scRNA data shows its relations among the points inside better if you take the log/sqrt* processing instead of normalize the row sum. In fact, log/sqrt processing can reduce the influence of some extremely high value in the row (point) and increase the influence of some low value. So, we have reasonable suspects that the col diversity is larger compared with rows, which means genes diversity is larger than cells. Some genes might have very high expression and controls the distance between cells too much but they don't have the enough information to distinguish different types of cells. However, there're also some little problems with the method only taking log/sqrt calculation. For example, *quiescent_stellate* and *activated_stellate* don't have the clear bound.
Taking the cp10k normalization and log/sqrt processing together is the best choice for tSNE, most cells are classfied well in this condition.

**log/sqrt: log method for matrix m is $ln(m+1)$, sqrt method for matrix m is $sqrt(m)$. They share the similar functions that reduce the influence of large number and increase the influence of smaller number.



(a) log(X+1)



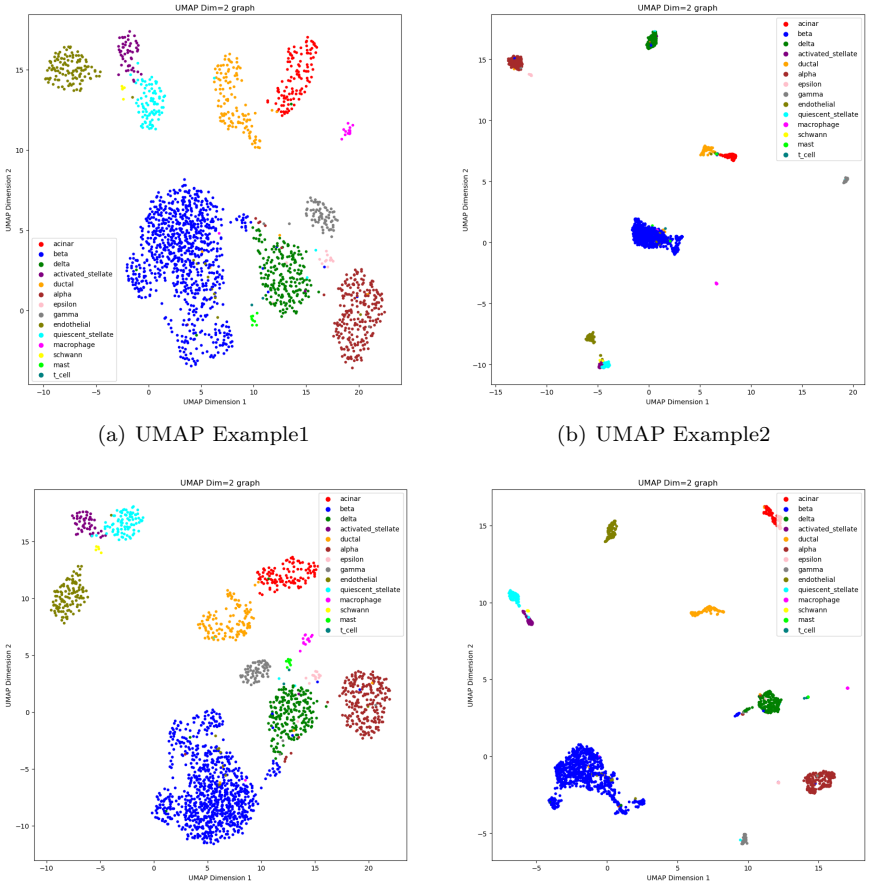
(b) sqrt(X)

Figure 15: tSNE results for the different types of scRNA data

For example, 1000 will be decrease to 6.9(log) and 100(sqrt) and 0.25 will be increase to 0.22(log) and 0.5(sqrt). Actually, function with the similar shape can be considered to replace log and sqrt here.

2.3 UMAP

Uniform Manifold Approximation and Projection. It's also a non-linear manifold analysis method. In conclusion, it's like a method to find the manifold that the points uniformly lying on. And then project this mainfold space to the lower-dimension space (find the projection method). Lastly, find the point position in the lower-dimension space via the projection method we get. (Code for the following graphs are in the visualization.ipynb)



(a) UMAP Example1

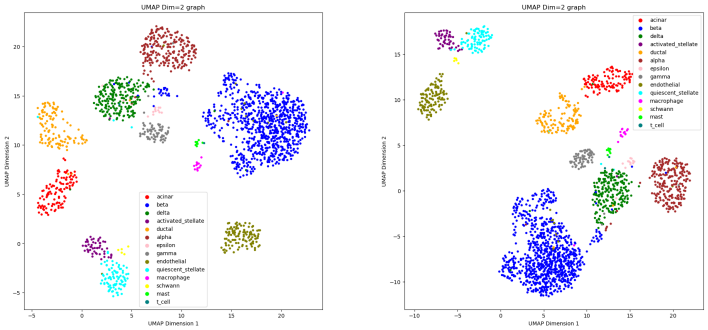
(b) UMAP Example2

(c) UMAP Example1 after PCA

(d) UMAP Example2 after PCA

Figure 16: UMAP results for the different types of scRNA data

The 2 graphs above are the UMAP visualization results (The first one has a larger min_dist). The UMAP visualization result is well but there're some types of cells mixed up together and some cells in the same type separated too far. UMAP also need to set more proper parameters for the correct visualization result and tSNE only needs "perplexity" to control.
From the graphs we can find the similar conclusion in tSNE part that it's better to process PCA at the initial large sparse scRNA matrix. Because PCA help to reduce the noise, which letting the data be distributed among the main vector space. With PCA before UMAP, the cells in the same type come together more closely as a circle. We can distinguish different types of cells more easily from this graph.



(a) Manhattan Distance UMAP (b) Euclid Distance UMAP

Figure 17: UMAP Example With different Distance

In the high dimension sparse matrix. Manhattan distance may be a better choice instead of Euclid distance because of curse of dimensionality, more direct similarity calculation, and larger robustness.

In the UMAP processing, using Manhattan & Euclid distance don't make a very large difference. We'll use Manhattan distance in the further work for the robustness and higher calculation speed.

The main purpose of our visualization process is to see the rough construction and have a direct feeling about the scRNA data. So tSNE is a better choice with less parameter to judge and clearer graph.

2.4 Some Other Potential Choice

there're also many other potential feasible choice for dimensionality reduction:

1. PHATE (Potential of Heat-diffusion for Affinity-based Transition Embedding)
2. Diffusion Map
3. PyMDE (Python Manifold Dimensionality Embedding)
4. etc.

3 Denoise

3.1 K-Means

For denoising work, we can use the cluster method to find the points being in one similar group. And then we can use the points in this group (relatives) to

denoise.
K-Means result and analysis can be seen in (4.1).
K-Means don't work well in scRNA data, so it's hard for us to denoise data using this clustering result.

3.2 MAGIC

3.2.1 Overview of MAGIC

The following graph is cited from *Recovering Gene Interactions from Single-Cell Data Using Data Diffusion* (David van Dijk, Roshan Sharma, Juozas Nainys,etc.,2018)

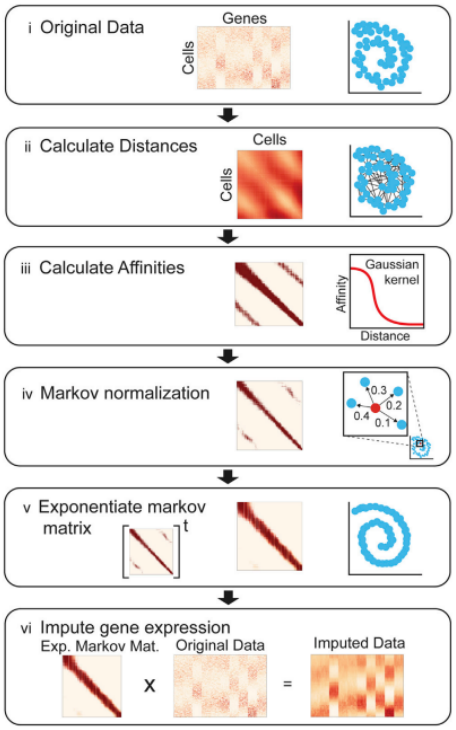


Figure 18: Overview of MAGIC

MAGIC uses distance between every 2 cells (matrix D)to get their affinity (matrix A). And then get the Markov Matrix (matrix M) from the affinity data. t power of nomalized M' represent the affinity matrix after t transfers, so we can get the denoised matrix by multipling this matrix with original data.
It uses kernel function to give the affinity non-linearly, which can help cells that are closer together are much closer together in the affinity matrix, and cells that are farther away are much farther away in the affinity matrix.

Using the Markov matrix to performing Markov diffusion on a affinity graph (similar to performing a random walk on the graph). It can help cells with smaller distance sharing their data to denoise. In other word, using the expression value after diffusion to replace the original data.

3.2.2 Pseudo-code of MAGIC

```
#An example of MAGIC using Gaussian Kernel Function
#matrix : the input matrix with "coordinates" of cells
#[c1,c2 ...]
#t: the diffusion times
Define function MAGIC(matrix, t):
    #Get a square matrix D, that
    D_ij=distance between c_i & c_j
    #Put the entry in D into Gaussian Kernel Function
    #and get the corresponding A
    A_ij = Gaussian_kernel(D_ij,sigma)
    #Normalize A to get Markov matrix M
    M=cp1(A)
    #Diffusion
    M=M^t
    #get denoised data
    return matrix@M
```

3.2.3 MAGIC Distance Matrix

Recall the matrix distribution we have shown in (1.4).

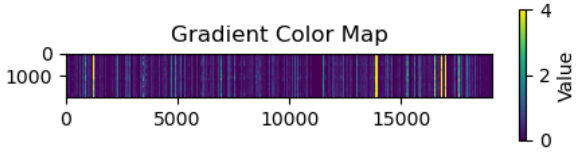


Figure 19: inDrop1 distribution

We can find that the matrix is very sparse and there're many 0s in the matrix. It'll cost too much calculation resourse on 0s and there might be lots of noise inside. So *maybe* (not so sure about it) choosing the genes with high expression can help to cluster and denoise the matrix, and at least, it will increase the speed we get the affinity and distance matrix.

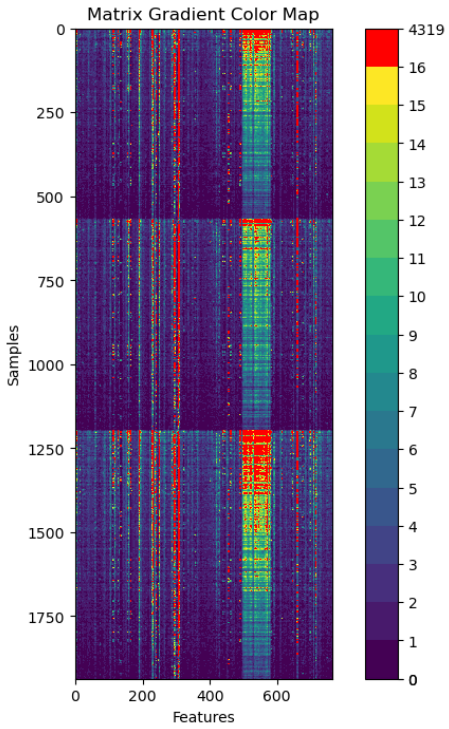


Figure 20: matrix with HEG(High Expression Gene)

(By the way, this data(pancreas.h5ad) is a little strange, because it seems that the data has already been arranged)
And here's the distance matrix of HEG:

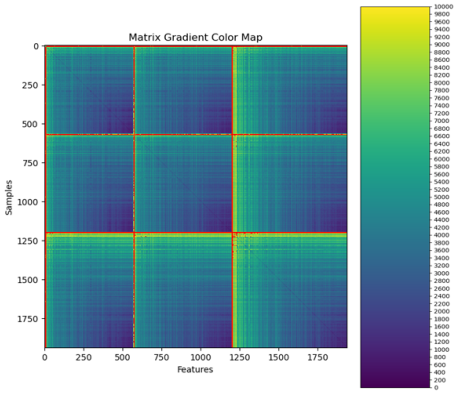


Figure 21: distance matrix with HEG (red block is >10000)

3.2.4 MAGIC Affinity Matrix

Once we get the distance matrix, we need to transfer the distance parameter to the affinity parameter.
We choose the kernel function to design the relations between distance and affinity. Recall that it has the similar thought in tSNE (2.2)that we can build the relations between distance and conditional probability. We also use this thought in the article (*Recovering Gene Interactions from Single-Cell Data Using Data Diffusion* (David van Dijk, Roshan Sharma, Juozas Nainys,etc.,2018)) they choose the Gaussian Kernel Function to measure the relation between distance and affinity, which is similar to the conditional probability.

Gaussian Kernel Function is shown below:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

t Kernel Function is shown below:

$$K(y_i, y_j) = \frac{1}{1 + \|y_i - y_j\|^2}$$

For the matrix with high dimension like distance matrix, maybe Gaussian Kernel Function could perform better. The MAGIC result for both kernel function is shown at (3.2.6)

For the σ, it's the parameter to control the bandwidth of kernel function, and commonly for determining the affinity calculation parameter of point p_i, we can set its neighbors into 3 layers. 1^{st} is the nearest neighbors, we set a fixed k for nearest neighbors, we use this range of points to determine the value of σ (p_{i_k} is the k^{th} nearest neighbor for p_i):

$$\sigma = \|p_i - p_{i_k}\|$$

Because all the points in this range are likely to be similar, they have affinity $\geq$ 0.6065.
And then we increase the distance to get our second layer.

If distance = 2σ, affinity will decrease to around exp(-2) = 0.1353.
if distance = 3σ, affinity will decrease to around exp(-9/2) = 0.0111.

The second layer contains p_j that satisfies (k is a parameter):

$$\sigma < \|p_i - p_j\| \leq k * \sigma$$

For example, k = 2, then we will find all the j points that $\sigma < \|p_i - p_j\| \leq 2 * \sigma$, the farthest point is p_{i_j}. All the j points in the 1^{st} and 2^{nd} range have the affinity $\geq$ 0.1353, and these points should be put. Other points with distance larger than $k\sigma$ in the 3^{rd} layer don't have enough evidence to be similar with

p_i. In other word, they're likely to be the different kinds of cells, thus we'll set the affinity of these points as 0 because we don't want these different kinds of cells share their information.

Here's a rough example for layers.

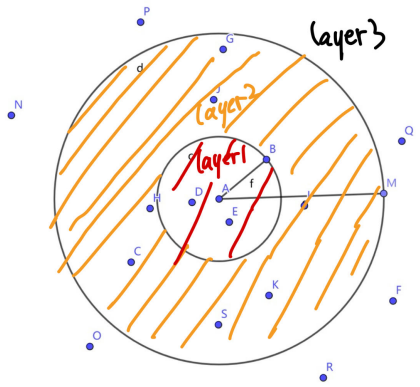


Figure 22: An rough example for layers

The inner circle has a radius = σ and the outer circle has a radius = 3σ. The nearest neighbors amount is 3 and we'll only consider the points in orange and red regions.

3.2.5 Improve the Speed of Calculating Distance Matrix

For the period calculating distance matrix should be O(n$\hat{3}$), it takes most time while we're running MAGIC. So we choose to take some methods to speed up this step.

Since our scRNA data matrix is sparse matrix and there're so much 0s, we'll waste much time on calculating $\|0-0\|$ or $\|p_j[k]-0\|$ (In our original matrix, there're only 9.92% non-zero entry, so only around 0.984% calculations in our L1 norm distance calculation period have non-zero items in both sides).

My first thought about dealing with this matrix is to store the index of non-zero entry in every row (This period is $O(n^2)$ but we only need to do it once in the whole calculation period) and before the distance matrix calculation, we'll compare the index list of both points. For the same index appears in both list, we'll apply $\|p_i[k]-p_j[k]\|$, for the index appears in one list, we'll add the entry directly and we won't consider other entry because they're all 0s. Conceptually, it'll decrease the minus operation to original 1% and decrease the add operation(add distance) to original 20%.

In this part, we'll discuss several functions provided by numpy, scipy and some other reposity for distance matrix calculation and k nearest neighbors searching:
For distance matrix, we have the following functions:

1. np.linalg.norm
2. np.sum(np.abs)
3. scipy.pdist
4. numpy broadcast

These are the accurate and speed results for these functions applied on normal matrix, sparse matrix.
As the sparse degree of matrix increases, will the speed change?
In conclusion, pdist > np.sum(np.abs) $\geq$ np.linalg.norm > numpy broadcast
For k nearest neighbors searching, we have the following functions:

1. np.argsort
2. sklearn.neighbors
3. BallTree
4. KDTree

In conclusion, sklearn.neighbors$\geq$ np.argsort> KDTree> BallTree

The methods above can calculate the accurate distance for every point pairs. However, from (3.2.4), it's obvious the affinity matrix only need the accurate distance of some point pairs (the distance of points in 1^{st} and 2^{nd} layers). So we can speed up this period via reducing the calculation amount of distance. We can firstly get the points in 1^{st} and 2^{nd} layers and only calculate their distance to our target point.
*(*Bandit-Based Monte Carlo Optimization for Nearest Neighbors)
Instead of calculating the accurate distance via traversing and minusing every entry in both list. This method uses the Multi-bandits thoughts to find the k nearest neighbors. For the problem finding k nearest neighbors for p_i, it treats other point as the bandits. We'll pull every arms (other points) once in one iteration. And then calculating the confidence interval of each arms. If there exists one arm having lower (upper) bound compared with the (lower) bound of second lowest arm, we'll treat it as the nearest neighbor. This method doesn't require us to calculate the difference of all the entries in list.
However, this method also has some problems. For example, since we have many cells in the same types, so after we pull the arms, there might be more than 1 (even more than k) arms (assumed to be a) having similar confidence interval. In this condition, we can never get the result of nearest neighbors because our

algorithm requires us to find the 1/textsuperscriptst neighbor firstly and these a neighbors aren't different enough. Though maybe we can distinguish the neighbors at last after many times of pulling, We'll waste too much time in this period, which is contradict to our target.
We don't need the arrangement of our k

This is the graph of calculating the 1^{st} neighbor for our scRNA data:

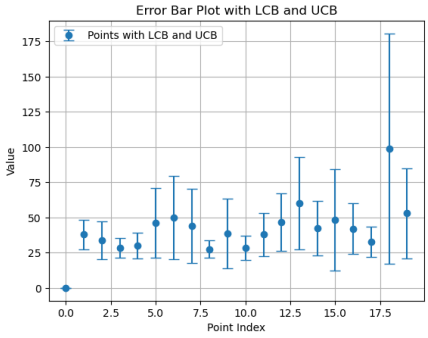


Figure 23: Confidence Interval Graph for Calculating 1^{st} Neighbor (5 Pull)

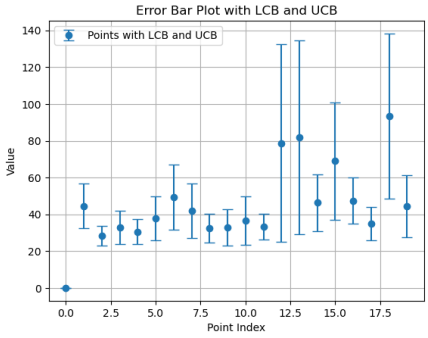


Figure 24: Confidence Interval Graph for Calculating 1^{st} Neighbor (15 Pull)

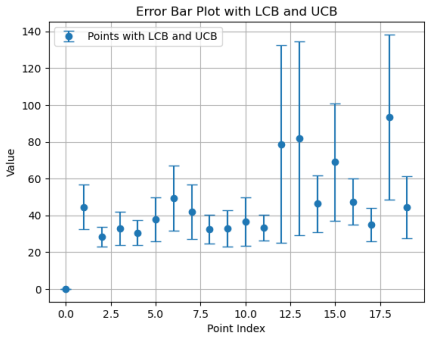


Figure 25: Confidence Interval Graph for Calculating 1st Neighbor (25 Pull)

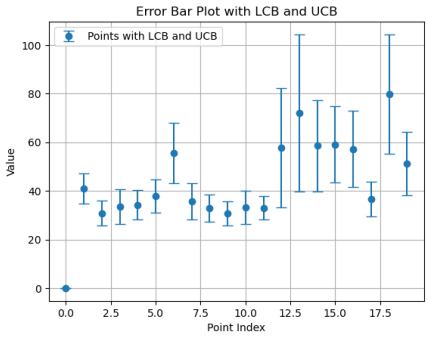


Figure 26: Confidence Interval Graph for Calculating 1st Neighbor (35 Pull)

These are the graph of range (*cell* 0 *cell* 19) as we have pulled from 5 to 35 times arms for finding the neighbors of *cell* 0. We can find that the acinar cells don't have a very large difference in their group so there're many very low bars there, which cost us much time to find the "real lowest" bar. If we insist to pick 1 lowest bar that will satisfy the condition that its UCB lower than others' LCBs as the article *Bandit-Based Monte Carlo Optimization for Nearest Neighbors* says, we'll waste much time on distinguish the same kind of cells to find the nearest neighbor (As we can see, the 3rd and 4th graph show that the same kind cells won't show a large difference while we're pulling them). We only need the kth neighbor's distance.

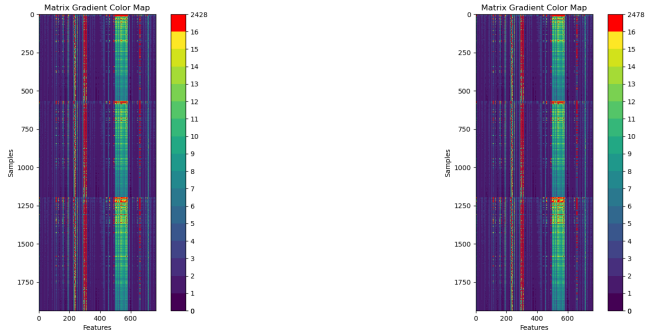
The label list for these 20 cells is:

[acinar ,acinar ,acinar ,acinar ,acinar ,acinar ,beta ,acinar ,acinar ,acinar ,acinar ,acinar ,delta ,delta ,beta ,activated_stellate ,beta ,acinar ,ductal ,beta]

There're several method trying to solve this problem, like adding some judging period to detect if there're k arms similar and they're all very low. Reducing the σ value (or set a fixed one) is also a considerable method. Or maybe we can

just choose the k lowest bar while we're reaching some pull amounts.
To be continued

3.2.6 Result of MAGIC



(a) t Kernel MAGIC (b) Gaussian Kernel MAGIC

Figure 27: MAGIC result with different kernel function

Don't have the ground truth matrix, having no idea about the result of MAGIC

4 Cluster

4.1 K-Means

K-Means is a popular clustering algorithm used in unsupervised learning. Its primary goal is to partition a dataset into K distinct, non-overlapping clusters, where each data point belongs to the cluster with the nearest mean.
This is the graph of visualization result after labelling the points using K-Means. (Code for K-Means graph is in the Visualization.ipynb)

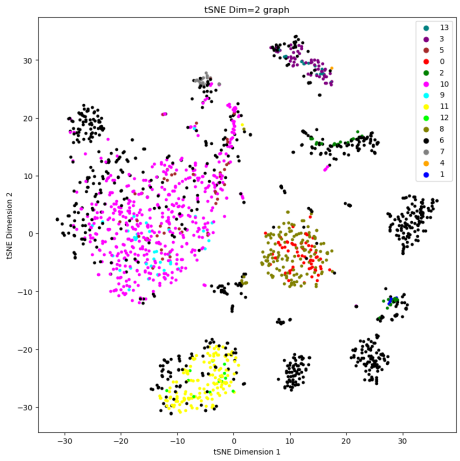


Figure 28: tSNE graph for K-Means Cluster

This is the pseudo-code of K-Means:
(Code for K-Means is in the Kmeans_cluster.ipynb)

```
#Data: Matrix of scRNA data
#C: clustering center amounts
#max_iter: maximum iteration amounts
#tol:tolerance of our function, if the difference of
new cluster center and old one during iteration < tolerance,
then return this center
Define KMeans_Manhattan(Data, C, max_iter, tol):
    Randomly select C initial cluster centers from Data
    For each iteration up to max_iter:
        Assign each data point to the nearest cluster
        center using Manhattan distance

        Calculate new cluster centers
        based on the mean of assigned data points

        If |old center - new center| < tol:
            Break

        cluster centers = new centers

    Return final cluster centers and labels
```

K-Means doesn't work so well in this condition.
For example, *Cell* 6 are separated everywhere and *Cell* 8 and *Cell* 0 are mixed up.
These problems may occur because of the data sparsity and the complexity

of noise. It is also assumed that the data are distributed in spherical clusters for K-Means and data with high dimensions can degrade the performance of K-Means. So it's better to choose some other available method for clustering.

4.2

5 some web

5.1 probability model

5.2 denoising

MAGIC: DREMI:https://krishnaswamylab.org/projects/dremi

5.3 cluster