Dimensionality Reduction and Clustering: single cell RNA-seq Data

Group 8: Spencer Haupert, Boya Jiang, Kailin Wang

12/17/2021

Introduction

In recent years, there has been considerable interest in analyzing single-cell RNA (scRNA) data. When it comes to complex pathologies like cancer, scRNA data can illuminate the heterogeneities and commonalities across different types of cell. Unsupervised learning techniques are often employed to cluster cells.

However, scRNA data analysis can suffer from the so-called "curse of dimensionality" since it is often the case that $p \gg n$ or at least $p \approx n$. Therefore, a dimensionality reduction step is often in order before clustering to make this computationally challenging problem tractable. Another challenge we face is that common clustering algorithms' performance can vary wildly depending on the initialization procedure. Therefore, our objectives are two fold: 1) implement a robust and efficient dimensionality reduction technique and 2) improve existing, widely-used clustering methods to mitigate issues related to initialization.

Our test data comes from The Cancer Genome Atlas's Pan-Cancer Atlas data products [1]. The particular dataset we use is hosted on UCI's ML repository [2], and it contains labeled scRNA data for 5 cancer types.

For our project, we implement a sparse version of Principal Component Analysis (PCA), k-means clustering, and the EM algorithm for a gaussian mixture model. We make these implementations available to the public through two R packages, `spcaRcpp` (for sparse PCA) and `clusteringscRNA` for k-means and the EM algorithm. Links to these packages, to a Google Colab page with a brief tutorial, and to the dataset used for evaluation can be found below.

Links:

R Packages:
https://github.com/srhaup2/clustering_scRNA
https://github.com/BoyaJiang/spcaRcpp
Google Colab:
https://colab.research.google.com/drive/14U0oFzB21j1-rswnQfkHt3YT93l2Z9-7#scrollTo=v3tym2Lcq5v-
Database:
https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq

Algorithms

1. spcaRcpp

Principal component analysis (PCA) is a popular data-processing and dimension-reduction technique. However, PCA suffers from the fact that each principal component is a linear combination of all the variables, making it difficult to interpret the results. Sparse principal component analysis (SPCA) was designed to remedy this inconsistency and to give additional interpretability to the projected data. Specifically, SPCA promotes sparsity in the modes, and the resulting sparse modes have only a few active (non-zero) coefficients, while the marjority of coefficients are zero. As a consequence, the model has improved interpretability, because the principal components are formed as a linear combination of only a few of the original variables. This method also prevents overfitting in a data setting where the number of variables is much greater than the number of observations ($n >> p$).

The formulation of SPCA by Zou, Hastie and Tibshirani [3] directly incorporates sparsity inducing regularizers into the optimization problem:

$$\begin{aligned} &\underset{\mathbf{A},\mathbf{B}}{\text{minimize}} f(\mathbf{A},\mathbf{B}) = \tfrac{1}{2}\left\|\mathbf{X}-\mathbf{X}\mathbf{B}\mathbf{A}^\top\right\|_{\mathrm{F}}^2 + \psi(\mathbf{B}) \\ &\text{subject to } \mathbf{A}^\top\mathbf{A} = \mathbf{I} \end{aligned}$$

where B is a sparse weight matrix and A is an orthonormal matrix. The penalty ψ denotes a sparsity inducing regularizer such as the elastic net. Specifically, the optimization problem is minimized using an alternating algorithm:

- **Update A**. With B fixed, we find an orthonormal matrix $\mathbf{A}^\top\mathbf{A} = \mathbf{I}$ which minimizes

  $$\left\|\mathbf{X}-\mathbf{X}\mathbf{B}\mathbf{A}^\top\right\|_F^2.$$

  which has the closed form solution $\mathbf{A}^* = \mathbf{U}\mathbf{V}^\top$, where $\mathbf{X}^\top\mathbf{X}\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$.
- **Update B**. With A fixed, we solve the optimization problem

  $$\min_{\mathbf{B}} \frac{1}{2}\left\|\mathbf{X}-\mathbf{X}\mathbf{B}\mathbf{A}^\top\right\|_{\mathrm{F}}^2 + \psi(\mathbf{B}).$$

The problem splits across the k columns of $\mathbf{B}$, yielding a regularized regression problem in each case:

$$\mathbf{b}_j^* = \arg\min_{\mathbf{b}_j} \frac{1}{2}\left\|\mathbf{X}\mathbf{A}(:,j) - \mathbf{X}\mathbf{b}_j\right\|^2 + \psi\left(\mathbf{b}_j\right)$$

The principal components can then be calculated as a sparsely weighted linear combination of the observed variables $\mathbf{Z} = \mathbf{X}\mathbf{B}$. The $\mathbf{B}$ update step relies on an iterative method using proximal gradient methods to find a stationary point.

There are several existing R packages that implements SPCA, i.e. `sparsepca`, `elasticnet`, and `EESPCA`. Among these, the `sparsepca::spca` provided a starting point for optimizing the SPCA function in terms of computational efficiency[4,5]. In order to improve the performance of the function, the iterative step was re-implemented in RcppArmadillo, which provides an interface to the Armadillo C++ numerical algebra library. RcppArmadillo offers a balance between performance and ease of use. The resulting package is `spcaRcpp`. The `spcaRcpp` function from the package takes in a $n \times p$ data matrix or data frame X, a parameter k indicating the maximal rank, the sparsity controlling parameter α,

the ridge shrinkage parameter β, a logical value `center`, the maximum number of iterations and the stopping criteria for the convergence. The function then returns a list containing the following: a matrix of variable loadings, standard deviations, eigenvalues, centering, variance, and the principal component scores.
By performing SPCA on the `tumor` data using the following code:

```
spcaRcpp(tumor, k = 15, alpha = 1e-04, beta = 1e-04, center = TRUE, max_iter = 1000, tol = 1e-05)
```

The function returns 15 principal components (PCs) with a cumulative explained variance ratio of 70%. Figure 1. is a visualization of PC1 vs PC2 by each known true tumor label. The validity of `spcaRcpp` is confirmed by `all.equal()` tests comparing to the original `sparsepca::spca` function. The performance of `spcaRcpp` is tested using `microbenchmark` after 100 runs. As shown in Figure 2a., the re-implementation of SPCA using Rcpp (top) successfully increased its speed comparing to the original `sparsepca::spca` (bottom) function.

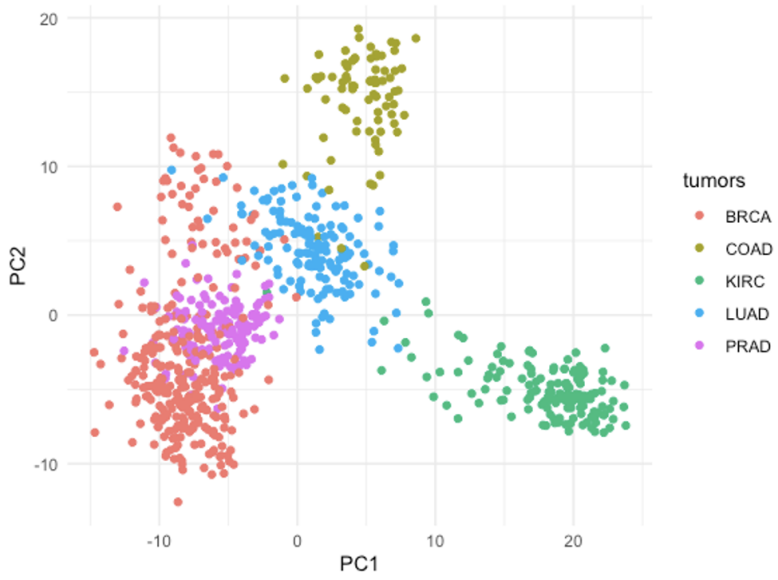


Figure 1. Cluster of Tumor scRNA-seq

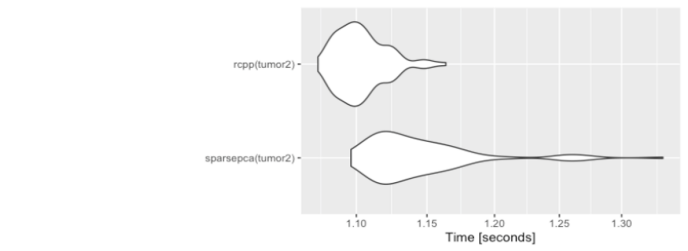


Figure 2a. Microbenchmark plot of spcaRcpp() vs sparsepca::spca()

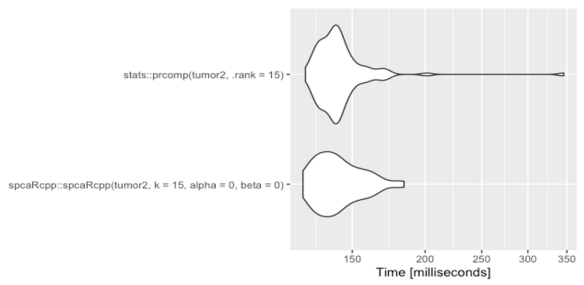


Figure 2b. Microbenchmark plot of stats::prcomp() vs spcaRcpp()

It is also worth noting that when setting both α and β to 0, the `spca` function is no longer introducing sparsity, and the results returned are the same as traditional PCA. Figure 2b. shows the speed of `spcaRcpp` and `stats::prcomp` are similar when removing the sparsity in the modes.

2. EM Clustering in Multivariate Normal Mixtures

EM clustering is one of the most widely-used algorithm for clustering on RNA-seq data. There are several existing R packages that implement EM algorithm for clustering on multivariate normal mixture data. One of the most popular EM clustering algorithm that addresses this kind of data is `mixtools::mvnormalmixEM` [6,7,8]. However, this is a general version that does clustering on multivariate normal mixtures data without discriminations on the mutual independence among the variables, which brings in unnecessary complexities when dealing with the mutually independent case. For this reason, We developed the class `normMixEM` in `clusteringscRNA` package specifically for the case of mutual independent variables data.

Simplification is realized based on the fact that when the independency holds, the covariance matrix in each mvnormal distribution would be diagnal, thus can be further compressed into vectors. In this case, `for` loops on a list of matrix can be replaced by matrix operations, which makes improvements on efficiency possible. The likelihood and log-likelihood function for mutually independent case can be expressed as follows:

$$L(\theta) = p(\boldsymbol{x}, \boldsymbol{z}|\theta) = \prod_{c=1}^{k}\prod_{i=1}^{n}\left\{\pi_{z_i}\prod_{j=1}^{p}P(x_{i,j}|p_{z_i,j})\right\}^{I_{z_i=c}}$$

$$l(\theta) = logL(\theta) = logP(\boldsymbol{x}, \boldsymbol{z}|\theta) = \sum_{i=1}^{n} log(\pi_{z_i}) + log(P(\boldsymbol{x_i}|\boldsymbol{\mu}_{z_i}, \sigma_{z_i}))$$

$$= \sum_{c=1}^{k}\sum_{i=1}^{n} I_{z_i=c}\left\{log\pi_{z_i} + log(P(\boldsymbol{x_i}|\boldsymbol{\mu}_{z_i}, \sigma_{z_i}))\right\} = \sum_{c=1}^{k}\sum_{i=1}^{n} I_{z_i=c}\left\{log\pi_{z_i} + \sum_{j=1}^{p} logP(x_{i,j}|\mu_{z_i,j}, \sigma_{z_i,j})\right\}$$

where: $logP(x_{i,j}|\mu_{z_i,j}, \sigma_{z_i,j}) = log(\frac{1}{\sqrt{2\pi}\sigma_{z_i,j}}e^{-\frac{1}{2}(\frac{x_{i,j}-\mu_{z_i,j}}{\sigma_{z_i,j}})^2}) = -\frac{1}{2}log(2\pi) - log(\sigma_{z_i,j}) - \frac{1}{2}(\frac{x_{i,j}-\mu_{z_i,j}}{\sigma_{z_i,j}})^2$

The `normMixEM` algorithm consists of three main parts: `normMixEM` (class), `rmvnorm_chol` (function) and `normalmix_init` (function). A comparison table between the `clusteringscRNA` and `mixtools` is displayed below:

Comps in clusteringscRNA	Comps in mixtools	Usage	Improvements
normMixEM( )	mvnormalmixEM( )	A class includes methods realizing EM algorithm steps	Replace loops with matrix operations
rmvnorm_chol( )	rmvnorm( )	Random number generation function based on Cholesky-decomp.	Replace Eigen-decomp. with Cholesky-decomp.
normalmix_init( )	mvnormalmixinit( )	Parameter initialization function for normMixEM( )	Replace loops with matrix operations

Table 1. Summary of functions and mixtools counterparts

We conducted an example test comparing the runtime of `normMixEM()` with `mvnormalmixEM()` using the following code. In this case, we can see that `normMixEM()` gives 'perfect' clustering, and runs 6 times faster than the `mvnormalmixEM()` in `mixtools`. Further implementations of `normMixEM()` on the tumor data is discussed in the results part.

```
set.seed(1000)
x.1 <- rmvnorm(40, c(0, 0))
x.2 <- rmvnorm(60, c(3, 4))
X.1 <- rbind(x.1, x.2)
class_t <- c(rep(1,40),rep(1,60))
EM <- normMixEM$new(input_dat = X.1,num_components = 2L)
res_EM <- EM$run.EM(loglik_tol=1e-5)
class <- apply(res_EM$prob_mat, 1, which.max)
```

```
plot(1L:res_EM$iter,res_EM$loglik_list,type="l")
kable(data.frame(est=class, true=class_t) %>% count(true,est) %>% mutate(freq=n/sum(n)))
```

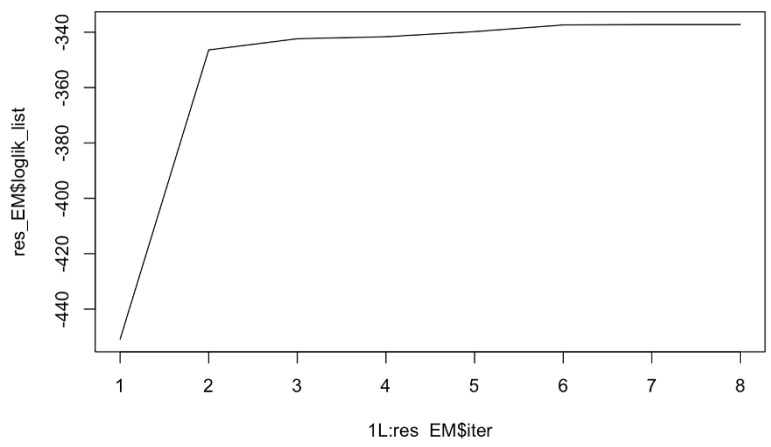


Figure 3. Log-likelihoood vs number of iterations

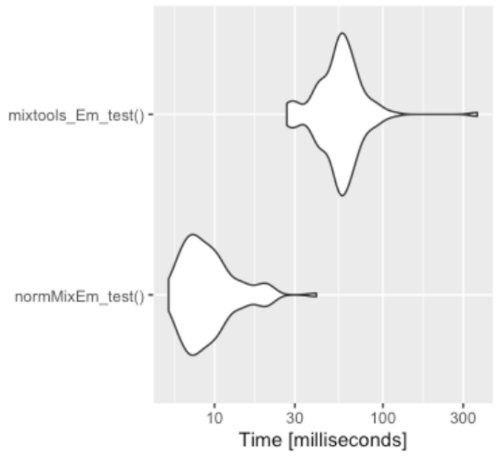


Figure 4. Microbenchmark plot of mixtools::mvnormalmixEM() vs normMixEM()

```
Unit: milliseconds
              expr       min        lq     mean    median        uq       max neval
   normMixEm_test()  5.169736  7.715231 10.18093  8.857127 10.52796  62.17496   100
 mixtools_Em_test() 26.886495 43.181047 62.65075 57.349866 68.84627 200.00337   100
```

true	est	n	freq
1	1	40	0.4
1	2	60	0.6

3. K-means Clustering

K-means is a very popular and relatively straightforward clustering algorithm. Many variations exist, but the most common implementation is referred to as Lloyd's algorithm [9]. LLoyd's K-means algorithm is as follows:

1. Randomly assign k datapoints to be the initial centroids (cluster centers)
2. Iterate until clusters do not change:
   (a) Assign rest of data to closest centroid (according to squared Euclidean distance)
   (b) Re-calculate centroids

Formally, cluster quality is assessed by **within-cluster sum of squared error (WCSSE)** given by:

For clusters $C_1, ..., C_k$, $WCSSE = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$ where μ_i is the centroid for each cluster.

Lloyd's algorithm is attractive because of its ease of implementation and because it is rather intuitive. However, because the initialization procedure is completely random and because K-means converges only to a local optima, Lloyd's method can result in quite poor clusters if bad initial centroids are chosen by chance.

For this reason, attempts have been made to improve Lloyd's method's initialization [10]. One very popular method is kmeans++, which is the default method in many software packages including the ubiquitous Python scikit-learn package [11].The method is as follows:

1. Randomly choose one centroid
2. Iterate until we have k centroids:
   (a) Compute the squared distance between non-centroids and all current centroids and take the minimum for each non-centroid
   (b) The next centroid is chosen with probability proportional to the distance computed in (a)
3. Proceed with Lloyd's k-means algorithm with these centroids

Kmeans++ represents a considerable improvement over random initialization. This method operates under the idea that cluster centers should be spread out to maximize the opportunity to find the global maxima. Despite the popularity of kmeans++, some studies suggest it is not the optimal initialization procedure [12]. The authors of kmeans++ also propose another even better method called greedy kmeans++. The algorithm is as follows:

1. Randomly choose one centroid
2. Iterate until we have k centroids:
   (a) Compute the squared distance between non-centroids and all current centroids and take the minimum for each non-centroid
   (b) **j new centroids are sampled with probability proportional to the distance computed in (a)**
   (c) **The next centroid is the one that, if chosen, would result in the lowest total WCSSE**
3. Proceed with Lloyd's k-means algorithm with these centroids

The difference between kmeans++ and greedy kmeans++ is highlighted in bold. In essence, instead of sampling one new data point each iteration, j are sampled and the best one is chosen.

The most popular implementation of kmeans in R is `stats::kmeans()`, but several others exist including `flexclust::kcca()`, `clustR::KMeans_rcpp()`, and `pracma::kmeanspp`. Some of these functions implement kmeans++, but to our knowledge, no current R package implements greedy kmeans++.

Our kmeans implementation is capable of running Lloyd's algorithm with the following initialization methods: random, kmeans++, and greedy kmeans++. The function can be used as shown below:

```
kmeans_clust(X, k, nstart = 1L, iter.max = 10L, init.method = "random", center = TRUE, scale = TRUE)
```

It takes as input:

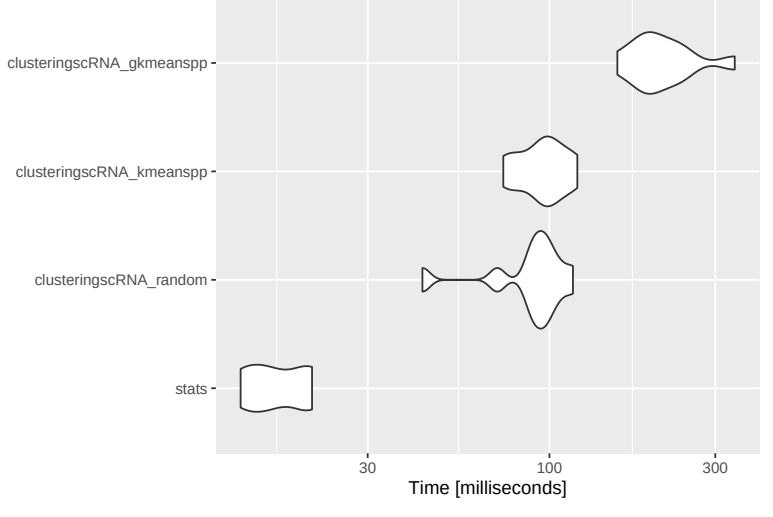
- **X** n x p data matrix,
- **k** number of clusters
- **nstart** number of times to perform k means on the data
- **iter.max** max iterations per run
- **init.method** method for centroid initialization - choose from random, kmeans++, greedy kmeans++ (gkmeans++)
- **center** if TRUE, subtracts the mean for each feature
- **scale** if TRUE, scales each feature by its standard deviation

And it returns:

- **clusters** n x p+1 matrix of cluster assignments where the first column is cluster assignments
- **iter** number of iterations
- **centroids** k x p+1 matrix of centroids where the first column is cluster assignments
- **wcsse** - min within-cluster SSE over all nstart iterations

First, we present a speed comparison of `kmeans_clust()` with `stats::kmeans()`.

Figure 5. Speed Comparison of stats vs clusteringscRNA



We can see that the function in the `stats` package is the fastest, which is not surprising since it is written in C and Fortran. However, considering that only a small portion of the `clusteringscRNA` function is written in Rcpp, these results are not too bad. The random initialization and kmeans++ method have a similar runtime, but the greedy kmeans++ method has a much longer runtime because it does many more distance calculations.

Next, we can evaluate accuracy on the raw data using Adjusted Rand Index:

```
##                             Median.ARI Median.WCSSE
## stats                        0.8217335      1011316
## clusteringscRNA_random       0.8042967      1041367
## clusteringscRNA_kmeanspp     0.7741871      1051000
## clusteringscRNA_gkmeanspp    0.8800288       985680
```

Greedy kmeans++ offers an improvement in accuracy and precision as measured by ARI and WCSSE. It is worth noting that the lowest WCSSE does not always correspond to the best ARI. In testing these functions, we found greedy kmeans++ fairly consistently provided the lowest WCSSE, but it does not always provide the highest ARI.

Results

In order to test the performance and speed of our functions, we performed SPCA and EM or k-means clustering on the `tumor` data. Based on prior knowledge, the number of clusters was set to 5. First, dimensionality reduction was applied to the data using `spcaRcpp`. The resulting principal components were clustered using either EM or k-means algorithm. Table 2a shows the frequency of observations assigned to each cluster compared with the true labels from one run using EM clustering. The percentage of accurately clustered entries is approximately 98.9%. However, since EM algorithm depends on the initialization point, this result is not reliable. Therefore, we also calculated the average Adjusted Rand Index (ARI) from 10 runs. ARI computes the similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings. The resulting average ARI from EM algorithm was **0.733**, and the average runtime was **0.172** seconds.

true	est	n	freq
BRCA	2	1	0.0012484
BRCA	3	299	0.3732834
COAD	2	3	0.0037453
COAD	4	75	0.0936330
KIRC	2	1	0.0012484
KIRC	5	145	0.1810237
LUAD	2	139	0.1735331
LUAD	3	2	0.0024969
PRAD	1	134	0.1672909
PRAD	3	2	0.0024969

Table 2a. Clustering results from SPCA + EM

true	est	n	freq
BRCA	1	252	0.3146067
BRCA	4	48	0.0599251
COAD	4	4	0.0049938
COAD	5	74	0.0923845
KIRC	2	144	0.1797753
KIRC	4	2	0.0024969
LUAD	4	141	0.1760300
PRAD	3	135	0.1685393
PRAD	4	1	0.0012484

Table 2b. Clustering results from SPCA + K-means

Next, we computed the accuracy of k-means clustering after dimensionality reduction using the following code. The initialization method `gkmeans++` was chosen since it was expected to outperform other methods as discussed previously. The `nstart` was set to 10 in order to achieve higher accuracy.

```
clusteringscRNA::kmeans_clust(spca_out$scores,
                              k =5,
                              nstart = 10,
                              init.method = "gkmeans++",
                              center = F,
                              scale = F)
```

From Table 2b, the accuracy of k-means clustering was approximately 93%. Again, in order to better assess the reliability and accuracy of this method, average ARI was computed from 10 runs. The average ARI of k-means clustering was **0.818**, and the average runtime was **1.095** seconds. In comparison, the same k-means function took approximately **1.612** seconds to perform clustering on the raw data without dimensionality reduction. This is expected because the loss of information from SPCA has an impact on both the speed and accuracy of clustering methods.

Discussion

In this study, we presented new versions of SPCA, EM and k-means algorithms with improvements on efficiency and accuracy. To certify the accuracy of our method, we applied SPCA and EM clustering or k-means clustering on the `tumor` data and estimated the consistency of the real labels and ours by calculating ARI values. Specifically, greedy k-means method achieved higher accuracy (average ARI = 0.818) compared to EM algorithm (average ARI = 0.733). However, the performance speed of greedy k-means is much slower than EM algorithm (1.095 secs vs 0.172 secs on average). This result provides some insight on the speed/accuracy trade-off when selecting clustering methods for scRNA data.

Although our methods were able to demonstrate similar or even better performance compared to their existing counterparts, there are still several limitations. One main difficulty is that both k-means and EM algorithm are sensitive to initialization parameters, which include the starting point and also the cluster number `k`. Given the randomness of the initialization, the clustering result may vary from time to time, which reduces the stability of the algorithms and increases the difficulty of interpretation.

The speed of clustering algorithms is another concern. As the single-cell datasets have become larger and larger, improvements on efficiency would be required,and this can be realized by writing in C++, Fortran, etc. Since only a small part of the codes in our study is written in C++, it's reasonable to expect further improvements on efficiency by rewriting in other compiled languages.

References

1. "PanCanAtlas Publications | NCI Genomic Data Commons." Accessed December 16, 2021. https://gdc.cancer.gov/about-data/publications/pancanatlas.
2. "UCI Machine Learning Repository: Gene Expression Cancer RNA-Seq Data Set." Accessed October 27, 2021. https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq.
3. Hui Zou, Trevor Hastie & Robert Tibshirani (2006) Sparse Principal Component Analysis, Journal of Computational and Graphical Statistics, 15:2, 265-286, DOI: 10.1198/106186006X113430
4. N.B.Erichson, P.Zheng, K.Manohar, S.Brunton, J.N.Kutz, A.Y.Aravkin."SparsePrincipal Component Analysis via Variable Projection." Submitted to IEEE Journal of Selected Topics on Signal Processing (2018). (available at 'arXiv https://arxiv.org/abs/1804.00341).
5. N. B. Erichson, P. Zheng, S. Aravkin, sparsepca, (2018), GitHub repository
6. McLachlan, G. J. and Peel, D. (2000) Finite Mixture Models, John Wiley & Sons, Inc.
7. Benaglia T, Chauveau D, Hunter DR, Young D (2009). "mixtools: An R Package for Analyzing Finite Mixture Models." Journal of Statistical Software, 32(6), 1–29. http://www.jstatsoft.org/v32/i06/.
8. McLachlan, G. J. and Peel, D. (2000) Finite Mixture Models, John Wiley & Sons, Inc.
9. Lloyd, S. P. (1957). Least squares quantization in PCM. Technical Report RR-5497, Bell Lab, September 1957.
10. Arthur, David, and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding." In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 1027–35. SODA '07. USA: Society for Industrial and Applied Mathematics, 2007.
11. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
12. Celebi, M. Emre, Hassan A. Kingravi, and Patricio A. Vela. "A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm." CoRR abs/1209.1960 (2012). http://arxiv.org/abs/1209.1960.