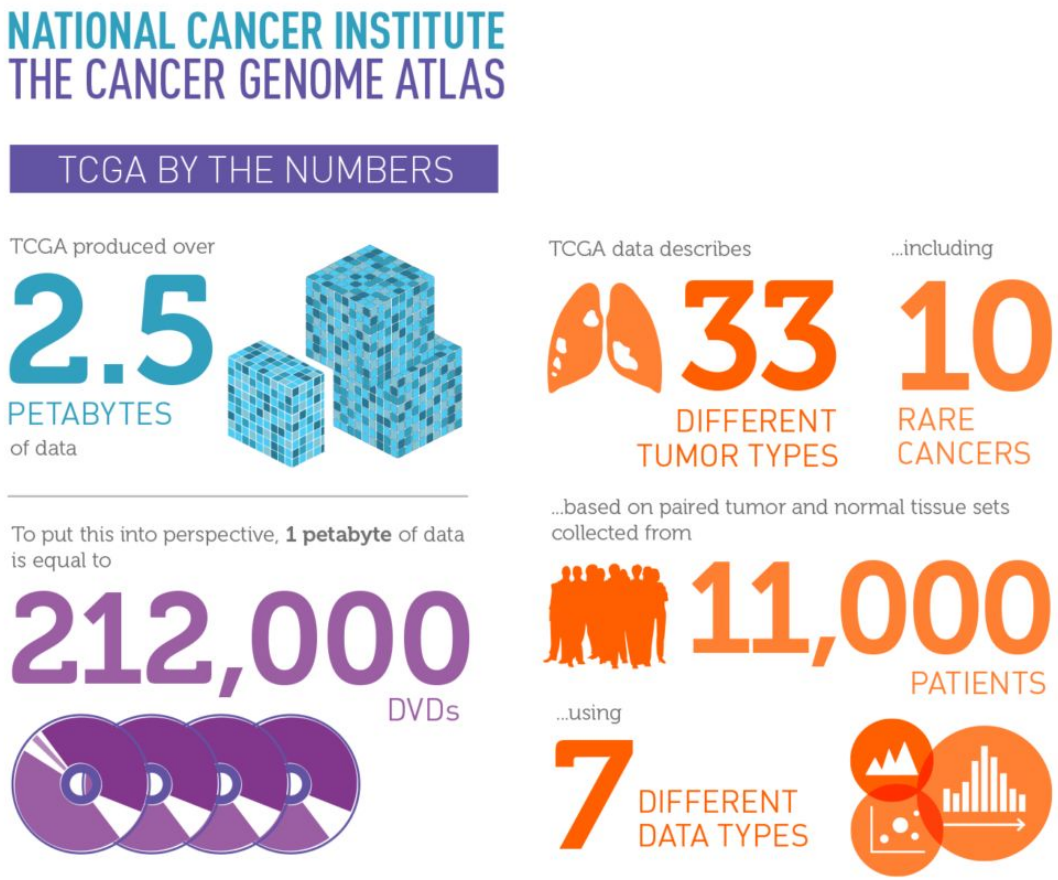
Dimensionality Reduction and Clustering: single cell RNA-seq Data

Spencer Haupert, Boya Jiang, Kailin Wang

Cancer scRNA data

- Lots of clinical interest in what cells express what genes, especially when it comes to cell pathologies (ex: cancer)
- Since 2006, The Cancer Genome Atlas (TCGA) project has sequenced over 20,000 samples acros 33 cancer types
- Their Pan-Cancer initiative aims to provide data products that let analysts compare gene expression across cancer types
- Our dataset consists of 5 different types of tumors: BRCA, KIRC, COAD, LUAD and PRAD (200 genes, 801 samples)



REF: https://www-nature-com.proxy.lib.umich.edu/articles/ng.2764, https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga, https://computerhistory.org/blog/decoding-cancer-with-the-cancer-genome-atlas-dr-jean-claude-zenklusen/

Clustering for scRNA data

Challenges:

- Data can be very high dimensional ($n \approx p$ or $n << p$)
- Common clustering algorithms' performance can vary wildly depending on the initialization procedure

Our Solutions:

- Use a combination of dimensionality reduction and clustering
- Make dimensionality reduction work well in this high-dimensional setting
- Improve upon existing, widely-used clustering methods

Dimensionality Reduction: Sparse Principal Component Analysis (SPCA)

- SPCA promotes sparsity in the modes; i.e., the sparse modes only have a few *active* coefficients
- highly localized, more interpretable, avoids overfitting when p >> n
- minimizes the following objective function:

$$\underset{\mathbf{A},\mathbf{B}}{\text{minimize}} \; f(\mathbf{A},\mathbf{B}) = \tfrac{1}{2}\|\mathbf{X} - \mathbf{X}\mathbf{B}\mathbf{A}^\top\|_{\mathrm{F}}^2 + \psi(\mathbf{B})$$

$$\text{subject to} \; \mathbf{A}^\top\mathbf{A} = \mathbf{I},$$

- existing SPCA packages:
  - **sparsepca**
  - elasticnet
  - EESPCA

spcaRcpp

- a RcppArmadillo integration of sparsepca
- https://github.com/BoyaJiang/spcaRcpp

```
devtools::install_github("BoyaJiang/spcaRcpp")
library(spcaRcpp)
spca_out = spcaRcpp(tumor2, k = 21, center = TRUE, alpha = 1e-4, beta = 1e-4)
```

Usage

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

Arguments

X	a numeric matrix or data.frame which provides the data for the sparse principal components analysis.
k	optional, a number specifying the maximal rank.
alpha	Sparsity controlling parameter. Higher values means sparser components.
beta	Amount of ridge shrinkage to apply in order to improve conditioning.
center	a logical value indicating whether the variables should be shifted to be zero centered.
max_iter	maximum number of iterations to perform.
tol	stopping criteria for the convergence.

Value

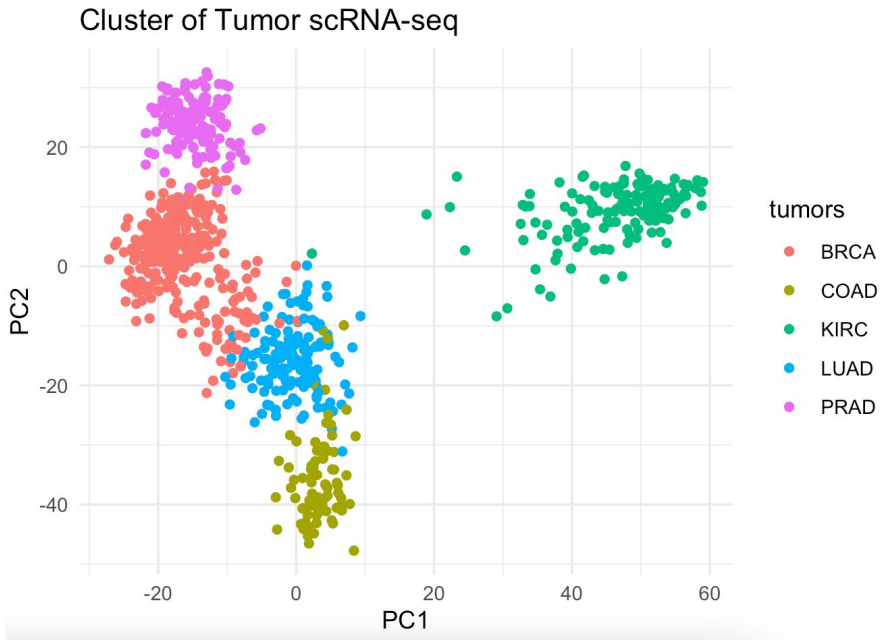
spcaRcpp returns a list containing the following six components:

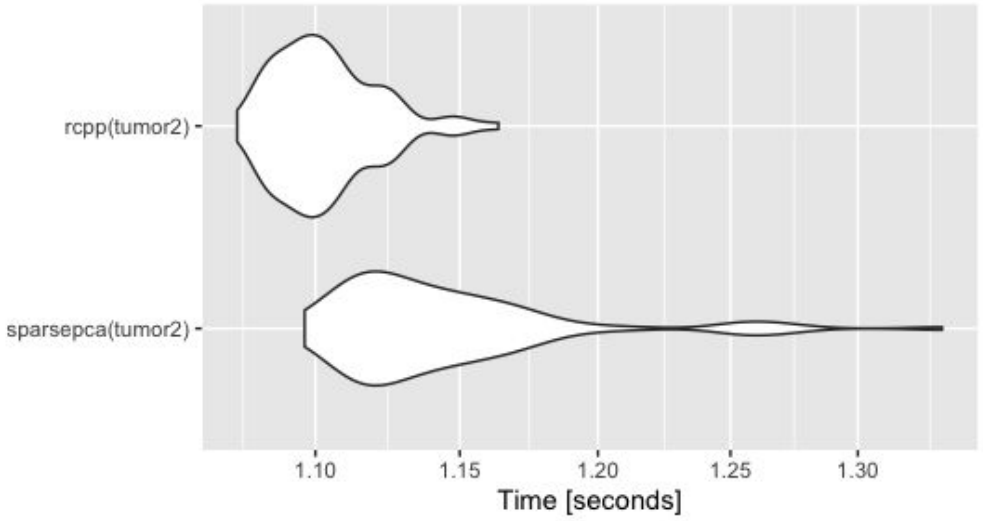
loadings	the matrix of variable loadings.
standard deviations	the approximated standard deviations.
eigenvalues	the approximated eigenvalues.
center	the centering used.
var	the variance.
scores	the principal component scores.

spcaRcpp

```
sparsepca <- function(data) return (sparsepca::spca( data, k = 21, alpha = 1e-4, beta = 1e-4,verbose = F ))
rcpp <- function(data) return (spcaRcpp( data, k = 21, alpha = 1e-4, beta = 1e-4 ))

mb = microbenchmark(sparsepca(tumor2),
                    rcpp(tumor2),
                    times = 100)
autoplot(mb)
```





Multivariate Normal Mixture EM Clustering

1. **Key assumptions:**
    a. Independent random variables(features)
    b. Normal distributions
2. **Likelihood function**

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

3. **Log-likelihood function**

$$\begin{aligned} l(\theta) &= logL(\theta) \\ &= \sum_{c=1}^{k} \sum_{i=1}^{n} I_{z_i=c} \left\{ log\pi_{z_i} + \sum_{j=1}^{p} logP(x_{i,j}|\mu_{z_i,j}, \sigma_{z_i,j}) \right\} \end{aligned}$$

a. E-step… b. M-step…

normMixEm

- **normMixEm(**input_dat, num_components**):**

  (based on Class *normMixEm* from lecture14)

- **normalmix_init( )** : Initialization function of parameters

  (based on *mvnormalmix.init()* from mixtools)

- **rmvnorm_chol( )** : Random number generation function of MVNormal based on Cholesky decomposition

  (based on lecture 9)

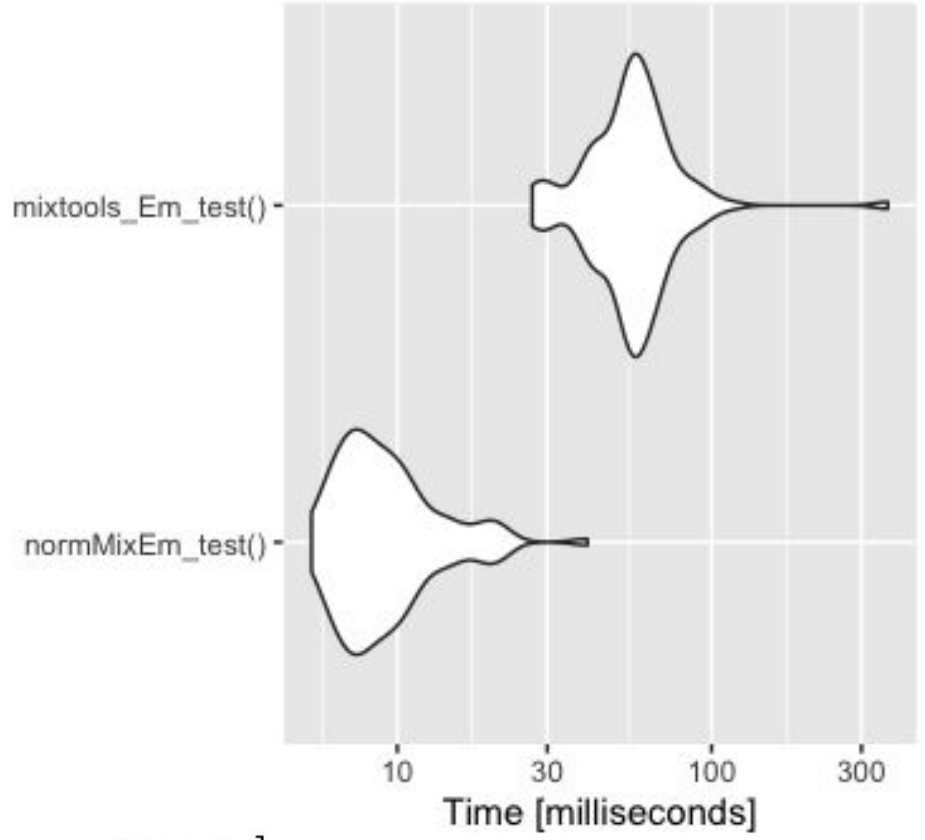
normMixEm V.S. *mixtools::mvnormalmixEM*

Speed

```
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(2,60)) # true labels
mu <- list(c(0, 0), c(3, 4))

normMixEm_test <- function(data = X.1, num_components= 2L ){
  EM <- normMixEm$new(input_dat = data,num_components = num_components)
  out.2 <- EM$run.EM(loglik_tol=1e-5)
}
mixtools_Em_test <- function(){
  out.1 <- mixtools::mvnormalmixEM(X.1, arbvar = FALSE, mu = mu,
                                   epsilon = 1e-02)
}

result <- microbenchmark(normMixEm_test(), mixtools_Em_test())
autoplot(result)
```




```
> result
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
```

normMixEm V.S. *mixtools::mvnormalmixEM*

- Accuracy of clustering in the previous example

Accuracy

Clustering result of *normMixEm*

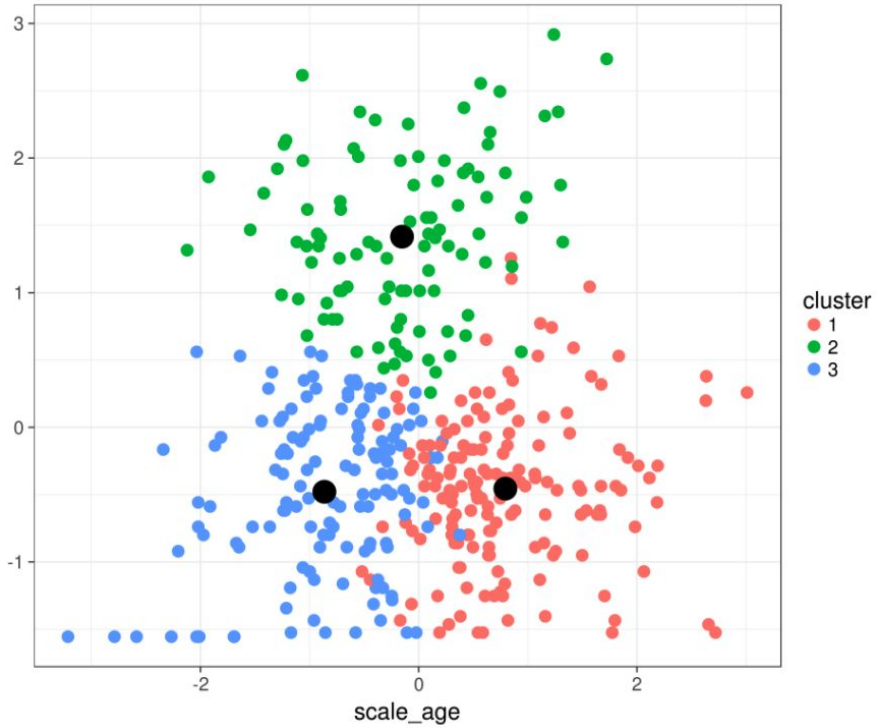
	true	est	n	freq
1	1	1	40	0.4
2	2	2	60	0.6

True data

```
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(2,60)) # true labels
mu <- list(c(0, 0), c(3, 4))
```

K-means Clustering

K-means is a very popular and relatively straightforward clustering algorithm. The most common implementation is referred to as **Lloyd's algorithm.**



Steps:

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

REF: https://rpubs.com/cyobero/k-means
https://en.wikipedia.org/wiki/K-means_clustering

K-means - initialization problem

Quality of clusters assessed by **within-cluster sum of squares** given by:

For clusters $C_1, ..., C_k$,

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$$

where μ_i is the centroid for each cluster

Problem: cluster quality is affected by the random initialization method

How can we improve our initialization method, and thus our cluster quality?

K-means++

Another popular initialization method

Steps:

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

REF: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

Greedy K-means++

Steps:

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 WCSS
3. Proceed with Lloyd's k-means algorithm with these centroids

REF: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

Our K-means implementation

Many existing packages:

- stats::kmeans()
- flexclust::kcca()
- pracma::kmeanspp()
- clustR::KMeans_rcpp()

Description

Clusters data according to k-means algorithm

Usage

```
kmeans_clust(X, k, nstart, iter.max, init.method)
```

Arguments

`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++)

Value

A list

- 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

Examples

```
kmeans_clust(X, k = 3, nstart = 5, init.method = "kmeans++")
```

Our K-means - speed

```
Unit: milliseconds
                                                                 expr        min         lq       mean     median         uq
      kmeans(brain, centers = 5, nstart = 1, algorithm = "Lloyd")   16.56256   17.70935   28.53108   23.02219   36.40693
   kmeans_clust(brain, k = 5, nstart = 1, init.method = "random") 2361.56747 2511.42605 2868.15236 2579.80416 3221.93433
  kmeans_clust(brain, k = 5, nstart = 1, init.method = "kmeans++") 2561.34022 3136.05081 3449.07094 3169.78173 3470.96459
 kmeans_clust(brain, k = 5, nstart = 1, init.method = "gkmeans++") 5260.80046 5659.11857 6439.09512 6329.19468 7150.81366
       max neval
  49.27628    10
3930.11971    10
4898.88594    10
7898.80726    10
```

- Much slower than stats::kmeans()....
- stats::kmeans is written in **C** and **Fortran**, while kmeans_clust() is written in **R**
- As initialization method complexity increases, time also increases, with greedy kmeans++ having the longest runtime

Our K-means - performance

Algorithm	Adjusted Rand Index
stats::kmeans(X, centers = 5, nstart = 1, algorithm = "Lloyd")	0.7699028
kmeans_clust(X, k = 5, nstart = 1, init.method = "random")	0.7707366
kmeans_clust(X, k = 5, nstart = 1, init.method = "kmeans++")	0.7619758
kmeans_clust(X, k = 5, nstart = 1, init.method = "gkmeans++")	**0.8405325**

Results

- EM clustering on principal components generated from spcaRcpp

```
s = Sys.time()
res_EM = normMixEm_test(data = spca_out$scores, num_components= 5L)
Sys.time() - s
class <- apply(res_EM$prob_mat, 1, which.max)
kable(data.frame(est=class, true=TC) %>% count(true,est) %>% mutate(freq=n/sum(n)))
```

- run-time: 0.237 secs
- adjustedRandIndex: 0.7328

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

Results

- Kmeans clustering on principal components generated from spcaRcpp

```
s = Sys.time()
res_kmeans = kmeans_clust(spca_out$scores, k =5, nstart = 1, init.method = "gkmeans++")
Sys.time() - s
PC <- res_kmeans$clusters[,1]
kable(data.frame(est=PC, true=TC) %>% count(true,est) %>% mutate(freq=n/sum(n)))
```

- run-time: 1.058733 secs
- adjustedRandIndex: 0.6321

true	est	n	freq
BRCA	2	300	0.3745318
COAD	2	2	0.0024969
COAD	4	76	0.0948814
KIRC	2	1	0.0012484
KIRC	5	145	0.1810237
LUAD	1	136	0.1697878
LUAD	2	5	0.0062422
PRAD	3	136	0.1697878

Discussion

- After dimensionality reduction, EM algorithm outperforms k-means in terms of speed and accuracy

- Speed/accuracy trade-off of k-means (with and without SPCA):
  - ARI: 0.8405 vs 0.6321
  - Speed: 6.4 secs vs 2 secs

- Limitations:
  - Both k-means and EM algorithm are sensitive to initialization point
  - Speed of clustering algorithms could be improved by writing in compiled language

References:

1. 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).
2. N. B. Erichson, P. Zheng, S. Aravkin, sparsepca, (2018), GitHub repository
3. McLachlan, G. J. and Peel, D. (2000) Finite Mixture Models, John Wiley \& Sons, Inc.
4. 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/.
5. https://www.eecs.umich.edu/techreports/systems/cspl/cspl-401.pdf