Project 4 - Group 6

Zeyu Gan, Virgile Mison, Galen Simmons, Siyuan Yao, Qingyuan Zhang

4/14/2017

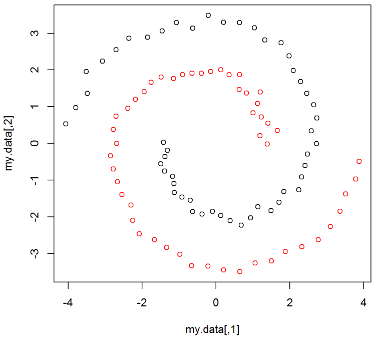
Setup: assign Knitr root directory and load dependencies

We set the `knitr` root.dir to the project directory (`projDir`) and load/install the necessary packages to run our `main.Rmd` script. [Code omitted]

Section I: Paper 3

"Name Disambiguation in Author Citations using a K-way Spectral Clustering Method"

Figure 1: Visual Representation of Spectral Clustering



As discussed in `Section 3.1` of the paper, authors Han, Zha, and Giles use **three** citation attributes to design features for name disambiguation. Those attributes are:

- co-author names
- paper titles
- publication venue titles

Together these attributes are called a 'citation vector.' For a dataset with m features, each citation is representated as an m-dimensional vector given by $M = (\alpha_1, \ldots, \alpha_m)$, where α_i is the weight assigned to feature i. Two types of feature weights assignments are profiled: (i) TFIDF and (ii) normalized TF ("NTF").

We demonstrate how we create the citation vector from our clean data sources in the `output` library in the chunks below:

Step 1: Load text file

We parse raw data from the `data` folder into the appropriate format using the `Data_Cleaning.R` script contained in the `lib` folder. THe parsed datasets are stored in the `output` folder.

```r
source(file.path(projDir,"lib","feature_extraction.R"))
exFile <- file.path(projDir,"output","Agupta.csv")
exText <- readTextFile(exFile)
exText %>%
  tbl_df() %>%
  select(QuestAuthor, Coauthor, Paper, Journal) %>%
  head() %>%
  kable(format = "markdown")
```

QuestAuthor	Coauthor	Paper	Journal
A Gupta	NA	Stanford DASH Multiprocessor: The Hardware and Software Approach	PARLE Parallel Architectures and Languages Europe
A Gupta	A Acero; Y Rui	Automatically extracting highlights for TV Baseball programs	ACM Multimedia
A Gupta	A Acharya; M Tambe	Implementation of Production Systems on Message-Passing Computers	IEEE Trans Parallel Distrib Syst
A Gupta	A Agarwal	Memory-Reference Characteristics of Multiprocessor Applications under MACH	SIGMETRICS Measurement and Modeling of Computer Systems
A Gupta	A Agrawal; N Chaddha; T Meng	Variable Compression Using JPEG	ICMCS International Conference Multimedia Computing and Systems
A Gupta	A Balachandran; E Sanocki; G Jancke; J Grudin; J Cadiz	Distance learning through distributed collaborative video viewing	CSCW Conference Computer Supported Cooperative Work

Step 2: Prep the co-author and journal terms to be included in the document term matrix

We want the individual co-author names and journal names to be considered as single 'terms' in our corpus. Therefore, we collapse the spaces separating the unique letters in someone's name, so that it appears to be a term. For example, "C L Zhang" would become "clzhang". Likewise, with journal names, we combine them into a single string without spaces so that each journal is a unique term in our Document Term Matrix.

```r
as_tibble(exText) %>%
    mutate(x = str_replace_all(Coauthor, " ", "")) %>%
    mutate(x = str_replace_all(x, ";", " ")) %>%
    mutate(y = str_replace_all(Journal, " ", "")) %>%
    mutate(term_col = tolower(paste(x, y, Paper))) %>%
    select(term_col) %>%
    head() %>%
    kable(format = 'markdown')
```

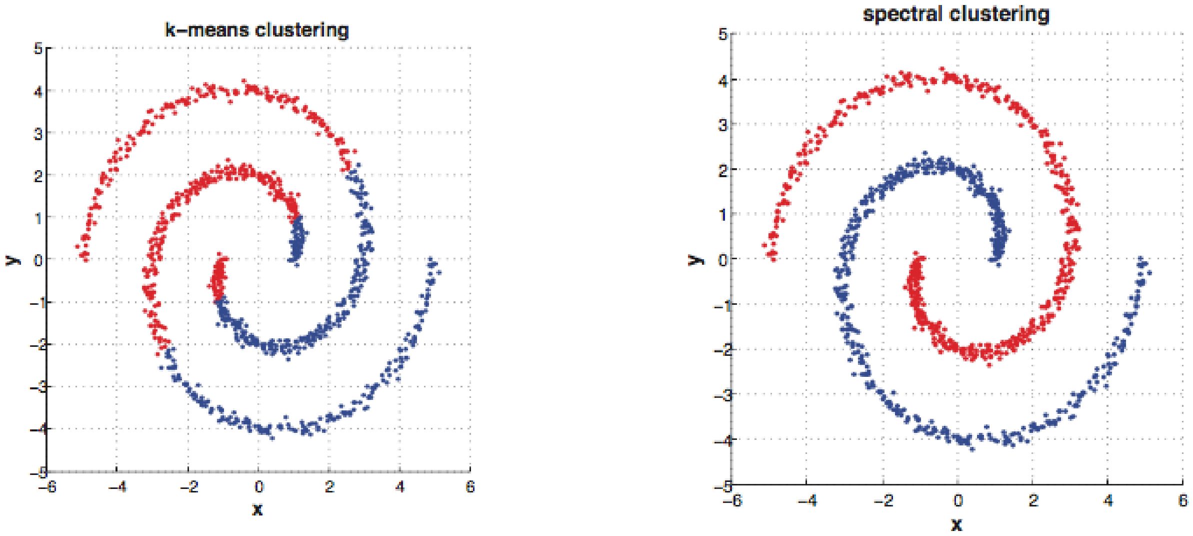
term_col
na parleparallelarchitecturesandlanguageseurope stanford dash multiprocessor: the hardware and software approach
aacero yrui acmmultimedia automatically extracting highlights for tv baseball programs
aacharya mtambe ieeetransparalleldistribsyst implementation of production systems on message-passing computers
aagarwal sigmetricsmeasurementandmodelingofcomputersystems memory-reference characteristics of multiprocessor applications under mach
aagrawal nchaddha tmeng icmcsinternationalconferencemultimediacomputingandsystems variable compression using jpeg
abalachandran esanocki gjancke jgrudin jcadiz cscwconferencecomputersupportedcooperativework distance learning through distributed collaborative video viewing

Step 3: Run our spectral clustering method on the example dataset

We use the `tm` package to construct citation vectors in the manner described above. We then use the `matching_matrix` and `performance_statistics` functions in the `lib/evaluation_metrics.R` file to benchmark our results. Below we describe the details of our spectral clustering implementation. Our spectral clustering implementation is contained in `lib/SpectralClustering.R` and includes the following key methods:

- **affinity** : creates an affinity matrix
- **simFunction** : similarity function
- **similarity** : creates a similarity matrix from a similarity kernel
- **specClusteringKM** : method for assigning clusters using k-means
- **specClusteringQR** : method for assigning clusters based on cosine similarities using QR decomposition with pivoting

Figure 2: Graphical comparison of kmeans vs spectral clustering



Details of Gaussian-similarity-kernel and k-means clustering implementation

First, we compute the simmilarity between citations from the TF-IDF or NTF matrix of citations. We use a Gaussian kernel as a measure of similarity. Then, we create an undirected graph based on the similarities to extract some manifold in the data, we thereby obtain A, the affinity matrix. After, we calculate the degree matrix D (diagonal) where each diagonal value is the degree of the respective vertex (*e.g.* sum of rows).

We compute the unnormalized graph Laplacian: $U = D - A$. Then, assuming that we want k clusters, we find the k smallest eigenvectors of U. This represents the points in a new k-dimensional space with low-variance. Finally, in this transformed space, it becomes easy for a standard k-means clustering to find the appropriate clusters.

Gram Matrix

From the TF-IDF or NTF matrix of citations, we compute the cosine similarity between each citation vectors as follows:

$$similarity = cos(\theta) = \frac{a \cdot b}{\|a\| \, \|b\|} = \frac{\sum\limits_{i=1}^{n} a_i b_i}{\sqrt{\sum\limits_{i=1}^{n} a_i^2} \sqrt{\sum\limits_{i=1}^{n} b_i^2}}$$

This matrix is called the **Gram matrix** A. In the first step of the algorithm, we determine the k largest eigenvectors of A: X_k, a n-by-k matrix. Each row of X_k corresponds to a citation vector. Then, we compute the **QR decomposition with column pivoting** applied to ${X_k}^T$, *e.g.* we find the matrices P (permutation matrix, n-by-n), Q (orthogonal, k-by-k) and R (left-upper-triangular, k-by-n), so that:

$${X_k}^T P = QR = Q[R_{11}, R_{12}]$$

R_{11} will be the k-by-k upper-triangular matrix. We then compute the matrix $\hat{R}$:

$$\hat{R} = R_{11}^{-1} R P^T = R_{11}^{-1}[R_{11}, R_{12}]P^T = [I_k, R_{11}^{-1} R_{12}]P^T$$

Finally, the cluster membership of each citation vector is determined by the row index of the largest element in absolute value of the corresponding column of $\hat{R}$.

Example study using methods in our `lib/SpectralClustering.R` script

For reproducibility, we avoid computing k-means in this `Rmd` file because of running time. We run this study on the Normalized Term Frequencies (NTF) and the TF-IDF objects we extract in the `lib/feature_extraction.R` script.

```r
source(file.path(projDir,"lib","SpectralClustering.R"))
source(file.path(projDir,"lib","evaluation_measures.R"))

# estimated time 6 sec
start.time <- Sys.time()

num_authors <- length(unique(exText$AuthorID))
author_id <- exText$AuthorID
exDTM <- createCitationMatrix(exText)

exTFIDF <- weightTfIdf(exDTM, normalize = FALSE)
exNTF <- weightTf(exDTM)

exResultsTFIDF <- runSpectralClusteringQR(exTFIDF, num_authors)
exResultsNTF <- runSpectralClusteringQR(exNTF, num_authors)
```

```
m0 <- matching_matrix(author_id,exResultsTFIDF)
m1 <- matching_matrix(author_id,exResultsNTF)

p0 <- performance_statistics(m0)
p1 <- performance_statistics(m1)

end.time <- Sys.time()
study.time <- end.time - start.time

kable(data.frame(study_time = study.time), format = "markdown")
```

study_time
4.385306 secs

Step 4: View results from sample study

We examine some results from our sample methods to confirm our methodology is producing reasonable results.

```
ex_df <- data.frame(
    study=rep("Agupta", 2),
    method=c("QR Spectral Clustering - TFIDF",
             "QR Spectral Clustering - NTF"),
    precision=c(p0$precision, p1$precision),
    recall=c(p0$recall, p1$recall),
    f1=c(p0$f1, p1$f1),
    accuracy=c(p0$accuracy, p1$accuracy),
    mcc=c(p0$mcc, p1$mcc)
    )

kable(ex_df, format = "markdown", digits = 2)
```

study	method	precision	recall	f1	accuracy	mcc
Agupta	QR Spectral Clustering - TFIDF	0.13	0.25	0.17	0.76	0.05
Agupta	QR Spectral Clustering - NTF	0.11	0.28	0.16	0.70	0.02

Evaluation Metrics

Precision:

$$P = \frac{T_p}{T_p + F_p}$$

Recall:

$$R = \frac{T_p}{T_p + F_n}$$

F1 Score: The F1 score can be interpreted as a harmonic mean of precision and recal. An F1 score reaches its best value at 1 and worst at 0. It is given by:

$$F1 = 2\frac{P \times R}{P + R}$$

Matthew's Correlation Coefficient (MCC): Our group implemented, which returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation. MCC is given by:

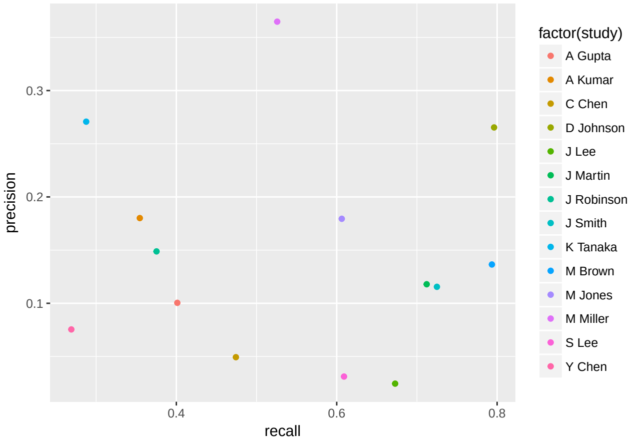
$$MCC = \frac{T_p \times T_n - F_p \times F_n}{\sqrt{(T_p + F_p)(T_p + F_n)(T_n + F_p)(T_n + F_n)}}$$

Step 5: Load results from all studies

Separately, we have run our spectral clustering methods on all of our cleaned and labeled datasets in `output` and saved the results in `output/studies`. [code omitted] We plot the results of our studies below.

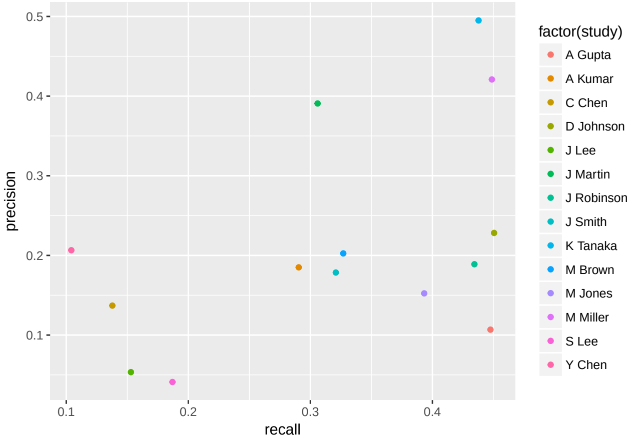
Precision-Recall Scatterplot of Spectral Clustering with QR Decomposition

```
ggplot(data=tb_qr, aes(x=recall, y=precision, col=factor(study))) + geom_point()
```



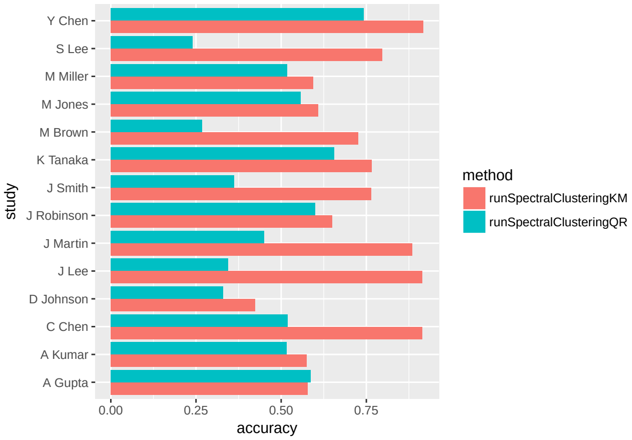
Precision-Recall Scatterplot of Spectral Clustering KMeans

```
ggplot(data=tb_km, aes(x=recall, y=precision, col=factor(study))) + geom_point()
```



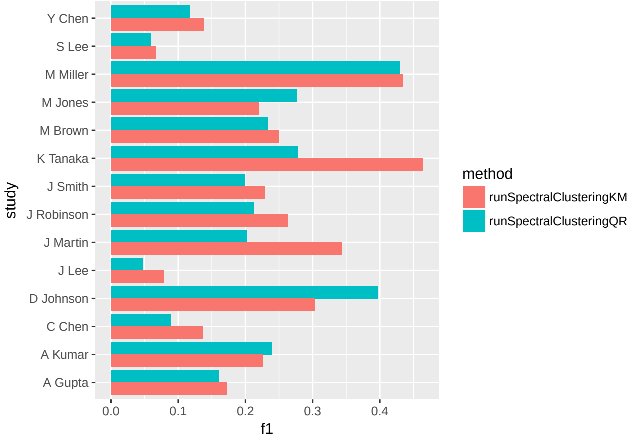
Comparison of Accuracy of Spectral Clustering with QR Decomposition vs Spectral Clustering KMeans

```
ggplot(data=tb_join, aes(x=study, y=accuracy, fill=method)) +
    geom_bar(stat="identity", position=position_dodge()) + coord_flip()
```



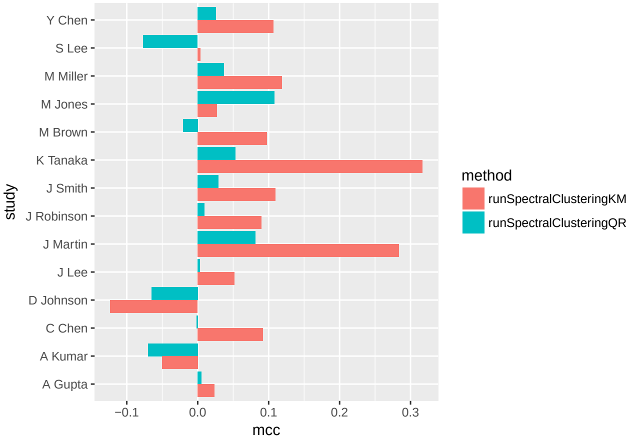
Comparison of F1 score of Spectral Clustering with QR Decomposition vs Spectral Clustering KMeans

```
ggplot(data=tb_join, aes(x=study, y=f1, fill=method)) +
    geom_bar(stat="identity", position=position_dodge()) + coord_flip()
```



Comparison of MCC score of Spectral Clustering with QR Decomposition vs Spectral Clustering KMeans

```
ggplot(data=tb_join, aes(x=study, y=mcc, fill=method)) +
    geom_bar(stat="identity", position=position_dodge()) + coord_flip()
```



General Notes on Spectral Clustering vs K-Means using Term Frequency:

Why TF-IDF? Term frequency counts the number of times a word from a corpus apprears in a particular document. Some words appear frequently in all documents (i.e. they are common locally). Other words appear rarely in the corpus (i.e. they are rare globally). Inverse document frequency weights words that appear rarely more heavily, implying that rare words are more important for determining the cluster assignment of a particular document than words that are common to all documents in a corpus.

Distance measures

Distance measures are the way convergence is measured. The selection of distance measures is very important to consider because different clustering problems require different measures of similarity. In fact, while researching distances, we discovered that there is a textbook called "Encyclopedia of Distances" by Michael Dez and Elena Deza which contains helpful information on the different ways of selecting the correct distance metric for a particular problem. Below are two common measures of distance.

Scaled Euclidean Distance:

$$d(x_i, x_q) = \sqrt{a_1(x_i[1] - x_q[1])^2 + \ldots + a_d(x_i[d] - x_q[d])^2}$$

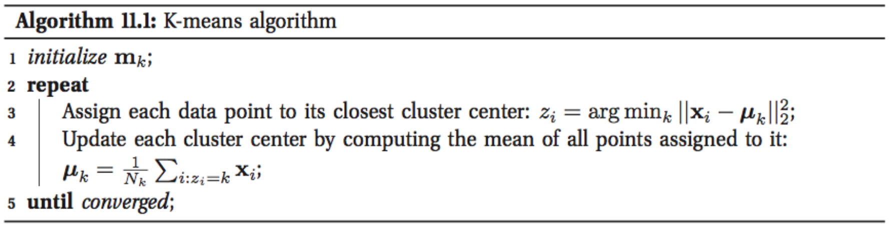
Cosine Similarity:

$$\frac{x_i^T x_q}{\|x_i\| \, \|x_q\|} = cos(\theta)$$

K-means algorithm

1. Initialize cluster centers
2. Assign observations to closest cluster center (hard assignment)
3. Revise cluster centers as mean of assigned observations
4. Repeat steps 1 & 2 until convergence

Figure 3: Kmeans pseudocode



Algorithm 11.1: K-means algorithm

```
initialize $\mathbf{m}_k$;
repeat
    Assign each data point to its closest cluster center: $z_i = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$;
    Update each cluster center by computing the mean of all points assigned to it:
    $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$;
until converged;
```

Complexity of brute-force search

Given a query point, $O(N)$ distance computations per 1-NN query. By extension, $O(N \log k)$ distance computations per k-NN query.

Failure modes of k-means

- disparate cluster sizes
- overlapping clusters
- different shaped / oriented clusters

In the most basic implementation, k-means tries to make cluster assignments of documents using euclidean distances based on distances from cluster centers:

$$z_i \leftarrow \arg\min \left\| u_j - \mathbf{x}_i \right\|_2^2$$

This method is problematic because only the center of the cluster matters. In other words, this method assumes that clusters are spherically symmetric. Although weighted Euclidean distances could be used, we would need to know which weights were appropriate, and even with weights, the clusters would still have the same axis-aligned ellipses.

In the author disambiguation problem, we are faced with a conundrum, namely that kmeans generally fails to give good results when the distance metric used produces ‘overlapping’ clusters. Since authors that share names may work in related fields, this type of disambiguation problem calls for a more probabalistic approach. This is the motivation for using ‘mixture models,’ which enables clustering based on **learned weightings** of features when computing cluster assignments.

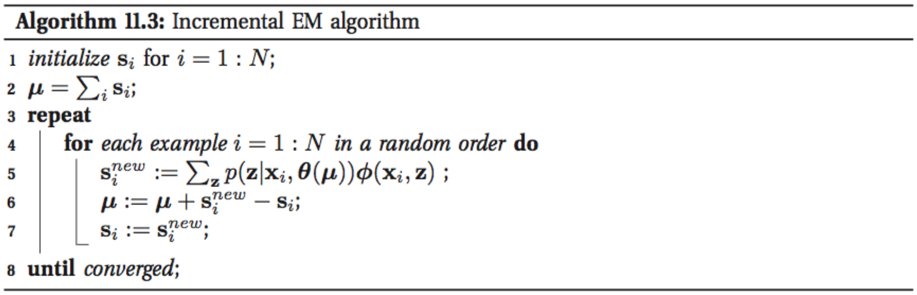
Section II: Paper 6

“A Constraint-Based Probabilistic Framework for Name Disambiguation”

In this paper, Zhang, Tang, Li, and Wang propose a mixture model using the expectations maximization algorithm to perform name disambiguation. They propose a ‘semi-supervised’ framework to “combine the advantages of supervised and unsupervised methods.” What does this mean in practice? First, we randomize the weights we assign to the constraints per `section 3.2`. Then, we try to use EM to minimize our objective function with two weighted constraints. The steps in the EM algorithm can be summarized at a high level as follows:

In the EM algorithm, $p(z_i = k|x_i, \theta)$ represents the posterior probability that point i belongs to cluster k. This is known as the responsibility of cluster k for point i, and can be computed using Bayes rule. This procedure is called *soft clustering*. EM is an iterative algorithm which alternates between optimizing the

Figure 4: EM pseudocode



Algorithm 11.3: Incremental EM algorithm

```
initialize $\mathbf{s}_i$ for $i = 1 : N$;
$\boldsymbol{\mu} = \sum_i \mathbf{s}_i$;
repeat
    for each example $i = 1 : N$ in a random order do
        $\mathbf{s}_i^{new} := \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x}_i, \boldsymbol{\theta}(\boldsymbol{\mu}))\boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z})$ ;
        $\boldsymbol{\mu} := \boldsymbol{\mu} + \mathbf{s}_i^{new} - \mathbf{s}_i$;
        $\mathbf{s}_i := \mathbf{s}_i^{new}$;
until converged;
```

objective function given the parameters (E step), and then optimizing the parameters given the “filled in” data (M step). An illustration of this iterative process is shown below:

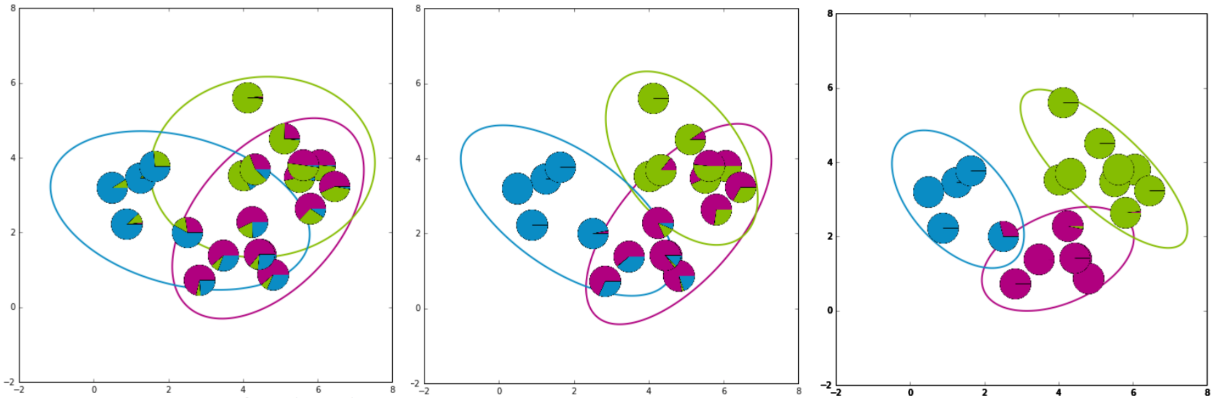


Table 2 contstraints:

- c_2 for any two papers, if they share a co-author, c_2 is 1, otherwise is 0
- c_6 if co-authors of any two papers are are also co-authors on another paper

Constraints: The constraint function is a Boolean-valued function and is defined as follow:

$$c(p_i, p_j) = \begin{cases} 1, & \text{if } p_i \text{ and } p_j \text{ satisfy the constraint } c_l \\ 0, & \text{otherwise} \end{cases}$$

Goal is to minimize the objective function by using EM algorithm. **Objective Function:**

$$\max p(Y|X) \propto \min f_{obj} = \sum_i \sum_j \{D(x_i, x_j) I(l_i \neq l_j) \sum_{c_k \in C} [w_k c_k(p_i, p_j)]\} + \sum_{x_i \in X} D(x_i, y_{l_i}) + \log Z$$

where, $ D(x_i, y_{l_i}) $ is the distance between paper i and researcher y_{l_i} w_k is the weight of for c_k Z is the normalization factor

Step 1: Load and process the data

We assign each of the author datasets to a variable [code omitted]

Step 2: Document Term Matrices

We compute the DTM for paper, co-author, and journal venue using our functions (with pre-processing) [code omitted]

Step 3: TF-IDF

We compute a TF-IDF on each DTM and then bind the TF-IDFs together to create a single TF-IDF object. We then compute cosine similarity.

```
dtm_paper_tfidf <- function(df) {
  tfidf <- TfIdf$new()
  return(fit_transform(dtm_paper(df), tfidf))
}

dtm_coauthor_tfidf <- function(df) {
  tfidf <- TfIdf$new()
  return(fit_transform(dtm_coauthor(df), tfidf))
}

dtm_journal_tfidf <- function(df) {
  tfidf <- TfIdf$new()
  return(fit_transform(dtm_journal(df), tfidf))
}

# We want to combine the three citations together
dtm_tfidf <- function(df) {
  total <- cbind(dtm_paper_tfidf(df), dtm_coauthor_tfidf(df), dtm_journal_tfidf(df))
  return(total)
}

## compute cosine similarities matrix (Gram Matrix)
docsdissim <- function(df) {
  cosSparse(t(dtm_tfidf(df)))
}
```

Step 4: Clustering

Following suggestion of Zhang, Tang, Li, and Wang, we carry out hierarchical clustering method under the cosine distance as the *baseline method*. The number of clusters is *assumed* to be known as stated in the paper (i.e. the method is 'semi-supervised').

The improved model developed in the paper is the EM algorithm applied on the objective function, with two constraints coauthor and tau-coauthor (objective function and constraints are referenced above). In order to implement the whole optimization process, we wrote up the objective function and EM algorithm with E step focused on optimizing assignments for papers and M step on tuning parameters used in objective function. Due to the complexity of high-dimension matrix computation, the M step of the EM algorithm as well as the c_6 constraint are extremely time-consuming and thus for reproducibility in this `Rmd` file we don't compute the result here.

```
basemodel <- function(df) {
  cossim <- docsdissim(df)
  rownames(cossim) <- c(1:nrow(dtm_tfidf(df)))
  colnames(cossim) <- c(1:nrow(dtm_tfidf(df)))

  h <- hclust(as.dist(cossim), method = "ward.D")
  result_hclust <- cutree(h,length(unique(df$AuthorID)))
  return(result_hclust)
}

basemodel_AGupta <- basemodel(AGupta)
basemodel_AKumar <- basemodel(AKumar)


# EM clustering
db <- c(as.character(AGupta$Coauthor), as.character(AKumar$Coauthor), as.character(CChen$Coauthor),
        as.character(DJohnson$Coauthor), as.character(JLee$Coauthor), as.character(JMartin$Coauthor),
        as.character(JRobinson$Coauthor), as.character(JSmith$Coauthor),
        as.character(KTanaka$Coauthor), as.character(MBrown$Coauthor), as.character(MJones$Coauthor),
        as.character(MMiller$Coauthor),as.character(SLee$Coauthor), as.character(YChen$Coauthor))

source("lib/functions_paper6.R")

k <- length(unique(JMartin$AuthorID))
x_jmt <- docsdissim(JMartin)
y_ini <- sample(1:k, length(x_jmt[1,]), replace = T)
l_jmt <- matrix(NA, ncol = ncol(x_jmt), nrow = k)
for(i in unique(JMartin$AuthorID)) {
  l_jmt[i,] <- mean(x_jmt[which(i==JMartin$AuthorID), ])
}
y_jmt <- matrix(NA, ncol = ncol(x_jmt), nrow = length(x_jmt[1,]))
for (i in 1:length(x_jmt[1,])) {
  y_jmt[i, ] <- l_jmt[y_ini[i], ]
}
A <- diag(1, nrow = nrow(x_jmt))
EM_algorithmFast(X = x_jmt, Y = y_jmt, l = l_jmt, df = JMartin)
```

Step 5: Performance

We recently finished our implementation of the EM algorithm, therefore the result is still computing.

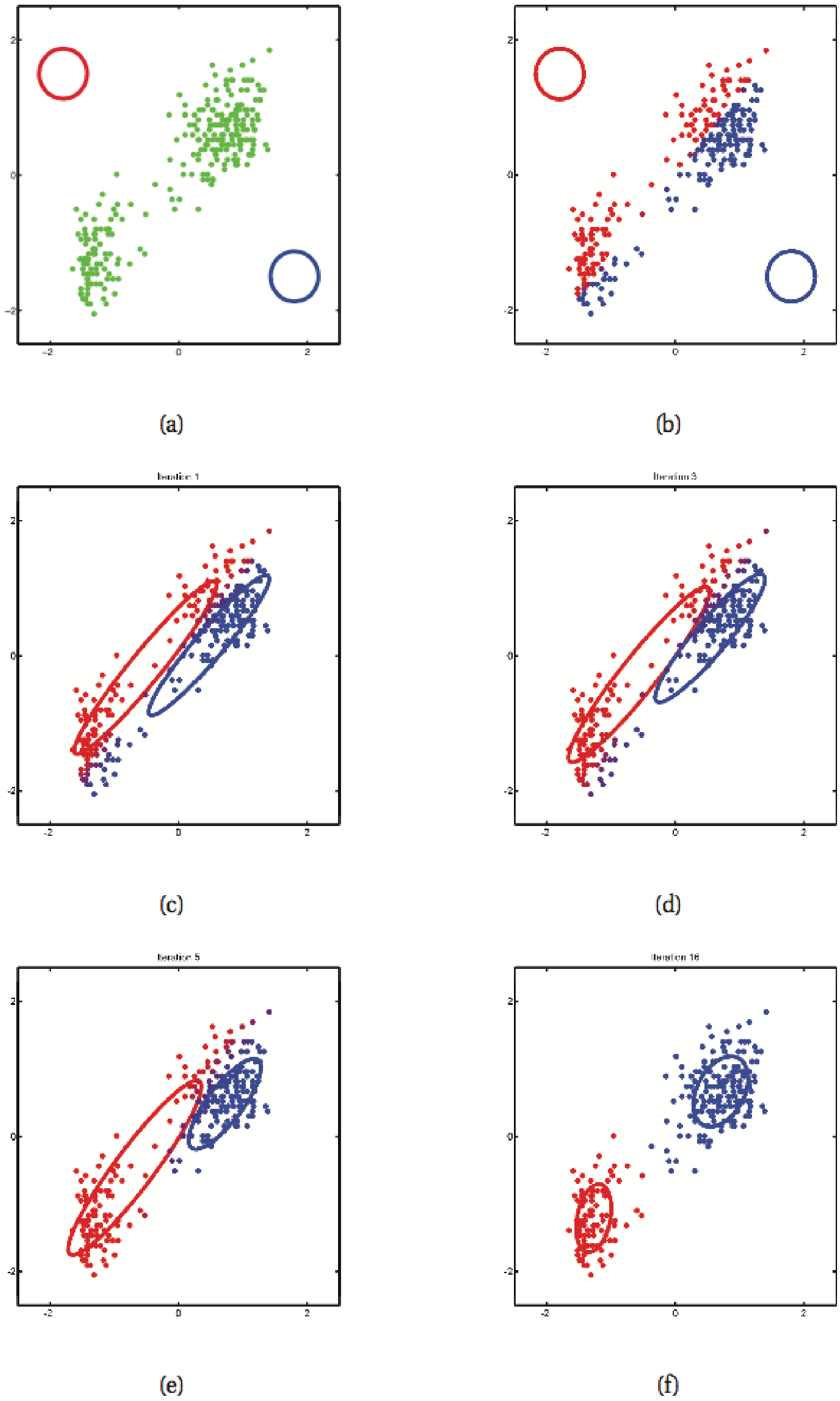
Computational Time: the EM algorithm requires a significant amount of computing resources and time to converge because of high-dimensional matrix computation. Below is the code used to implement our EM algorithm:

```
# All functions used for paper 6

### The EM algorithm being very slow, we tried to improve the performance
### by improving the implementation of the E and M steps.
### As a conclusion, we increased significantly the M step that is not a
### bottleneck anymore. However, even if faster, E step is not reasonably
### computable for n>30 (smallest dataset being 112...).

# to compute gain in performance (if any)
gainPerf <- function(new,old) sprintf("Gain from %s to %s: %s",round(old[3],4), round(new[3],4), round(
#differential function

dif_dist<-function(x_i,x_j,A,m){
  #take derivative of a_mmorm_i
  part1<-x_i[m]*x_j[m]*sqrt(t(x_i)%*%A%*%x_i)*sqrt(t(x_j)%*%A%*%x_j)
  part2<-(t(x_i)%*%A%*%x_j)*((x_i[m]^2)*(t(x_i)%*%A%*%x_i)
                             +(x_j[m]^2)*(t(x_j)%*%A%*%x_j))/(2*sqrt(t(x_i)%*%A%*%x_i)*sqrt(t(x_j)%*%A%
  part3<-(t(x_i)%*%A%*%x_i)*(t(x_j)%*%A%*%x_j)
  return((part1-part2)/part3)
}

# use crossprod(diag(A),x_i^2) <=> t(x_i)%*%A%*%x_i (twice faster)
dif_distFast<-function(x_i,x_j,A,m){
  #take derivative of a_mmorm_i
  norm2i <- crossprod(diag(A),x_i^2)
  norm2j <- crossprod(diag(A),x_j^2)
  prodNorms <- norm2i*norm2j
  return((x_i[m]*x_j[m]*sqrt(prodNorms)-(t(x_i)%*%A%*%x_j)*((x_i[m]^2)*norm2i+(x_j[m]^2)*norm2j)/(2*sqr
}

# Test increase in perf (divide by 3)
#gainPerf(system.time(dif_distFast(X[10,],Y[5,],A,5)),system.time(dif_dist(X[10,],Y[5,],A,5)))

dif_func<-function(X,Y,A,df,m){
  n<-nrow(X)
  sum<-0
  for (i in 1:n){
    for (j in i:n-1){
      sum<-sum+dif_distFast(X[i,],X[j+1,],A,m)*
        #(0.7*c_2(i,j,df)+c_6(i,j,M))+
        (0.7*c_2(i,j+1,df))+
        dif_distFast(X[i,],Y[j+1,],A,m)
    }
  }
  return(sum)
}

dif_funcFast<-function(X,Y,A,df,m){
  n<-nrow(X);  sum<-0;
  # for x and xx
  norm2x <- apply(matrix(1:n), 2, FUN=function(j)crossprod(diag(A),X[j,]^2))[[1]]@x#
  prodNormXX <- outer(norm2x,norm2x)#
  ximxjm <- outer(X[,m],X[,m])#
  xiAxj <- t(X)%*%A%*%X/2#
  xi2Ovnorm2xj <- matrix(rep(X[,m]^2,n),nrow=n)/t(matrix(rep(norm2x,n),nrow=n))
  xj2Ovnorm2xi <- t(matrix(rep(X[,m]^2,n),nrow=n))/matrix(rep(norm2x,n),nrow=n)
  # for xy
  norm2y <- apply(matrix(1:n), 1, FUN=function(j)crossprod(diag(A),Y[j,]^2))
  prodNormXY <- outer(norm2x,norm2y)
  ximyjm <- outer(X[,m],Y[,m])
  xiAyj <- t(X)%*%A%*%Y/2
  xi2Ovnorm2yj <- matrix(rep(X[,m]^2,n),nrow=n)/t(matrix(rep(norm2y,n),nrow=n))
  yj2Ovnorm2xi <- t(matrix(rep(Y[,m]^2,n),nrow=n))/matrix(rep(norm2x,n),nrow=n)
  # df/damm = sum(sum( upper.triangle(dDxx/damm*0.7*c2+dDxy/damm) ))
  # to calculate sum of dif_dist(X[i,],X[j+1,],A,m)*(0.7*c_2(i,j+1,df))+dif_dist(X[i,],Y[j+1,],A,m) for
  dDxx <- (ximxjm - xiAxj*(xi2Ovnorm2xj-xj2Ovnorm2xi) )/sqrt(prodNormXX)
  dDxy <- (ximyjm - xiAyj*(xi2Ovnorm2yj-yj2Ovnorm2xi) )/sqrt(prodNormXY)
  c2 <- matrix(NA,nrow=n,ncol=n)
  for (i in 1:n) {
    c2[i,] <- apply(matrix(1:n),1,FUN=function(j)c_2(i,j,df))
  }
  #sum <- apply(matrix(1:(n-1)),1,FUN=function(i)sum(apply(matrix((i+1):n),1,FUN=function(j)X[i,m]*X[j,
  return(sum((dDxx*.7*c2+dDxy)[upper.tri(dDxx, diag = FALSE)]))
}

# test performance (divide by sqrt(n))
#n=80; X = x_agu[1:n,1:n]; Y = y_agu[1:n,1:n]; df = AGupta[1:n,]; A <- diag(1+rnorm(n)/4, nrow = n, nco
#gainPerf(system.time(dif_funcFast(X,Y,A,df,m)),system.time(dif_func(X,Y,A,df,m)))


#-----------------------------------------------------------------
# Constraint Function:
#---
# p_i is the ith paper tagged to Principle author a
# p_j is the jth paper tagged to Principle author a
# df contains all information about Principle author a
#---

c_2 <- function(p_i, p_j, df) {
  a <- strsplit(as.character(df$Coauthor[p_i]), "; ")[[1]]
  b <- strsplit(as.character(df$Coauthor[p_j]), "; ")[[1]]
  return(ifelse(any(a %in% b)==T, 1, 0))
}


# Take AGupta for example
# AGupta <- read.csv("../output/Agupta.csv")
#
#c_2(6,7,AGupta)
#c_2(6,10,AGupta)



# part 4.2
# Union set of all pi.authors with author named a.
#---
# input: dataframe of author named a
#---
union_author <- function(df) {
  return(unique(unlist(strsplit(as.character(df$Coauthor), split = ";"))))
}

# Take AGupta for example, get union of unique values of authors.
uni <- union_author(AGupta)

# Matrix Mp
```

```r
M_p <- function(df){
  return(diag(1, nrow = nrow(df)))
}
# Matrix Mpa
M_pa <- function(a) {
  authors <- union_author(a)
  mat_pa <- matrix(NA, nrow = nrow(a), ncol = length(authors))
  for (i in 1:nrow(mat_pa)) {
    for (j in 1:ncol(mat_pa)) {
      mat_pa[i, j] <- ifelse(grepl(authors[j], a$Coauthor[i])==T, 1, 0)
    }
  }
  return(mat_pa)
}

x <- M_pa(AGupta)

# Matrix Map
M_ap <- function(a) {
  return(t(M_pa(a)))
}

# Matrix Ma
#---
# Input a is the author name
# First get all publications with an author named a.
# Then use union_author() to get union set of all pi.authors

## here we also need a database (called db) that has all publication information including coauthors.
## ( basically it's the rbind of all 14 datasets, with 6 columns. What we care is the coauthor column)
M_a <- function(a) {
  all_author <- union_author(a)
  mat_a <- matrix(NA, nrow = length(all_author), ncol = length(all_author))
  for (i in all_author) {
    for (j in all_author) {
      a_index <- grep(i, db, value = F)
      b_index <- grep(j, db, value = F)
      mat_a[i, j] <- ifelse(any(a_index %in% b_index), 1, 0)
    }
  }
  return(mat_a)
}

# For illustration, run following code:
# a <- grep(uni[50], AGupta$Coauthor, value = F)
# b <- grep(uni[55], AGupta$Coauthor, value = F)
# any(a %in% b)

# Now write up matrix M!
M <- function(a) {
  m_p <- M_p(a)
  m_pa <- M_pa(a)
  m_ap <- t(m_pa)
  m_a <- M_a(a)
  upper <- cbind(m_p, m_pa)
  lower <- cbind(m_ap, m_a)
  return(m = rbind(upper, lower))
}

#constraint 6
c_6<-function(i,j,M){
  k<-0
  while((M[i,j]!=1)&(k<20)){
    M<-M%*%M
    k<-k+1
  }
  return(0.7^k)
}

#-------------------------
# Distance Function

# First write a function to calculate norm of a given vector xi
# Since in the partical derivative part we repeatly use the norm
# and matrix A updates in each iteration

#---
# Input : x: feature vector subtracted from the data. (Generated use the paper title, journal name, coa
#         A: Parameter matrix. Initial value if A is identity matrix, updated along with algorithm
#---
norm_feature <- function(x, A) {
  return(sqrt(t(x)%*%A%*%x))
}


dist_feature <- function(x_i, x_j, A) {
  a <- t(x_i) %*% A %*% x_j
  b <- norm_feature(x_i, A) * norm_feature(x_j, A)
  return(1-a/b)
}

#objective function
obj_func<-function(X,Y,df){
  # X is the matrix of feature of all papers.
  # df is the list of all articles.
  # Y is the matrix of researchers for all papers.
  n<-nrow(X)
  sum<-0
  for (i in 1:n){
    for (j in i:n){
      if (any(Y[i,]!=Y[j,])){
        sum<-sum+dist_feature(X[i,],X[j,],A)*
          #(0.7*c_2(i,j,df)+c_6(i,j,df))
          (0.7*c_2(i,j,df))
      }
    }
    sum<-sum+dist_feature(X[i,],Y[i,],A)
  }
  return(sum)
}

obj_funcFast<-function(X,Y,df){
  # X is the matrix of feature of all papers.
  # df is the list of all articles.
  # Y is the matrix of researchers for all papers.
  n<-nrow(X)
  # for x and xx
  norm2x <- apply(matrix(1:n), 2, FUN=function(j)crossprod(diag(A),X[j,]^2))[[1]]@x
  prodNormXX <- outer(norm2x,norm2x)
  xiAxj <- t(X)%*%A%*%X/2
  # for xy
  norm2y <- apply(matrix(1:n), 1, FUN=function(j)crossprod(diag(A),Y[j,]^2))
  prodNormXY <- outer(norm2x,norm2y)
  xiAyj <- t(X)%*%A%*%Y/2

  YiYj <- matrix(0, nrow=n, ncol=n)
  for (i in 1:n){
    for (j in i:n){
      if (any(Y[i,]!=Y[j,])) YiYj[i,j] <- 1;
    }
  }
  dXiXj <- 1 - xiAxj/sqrt(prodNormXX)
  dXiYj <- 1 - xiAyj/sqrt(prodNormXY)
  c2 <- matrix(NA,nrow=n,ncol=n)
  for (i in 1:n) {
    c2[i,] <- apply(matrix(1:n),1,FUN=function(j)c_2(i,j,df))
  }
  return(sum(sum(dXiXj*YiYj*0.7*c2))+sum(diag(dXiYj)))
}

# increase in performance (divide computation time by approximatively sqrt(n)/2)
# gainPerf(system.time(obj_funcFast(X,Y,df)),system.time(obj_func(X,Y,df)))

# EM framework
# Initialization of the EM Framework:
# First cluster publications into disjoint groups based on the constraints over them.
#i.e.if two publications have a constraint, then they are assigned to the same researcher.
# After consideration, we randomly assign tags for papers as initial values.

init_tag <- function(df) {
  return(sample(1:length(unique(df$AuthorID)), nrow(df), replace = T))
}

switchYiLj <- function(Y,l,i,j) {
  if (i==1) {return(rbind(l[j,],Y[(i+1):length(Y[,1]),]));}
  else if (i==length(Y[,1])) return(rbind(Y[1:(i-1),],l[j,]))
  else return(rbind(Y[1:(i-1),],l[j,],Y[(i+1):length(Y[,1]),]))
}
#-------------
```

```r
# The input of EM algorithm is the tags we assign to every paper.
# And we try to use EM steps to minimize the objective function
#-----------

#EM algorithm
#l is the matrix of all author
#Y is the matrix of each paper's author
EM_algorithm <- function(X, Y, l, df) {
  k<-max(df$AuthorID)
  n<-nrow(X)
  numb<-0
  A <- diag(1+rnorm(n)/4, nrow = n, ncol = n)
  while(numb<1000){
    Y_stop<-Y
    #E-step
    for(i in 1:n){
      print(i)
      Y_tem<-Y
      tem<-NULL
      for (j in 1:k){
        Y_tem[i,]<-l[j,]
        tem[j]<-obj_func(X,Y_tem,df)
      }
      Y[i,]<-l[which.min(tem),]
      numb<-numb+1
    }
    if(obj_func(X,Y,df)==obj_func(X,Y_stop,df)) break
    #M-step
    for (i in 1:k){
      sum_x<-0
      c<-NULL
      for(j in 1:n){
        if (all(Y[j,]==l[i,])) {
          sum_x<-sum_x+X[j,]
          rbind(c,j)
        }
      }
      l[i,]<-sum_x/sqrt(t(sum_x)%*%A%*%sum_x)
      Y[c,]<-l[i,]
    }
    for(m in 1:n){
      print(m)
      A[m,m]<-A[m,m]+0.75*dif_func(X,Y,A,df,m)
    }
  }
}

EM_algorithmFast <- function(X, Y, l, df) {
  k<-max(df$AuthorID)
  n<-nrow(X)
  numb<-0
  A <- diag(1+rnorm(n)/4, nrow = n, ncol = n)
  while(numb<3){
    print(sprintf("Start step %s",numb+1))
    Y_stop<-Y
    # E-step
    for(i in 1:n){
      print(sprintf("E-step: %.2s%%",100*i/n))
      Y_tem<-Y
      tem<-apply(matrix(1:k),1,FUN=function(j)obj_funcFast(X,switchYiLj(Y,l,i,j),df))
      Y[i,]<-l[which.min(tem),]
      numb<-numb+1
    }
    if(obj_funcFast(X,Y,df)==obj_funcFast(X,Y_stop,df)) break
    # M-step
    for (i in 1:k){
      print(sprintf("M-step: %.2s%%",100*i/k))
      sum_x<-0
      c<-NULL
      for(j in 1:n){
        if (all(Y[j,]==l[i,])) {
          sum_x<-sum_x+X[j,]
          rbind(c,j)
        }
      }
      l[i,]<-sum_x/sqrt(t(sum_x)%*%A%*%sum_x)
      Y[c,]<-l[i,]
    }
    for(m in 1:n){
      print(m)
      A[m,m]<-A[m,m]+0.75*dif_funcFast(X,Y,A,df,m)
    }
  }
}
```

Figure 5: Convergence of EM algorithm: evolution of cluster responsibilities


(a)
(b)
Iteration 1
(c)
Iteration 3
(d)
Iteration 5
(e)
Iteration 16
(f)