Sandia_Seurat_Test

Benjamin Mellin

2025-02-05

MSC Dataset Clustering Analysis

- The purpose of this code notebook is to test a cluster based classification on Mesenchymal Stromal Cells mRNA sequencing data (MSCs)
- Essentially, each MSC has an experimentally determined label of antimicrobial/nonantimicrobial.
- We are aiming to create a unsupervised clustering based classification method for this data.
- The method for this is the following:

1. Divide the data in 5 folds.
2. Use Leiden clustering on 4/5 of the folds.
3. Label clusters that are dominated by antimicrobial or non antimicrobial MSC cells as antimicrobial or non antimicrobial. Remove remaining clusters
4. Classify remaining cells based on Euclidean distance to each cluster.
5. Compare true label to cluster assigned label, get accuracy.
6. Repeat steps 2-5 on remaining folds.

- The purpose of using this cluster based method is to try to develop an understanding of the cellular features (i.e. gene expression signatures) that may underlie antimicrobial/non antimicrobial behavior.
- This particular dataset is heterogeneous and high noise, which is why we hypothesize we are trying this method.
- This methodology was suggested by Dr. Mansoor Haider.
- Some of this code has been adapted from previous work by Dr. Raga Krishnakumar.

Note: to Run this, you must pip install leiden alg. To do this I used the following commands in the console: reticulate::py_config() reticulate::py_install("leidenalg")

Clear workspace

```r
rm(list=ls(all.names=TRUE))
gc()
```

```
##          used (Mb) gc trigger (Mb) limit (Mb) max used (Mb)
## Ncells 496446 26.6    1068261 57.1         NA   700311 37.5
## Vcells 936470  7.2    8388608 64.0      16384  1963572 15.0
```

###Read in packages

```r
library(Seurat)
```

```
## Loading required package: SeuratObject

## Loading required package: sp

## 'SeuratObject' was built under R 4.4.0 but the current version is
## 4.4.2; it is recomended that you reinstall 'SeuratObject' as the ABI
## for R may have changed

## 'SeuratObject' was built with package 'Matrix' 1.7.0 but the current
## version is 1.7.2; it is recomended that you reinstall 'SeuratObject' as
## the ABI for 'Matrix' may have changed

##
## Attaching package: 'SeuratObject'

## The following objects are masked from 'package:base':
##
##     intersect, t
```

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

###Read in seurat object

```r
msc <- readRDS(file="/Users/benjaminmellin/Desktop/Grad_School/Year 1/fds_project/AllData_COMBAT_CCNorm

#clearing the meta data so we can cluster and get UMAP Reductions ouselves.

msc@meta.data$RNA_snn_res.0.5 <- NULL
msc@meta.data$seurat_clusters <- NULL

msc@reductions <- list() #clearing slots that store NN embeddings and UMAP information
msc@graphs <- list()
```

From Raga scRNA_COMBAT.R

- Running Raga's code to get UMAP reductions
- This has all of her parameters
- Commented code was present in original analysis but is not used here.

```
#msc<-FindVariableFeatures(msc, selection.method = "vst", nfeatures = 1000)
msc <- RunPCA(msc, npcs = 20, verbose = FALSE, features = VariableFeatures(object = msc))
# t-SNE and Clustering
msc <- RunUMAP(msc, reduction = "pca", dims = 1:10,n.neighbors = 10,n.components = 10, min.dist = 0.05,
                   local.connectivity = 5)
```

```
## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to the R
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation'
## This message will be shown once per session

## 17:35:30 UMAP embedding parameters a = 1.75 b = 0.8421

## 17:35:30 Read 9938 rows and found 10 numeric columns

## 17:35:30 Using Annoy for neighbor search, n_neighbors = 10

## 17:35:31 Building Annoy index with metric = cosine, n_trees = 50

## 0%   10   20   30   40   50   60   70   80   90   100%

## [----|----|----|----|----|----|----|----|----|----|

## **************************************************|
## 17:35:33 Writing NN index file to temp file /var/folders/dw/n4jtvcxx0gl3p_mb684zhpq00000gn/T//RtmpVx
## 17:35:33 Searching Annoy index using 1 thread, search_k = 1000
## 17:35:35 Annoy recall = 100%
## 17:35:35 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 10
## 17:35:35 9938 smooth knn distance failures
## 17:35:35 Initializing from normalized Laplacian + noise (using RSpectra)
## 17:35:36 Commencing optimization for 500 epochs, with 73422 positive edges
## 17:35:53 Optimization finished
```

```
# Commented this out so we can do it later.
# msc <- FindNeighbors(msc, reduction = "pca", dims = 1:20)
# msc <- FindClusters(msc, resolution = 0.5)
# commented this out ^ to do it later

# print(msc[["pca"]], dims = 1:5, nfeatures = 5)
# VizDimLoadings(msc, dims = 1:2, reduction = "pca")
# DimPlot(msc, reduction = "umap", group.by = "orig.ident",cols=c('darkviolet','darkblue','blue2','deep
#                                                                   'darkred','brown3','darkorange','
```

More plotting. Not important.

```
# umap_plot<-DimPlot(msc, reduction = "umap", group.by = "label_ident",cols=c('darkviolet','darkblue','
#                                                                            'darkred','brown3','darkorange','
#
# ggsave("umap_plot.png", plot = umap_plot, width = 6, height = 5, dpi = 300)
#
#
# # DimPlot(msc, reduction = "umap", group.by = "seurat_clusters")
```

View meta-data

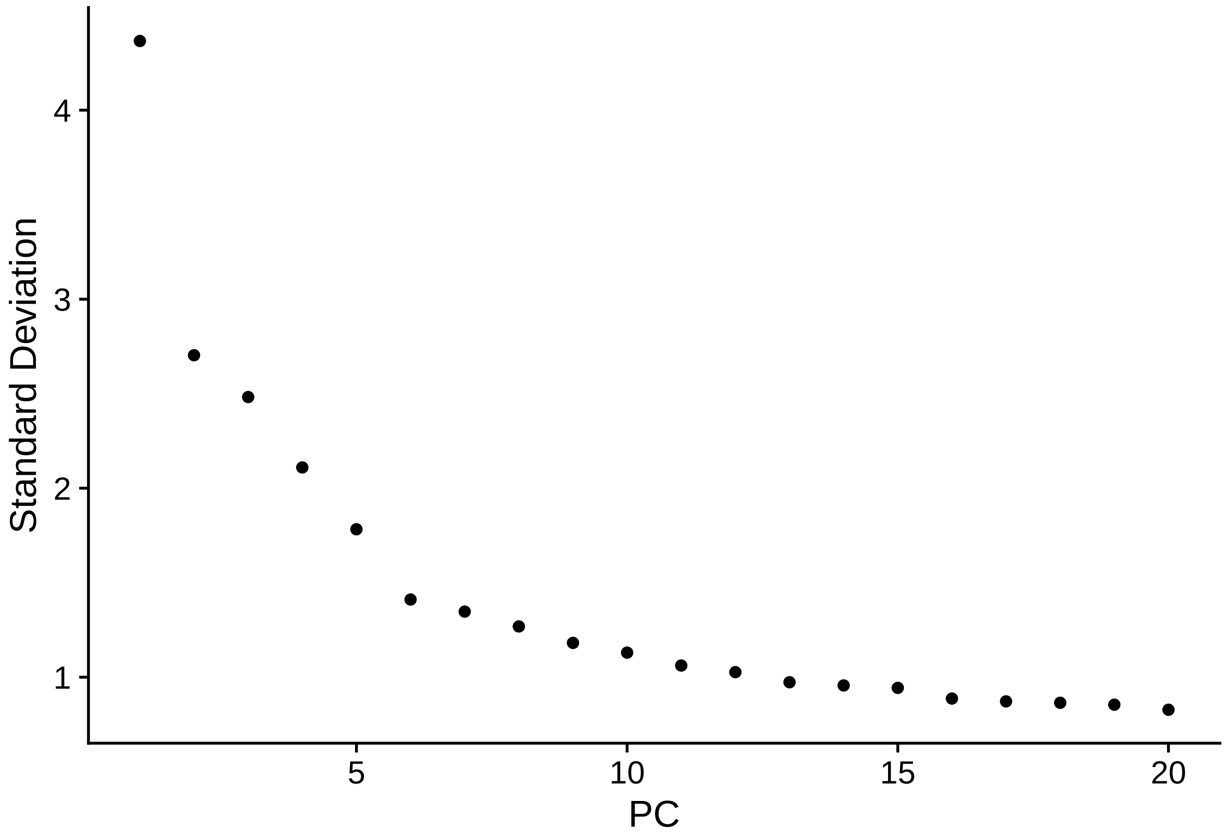
Below here is from https://satijalab.org/seurat/articles/msc3k_tutorial

Now we must assign label to cell.

```
msc$label_ident <- ifelse(substr(msc$orig.ident, 1, 5) == "bcScr", "suboptimal", "antimicrobial")
```

Elbow plot

```
ElbowPlot(msc)
```



- I will choose to use 15 principal components here, this is what Raga used. - The Elbow is at around 5 or 6, but I'm going to go with what has been used previously with this data set.

Find the percentage of antimicrobial / suboptimal by cluster

First, I will cluster here

```
set.seed(123)
msc <- FindNeighbors(msc, reduction = "pca", dims = 1:15)
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

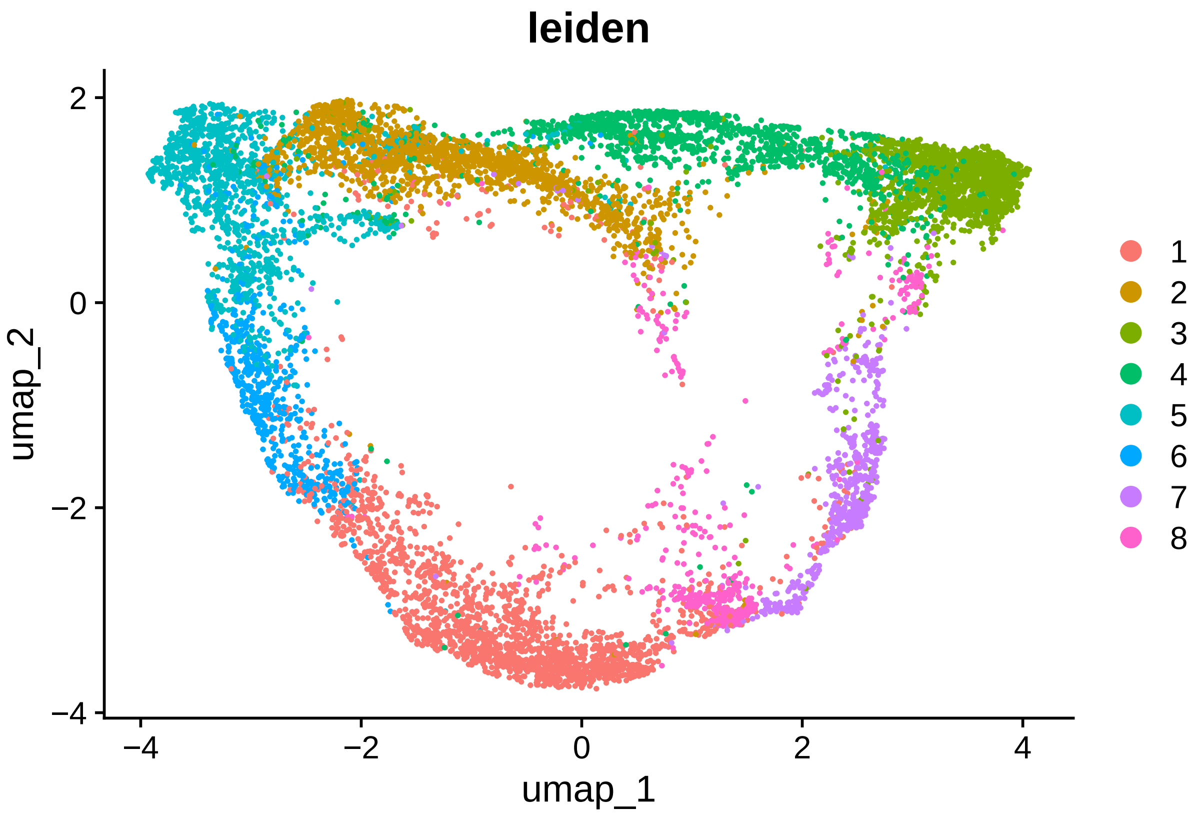
```
head(msc@meta.data)
```

```
##                        orig.ident nCount_RNA nFeature_RNA    S.Score   G2M.Score
## b6_AAACGCTGTAATGCGG-1 b6ScrNoLPS       4938         1828 -0.1223846 -0.84355509
## b6_AAAGAACTCAACTGGT-1 b6ScrNoLPS       6579         2446 -0.2113997 -1.02650728
## b6_AAAGAACTCTCATTAC-1 b6ScrNoLPS      15309         4113 -0.1372655  2.43762994
## b6_AAAGGTACAGAACGCA-1 b6ScrNoLPS      11700         3177 -0.3217893 -1.81756757
## b6_AAAGTCCAGACATAAC-1 b6ScrNoLPS       8912         3187  0.1165224 -1.33108108
## b6_AAATGGAAGAGTGTTA-1 b6ScrNoLPS       2358         1210 -0.0879329  0.02650728
##                       Phase  old.ident percent.mt percent.RPS percent.RPL
## b6_AAACGCTGTAATGCGG-1    G1 b6ScrNoLPS  11.563386    8.525719   10.976104
## b6_AAAGAACTCAACTGGT-1    G1 b6ScrNoLPS   9.180727    8.876729   10.503116
## b6_AAAGAACTCTCATTAC-1   G2M b6ScrNoLPS   8.694232    6.701940    8.746489
## b6_AAAGGTACAGAACGCA-1    G1 b6ScrNoLPS   9.495726   10.358974   12.512821
## b6_AAAGTCCAGACATAAC-1     S b6ScrNoLPS   3.803860    7.405745   10.334381
## b6_AAATGGAAGAGTGTTA-1   G2M b6ScrNoLPS   7.591179    8.312129   11.747243
##                       CC.Difference   label_ident
## b6_AAACGCTGTAATGCGG-1     0.7211705 antimicrobial
## b6_AAAGAACTCAACTGGT-1     0.8151076 antimicrobial
## b6_AAAGAACTCTCATTAC-1    -2.5748954 antimicrobial
## b6_AAAGGTACAGAACGCA-1     1.4957782 antimicrobial
## b6_AAAGTCCAGACATAAC-1     1.4476034 antimicrobial
## b6_AAATGGAAGAGTGTTA-1    -0.1144402 antimicrobial
```

```
msc$leiden <- FindClusters(msc, resolution = 0.5, random.seed=123, algorithm=4)$seurat_clusters
```

Let's view the leiden clusters in our dimplot:

```
leiden_umap<-DimPlot(msc, reduction = "umap", group.by = "leiden",pt.size=.2)
ggsave("leiden_umap.png", plot = leiden_umap, width = 6, height = 5, dpi = 300)
leiden_umap
```



Dominance by cell classification in clusters

```
contingency_table <- table(msc$leiden, msc$label_ident)

percentage_table <- prop.table(contingency_table, margin = 1) * 100

print(percentage_table)
```

```
##
##     antimicrobial suboptimal
##   1     0.6542526 99.3457474
##   2    40.4883589 59.5116411
##   3    98.0428135  1.9571865
##   4    99.5501285  0.4498715
##   5    79.1634981 20.8365019
##   6     5.1083591 94.8916409
##   7     5.5649241 94.4350759
##   8    22.4719101 77.5280899
```

5 Fold CV cluster based classification method

Now that we know that clustering works well, lets try and do Dr. Haider's method of classification. First, I will randomly divide the dataset up into 5 folds

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
set.seed(123)
folds <- createFolds(1:ncol(msc), k = 5, list = TRUE)
```

Now I will create a method that: 1. Leiden Unsupervised clusters 4/5 folds. 2. Keeps only clusters that have a split of at least 80/20 3. Calculate the centroids of the remaining clusters. 4. Determine which cluster each remaining cell is closest to. 5. Classify the cells based on their closest cluster. 6. Determine how accurate the classification is.

First things first: I assign the folds to the seurat object

```
msc$fold <- rep(NA, ncol(msc))
for (i in 1:5) {
  msc$fold[folds[[i]]] <- i  # Assign fold number to cells in each fold
}
```

Method outline:

Following method takes: training folds Returns: clusters centroids for only extreme clusters (more extreme than 80/20 split) Description: The following method takes a seurat object, ideally your training folds, and then it performs unsupervised clustering (leiden). Then, it finds the average 15 principal components of each cluster. Then it gets rid of clusters that are less extreme than 80/20. Then it returns a dataframe of the centroids.

```
get_rep_centroids<-function(train_folds){

  # we cluster here from our training folds
  set.seed(123)
  train_folds <- FindNeighbors(train_folds, dims = 1:15)
  train_folds$leiden <- FindClusters(train_folds, resolution = 0.5, random.seed=123, algorithm=4)$seura
  pca_coords <- train_folds[["pca"]]@cell.embeddings[, 1:15]

  # Intialize dataframe to store PCs
  clusters <- train_folds$leiden
  centroids <- data.frame(cluster = integer(0),
                          PC1 = numeric(0), PC2 = numeric(0),
                          PC3 = numeric(0), PC4 = numeric(0),
                          PC5 = numeric(0), PC6 = numeric(0),
                          PC7 = numeric(0), PC8 = numeric(0),
                          PC9 = numeric(0), PC10 = numeric(0),
                          PC11 = numeric(0), PC12 = numeric(0),
                          PC13 = numeric(0), PC14 = numeric(0),
                          PC15 = numeric(0))

  # Get average PCs for each cluster
  for (cluster_id in unique(clusters)) {

    #Getting average PCs
      cluster_cells <- which(clusters == cluster_id)
      cluster_centroid <- colMeans(pca_coords[cluster_cells, , drop = FALSE])
      cluster_centroid_df <- as.data.frame(t(cluster_centroid))
      cluster_centroid_df$cluster <- cluster_id
      centroids <- rbind(centroids, cluster_centroid_df)


  }

  # Now, filter out the clusters that have less than an 80/20 split

  contingency_table <- table(train_folds$leiden, train_folds$label_ident)
  percentage_table <- prop.table(contingency_table, margin = 1) * 100
  #print(percentage_table)
  extreme_splits <- apply(percentage_table, 1, function(x) any(x > 75 | x < 25))

# Filter clusters with extreme splits
  extreme_clusters <- names(extreme_splits)[extreme_splits]
  extreme_clusters<- as.numeric(extreme_clusters)

for (cluster_id in unique(clusters)) {
  #print("percentage table")
 # print(percentage_table)
  #print(cluster_id)
  cluster_id<-as.numeric(cluster_id)
  # Get the percentages for the current cluster
  cluster_percentages <- percentage_table[cluster_id, ]
  #print("cluster_percentages")
  #print(cluster_percentages)
  # Identify the majority label (the one with the highest percentage)
  majority_label <- names(cluster_percentages)[which.max(cluster_percentages)]
  #print("majority label:")
  #print(majority_label)
  centroids[centroids$cluster == cluster_id, "label"] <- majority_label


}

#print(centroids)


# Print the clusters with extreme splits
  #print(extreme_clusters)

  centroids_filtered <- centroids[centroids$cluster %in% extreme_clusters, ]
```

```r
  return(centroids_filtered)
}
```

Now that we have a method to get the majority clusters, I will create a method that finds the closest cluster, classifies the cell type as such, and compares to the true label-a testing method

```r
# Just doing this so we can use this method later.
euclidean_distance <- function(x, y) {
  sqrt(sum((x - y)^2))
}
```

```r
classify_test<-function(train_folds,test_fold){

  centroid_df<-get_rep_centroids(train_folds)
  test_pca_coords <- test_fold[["pca"]]@cell.embeddings[, 1:15]
  #print(test_pca_coords)


  closest_centroid <- numeric(nrow(test_pca_coords))

#confusion dictionary, antimicrobial is considered positive, suboptimal is considered negative
  conf_dict<-list()
  conf_dict[["TP"]]<-0
  conf_dict[["FP"]]<-0
  conf_dict[["TN"]]<-0
  conf_dict[["FN"]]<-0


# Loop over each cell in test_pca_coords
  for (i in 1:nrow(test_pca_coords)) {
    cell_coords <- test_pca_coords[i, ]

    min_dist<-Inf
    pred_label<-""
    for (j in 1:nrow(centroid_df)) {

      row<-(centroid_df[j,])
      label<-row[[17]]
      cent_pcs<-as.matrix(centroid_df[j, 1:15])
      dist<-euclidean_distance(cent_pcs,cell_coords)
      if (dist<min_dist){
        min_dist<-dist
        pred_label<-label
      }}

    true_label<-test_data$label_ident[[i]]
    if (true_label==pred_label){
      if(true_label=="antimicrobial"){ conf_dict[["TP"]]<-conf_dict[["TP"]]+1 }
      else if (true_label=="suboptimal"){conf_dict[["TN"]]<-conf_dict[["TN"]]+1 }}
      else{
        if(true_label=="antimicrobial"){conf_dict[["FN"]]<-conf_dict[["FN"]]+1}
        else if (true_label=="suboptimal"){conf_dict[["FP"]]<-conf_dict[["FP"]]+1}}
}
```

```r
 return(conf_dict)
}
```

The following script just runs this for one fold. Helpful for debugging purposes. Skip if you want to run for all folds.

```r
train_data <- subset(msc, fold !=5)
test_data <- subset(msc, fold ==5)

con_mat<-classify_test(train_data,test_data)
```

```
## Computing nearest neighbor graph

## Computing SNN
```

```r
print(con_mat)
```

```
## $TP
## [1] 961
##
## $FP
## [1] 179
##
## $TN
## [1] 758
##
## $FN
## [1] 90
```

```r
## Accuracy Score
(con_mat[["TP"]]+con_mat[["TN"]])/(con_mat[["TP"]]+con_mat[["TN"]]+con_mat[["FP"]]+con_mat[["FN"]])
```

```
## [1] 0.8646881
```

This method runs this for all folds. Use this to get results.

This method runs this for all folds. Use this to get results.

```r
acc_list<-list()
con_mat_list<-list()
for (test_fold_num in 1:5){
  train_data <- subset(msc, fold !=test_fold_num)
  test_data <- subset(msc, fold ==test_fold_num)
  con_mat<-classify_test(train_data,test_data)
  con_mat_list[[test_fold_num]]<-con_mat
  acc<-(con_mat[["TP"]]+con_mat[["TN"]])/(con_mat[["TP"]]+con_mat[["TN"]]+con_mat[["FP"]]+con_mat[["FN"
  acc_list[[test_fold_num]] <- acc
}
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

Print your confusion matrix

```
print(con_mat_list)
```

```
## [[1]]
## [[1]]$TP
## [1] 933
##
## [[1]]$FP
## [1] 245
##
## [[1]]$TN
## [1] 716
##
## [[1]]$FN
## [1] 92
##
##
## [[2]]
## [[2]]$TP
## [1] 662
##
## [[2]]$FP
## [1] 107
##
## [[2]]$TN
## [1] 880
##
## [[2]]$FN
## [1] 339
##
```

```
##
## [[3]]
## [[3]]$TP
## [1] 875
##
## [[3]]$FP
## [1] 204
##
## [[3]]$TN
## [1] 778
##
## [[3]]$FN
## [1] 131
##
##
## [[4]]
## [[4]]$TP
## [1] 905
##
## [[4]]$FP
## [1] 206
##
## [[4]]$TN
## [1] 780
##
## [[4]]$FN
## [1] 97
##
##
## [[5]]
## [[5]]$TP
## [1] 961
##
## [[5]]$FP
## [1] 179
##
## [[5]]$TN
## [1] 758
##
## [[5]]$FN
## [1] 90
```

Accuracy for each fold

```r
print(acc_list)
```

```
## [[1]]
## [1] 0.8303122
##
## [[2]]
## [1] 0.7756539
##
```

```
## [[3]]
## [1] 0.8314889
##
## [[4]]
## [1] 0.8475855
##
## [[5]]
## [1] 0.8646881
```

Get more metrics

```
# Initialize vectors to store metrics
precision_values <- c()
recall_values <- c()
f1_values <- c()
acc_values<-c()

# Loop through each confusion matrix in the list
for (conf_dict in con_mat_list) {
  TP <- conf_dict[["TP"]]
  FP <- conf_dict[["FP"]]
  TN <- conf_dict[["TN"]]
  FN <- conf_dict[["FN"]]

  # Avoid division by zero issues
  acc<-(TP+TN)/(TP+TN+FP+FN)
  precision <- ifelse((TP + FP) > 0, TP / (TP + FP), NA)
  recall <- ifelse((TP + FN) > 0, TP / (TP + FN), NA)
  f1 <- ifelse(!is.na(precision) & !is.na(recall) & (precision + recall) > 0,
               2 * (precision * recall) / (precision + recall), NA)

  # Store values
  acc_values <-c(acc_values, acc)
  precision_values <- c(precision_values, precision)
  recall_values <- c(recall_values, recall)
  f1_values <- c(f1_values, f1)
}

# Compute mean and standard deviation, handling NA values
acc_mean<-mean(acc_values, na.rm = TRUE)
acc_sd<- sd(acc_values, na.rm = TRUE)
precision_mean <- mean(precision_values, na.rm = TRUE)
precision_sd <- sd(precision_values, na.rm = TRUE)
recall_mean <- mean(recall_values, na.rm = TRUE)
recall_sd <- sd(recall_values, na.rm = TRUE)
f1_mean <- mean(f1_values, na.rm = TRUE)
f1_sd <- sd(f1_values, na.rm = TRUE)

# Print results
cat("Average Accuracy: Mean =", acc_mean, ", SD =", acc_sd, "\n")
```

```
## Average Accuracy: Mean = 0.8299457 , SD = 0.03342272
```

```
cat("Average Precision: Mean =", precision_mean, ", SD =", precision_sd, "\n")
```

```
## Average Precision: Mean = 0.8242757 , SD = 0.02738867
```

```
cat("Average Recall: Mean =", recall_mean, ", SD =", recall_sd, "\n")
```

```
## Average Recall: Mean = 0.851785 , SD = 0.107901
```

```
cat("Average F1-score: Mean =", f1_mean, ", SD =", f1_sd, "\n")
```

```
## Average F1-score: Mean = 0.833641 , SD = 0.04992197
```

```
## Don't need to run this code, used for saving files
## Extracting the clusters for gene ont anlaysis

# expr_matrix <- GetAssayData(msc, assay = "RNA", layer = "data")
# raw_counts <- GetAssayData(msc, assay = "RNA", layer = "counts")
#
# head(raw_counts)
# print(raw_counts)
# str(msc)
# slotNames(msc)
# msc
#
# metadata <- msc@meta.data
# leiden_clusters <- metadata$leiden
# expr_matrix <- GetAssayData(msc, assay = "RNA", slot = "data")
# expr_with_clusters <- cbind(leiden = leiden_clusters, as.data.frame(t(expr_matrix)))
#
# head(expr_with_clusters)
#
# # replace the path with what you want
# write.table(expr_with_clusters, file = "/Users/benjaminmellin/Desktop/Grad_School/fds_project/express
```