

Chemoinformatic project

M1- ISDD

Student name: Van Thinh TO
Student ID: 22518830

1 Preparation of data and computation of molecule similarity

1.1 Analysis of the data set

```
# load the necessary package
library(CemmineR)
```

```
# load data set
training_data <- read.csv("data/training_data.csv", sep=",")
dim(training_data)
```

```
## [1] 224945      5
```

```
colnames(training_data)
```

```
## [1] "SID"      "partition" "activity"  "potency"  "smiles"
```

```
table(training_data$activity)
```

```
##
##   Active Inactive
##   1453   223492
```

How many molecules are included in the data set? How many are Active and how many are Inactive? There are 224945 molecules in the data set, which have 1453 active and 223492 inactive data points.

What information is provided in each column? What is the data type of each variable? There are five columns in the data set:

- **SID**: unique identifier for each molecule (*integer*)
- **partition**: indicates Training or Test set (*character*)
- **activity**: biological class (Active / Inactive) (*character*)
- **potency**: numerical activity value, e.g., $-\log(\text{IC}_{50})$ (*numeric*)
- **smiles**: SMILES string representing each molecule (*character*)

```
str(training_data)
```

```
## 'data.frame': 224945 obs. of 5 variables:
## $ SID : int 405266972 405266969 405258654 405266973 405266971 439657910 440228774 381749740 4
## $ partition: chr "Training" "Training" "Training" "Training" ...
## $ activity : chr "Active" "Active" "Active" "Active" ...
## $ potency : num 9.3 9.3 9.3 9.3 9.22 ...
## $ smiles : chr "C1=CC(=C(C=C1N2C=C(N=N2)CN3C=CN=C3N)C(F)F)F" "C1=CC(=C(C=C1N2C=C(N=N2)CN3C=CN=C3N)C(F)F)F"
```

Check whether the data set contains any missing values (NA). If this is the case, remove all molecules with missing entries. How many molecules are removed?

```
# get nan data points
# which(is.na(training_data), arr.ind = TRUE)
sum(is.na(training_data))
```

```
## [1] 223157
```

```
# get nan values for each column
sapply(training_data, function(x) sum(is.na(x)))
```

```
##      SID partition activity potency smiles
##      0          0         0    223157      0
```

```
# drop rows with nan value
training_data <- na.omit(training_data)
dim(training_data)
```

```
## [1] 1788      5
```

1.2 Exploring the Chemical Space and Computing Molecular Similarity

```
# create data set for the sdf conversion
herg <- training_data[c("smiles", "SID")]
write.table(herg, "data/herg.smi", sep = "\t", quote = FALSE, col.names = FALSE,
  ↪ row.names = FALSE)
```

```
# read sdf file after creation
sdf_df <- read.SDFset("data/herg.sdf")
```

```
# compute the AP fingerprints
ap <- sdf2ap(sdf_df)
```

```
# # compute the similarity matrix
# simi_matrix = matrix(0, nrow=length(ap), ncol=length(ap))
# for (i in 1:length(ap)){
#   for (j in i:length(ap)){
#     simi_ij <- cmp.similarity(ap[i], ap[j])
#     simi_matrix[i, j] <- simi_ij
```

```
#      simi_matrix[j, i] <- simi_ij
#    }
#  }
#
# # save similarity matrix for the later usage
# save(simi_matrix, file="data/simi_matrix.RData")
```

```
# load the similarity matrix
load("data/simi_matrix.RData")
```

```
# compute the dissimilarity matrix
dissimi_matrix <- 1-simi_matrix
```

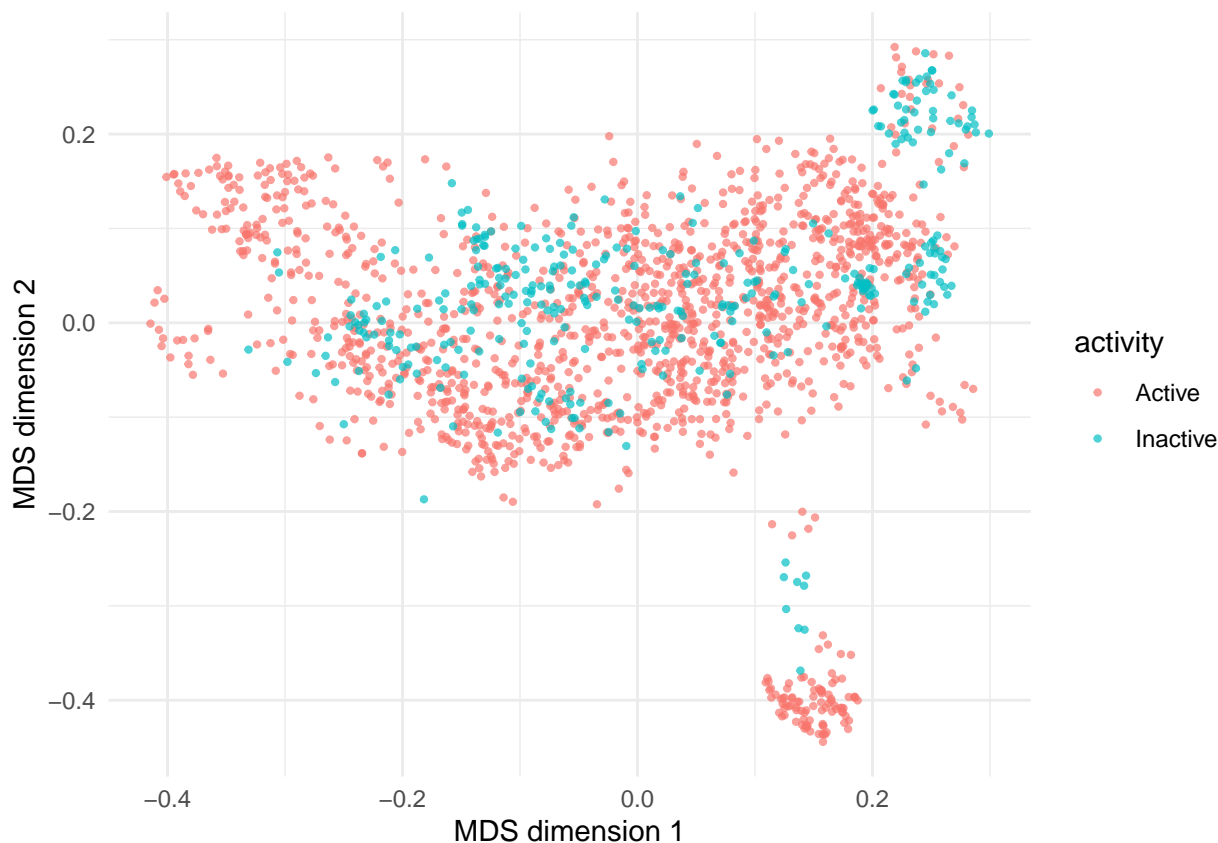
```
# perform MDS
mds <- cmdscale(dissimi_matrix)
```

```
# Visualization MDS
# load ggplot
library(ggplot2)

vis_mds <- function(mds, df, group_col){
  # create data frame for visualization
  df_mds <- data.frame(
    x = mds[,1],
    y = mds[,2],
    color = df[,group_col]
  )

  # plot
  ggplot(df_mds, aes(x = x, y = y, color = color, )) +
    # scatter plot
    geom_point(size = 0.8, alpha = 0.7) +
    labs(
      x = "MDS dimension 1",
      y = "MDS dimension 2",
      color = group_col
    ) +
    # set theme
    theme_minimal()
}

vis_mds(mds=mds, df=training_data, group_col="activity")
```



We can observe that the molecule are widely dispersed. However, there is a huge overlap between active and inactive data points.

```
table(training_data$activity)
```

```
##
##   Active Inactive
##   1453     335
```

2 Balancing the dataset and preparing training and test sets

2.1 Cluster-based undersampling method

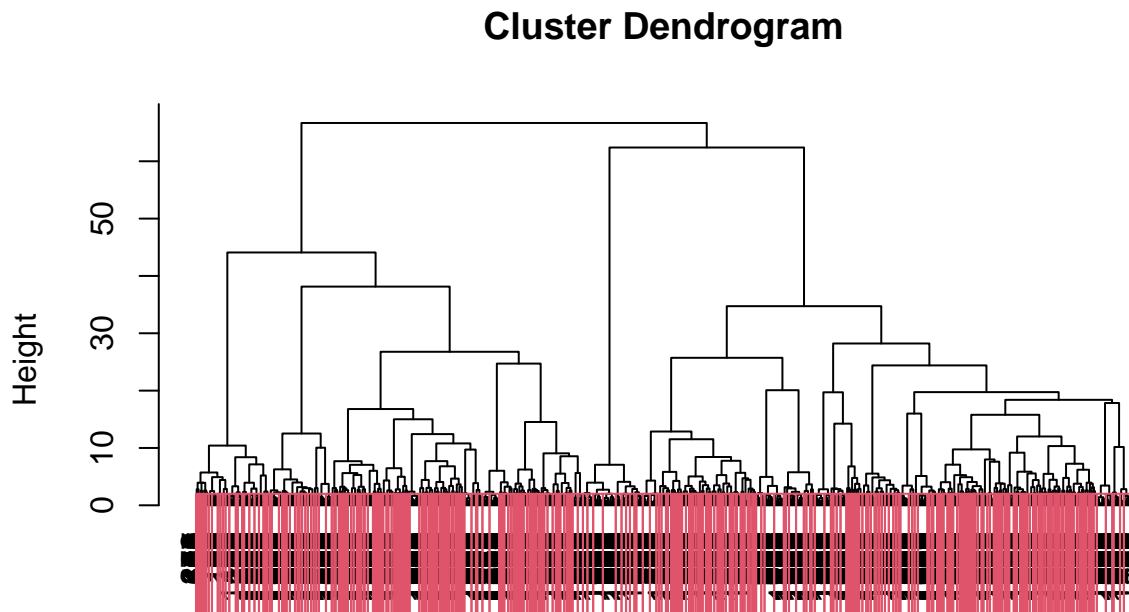
```
# Extract active molecules
active_idx <- which(training_data$activity== "Active")
```

```
# Calculate the dissimilarity matrix for actives only
dissimi_act_rep <- 1- simi_matrix[active_idx, active_idx]
```

```
# Perform hierarchical clustering
hc <- hclust(dist(dissimi_act_rep), method = "ward.D2")
```

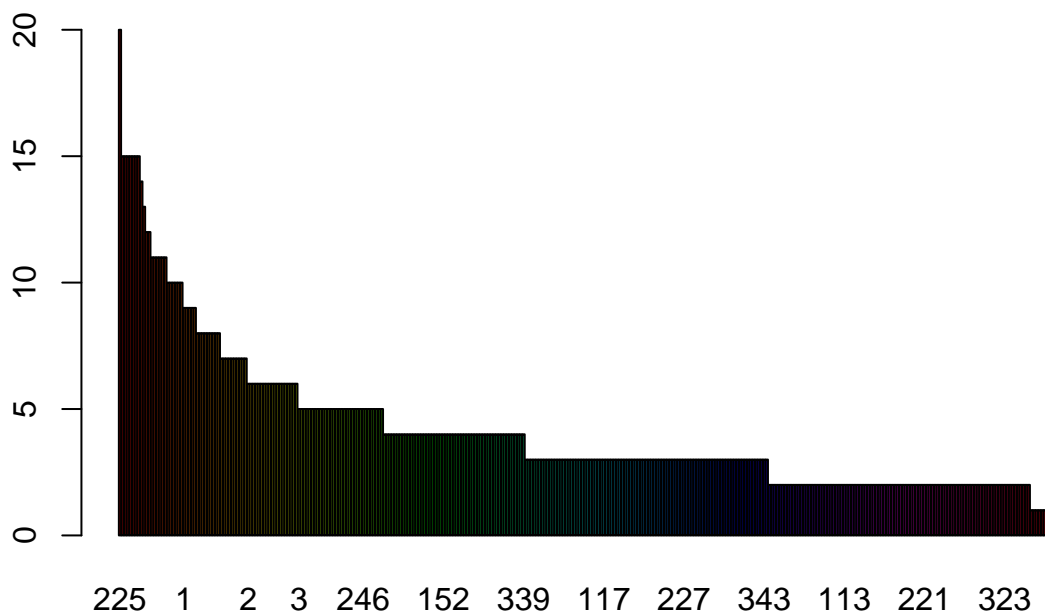
```
# Cut the dendrogram into 350 clusters
active_groups <- cutree(hc, k=350)
```

```
# plot the clusters
plot(hc, hang = -1)
rect.hclust(hc, k = 350)
```



```
dist(dissimi_act_rep)
hclust (*, "ward.D2")
```

```
# pdf("figs/comp_cluster.pdf", width = 10, height = 6)
# to see how many compounds each group has
table_group <- table(active_groups)
barplot(sort(table_group, decreasing = TRUE), col = rainbow(length(table_group)),)
```



```
# dev.off()
```

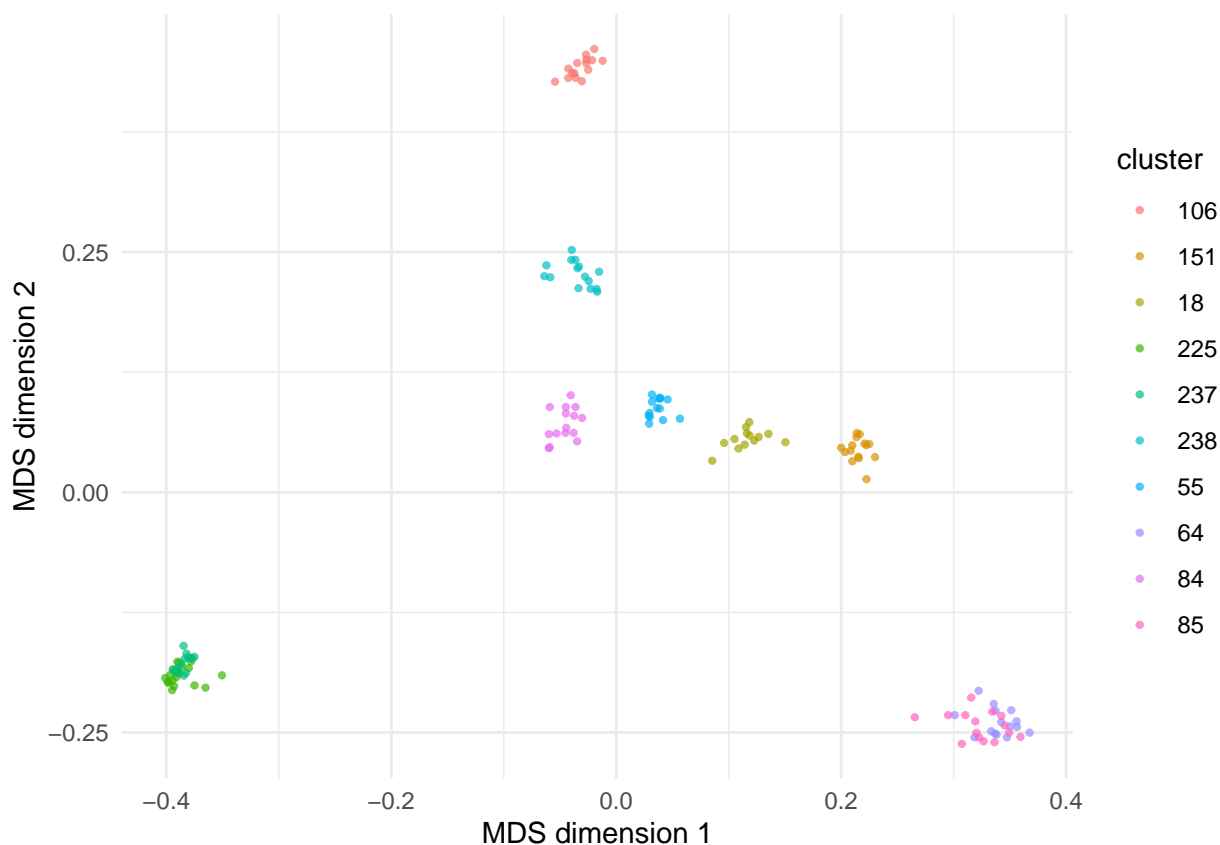
```
# visualize top clusters with the highest number of compounds
top_cls <- 10
top_cluster <- strtoi(names(sort(table_group, decreasing = TRUE)[1:top_cls]))

# create a copy of training data set
training_data_cls <- training_data

# get the indices of compounds belong to top clusters and create cluster column for
↳ separation
top_cluster_act_idx <- c()
for (clu in top_cluster){
  top_cluster_act_idx <- c(top_cluster_act_idx, which(active_groups==clu))
  training_data_cls[which(active_groups==clu), "cluster"] <- toString(clu)
}

# get the similarity matrix and compute mds
simi_matrix_top10 <- simi_matrix[top_cluster_act_idx, top_cluster_act_idx]
mds_top10 <- cmdscale(1-simi_matrix_top10)

# visualize
vis_mds(mds=mds_top10, df=training_data_cls[top_cluster_act_idx,], group_col="cluster")
```



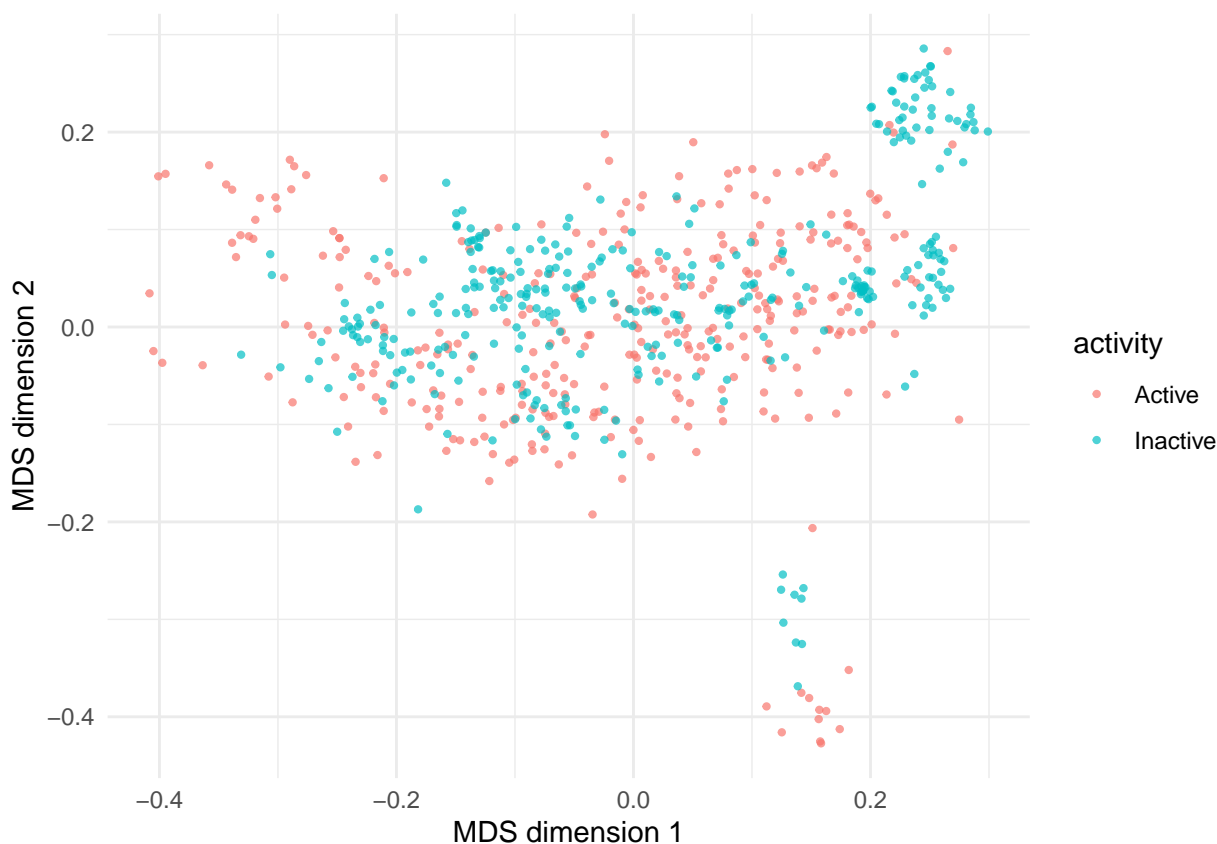
```
# Select one molecule per cluster
set.seed(20022001) # seed for reproduction
selected_active_idx <- c()
for (group in 1:350){
  # for each group, randomly choose one active molecule, then append to a vector
  selected_active_idx <- c(selected_active_idx, sample(which(active_groups==group),
↪ size=1))
}
```

```
# create a balanced data set
# select inactive
inactive_idx <- which(training_data$activity== "Inactive")
# concate inactive and selected active
balanced_df <- training_data[c(selected_active_idx, inactive_idx),]
dim(balanced_df)
```

```
## [1] 685 5
```

```
# create data frame for visualization
mds_selected <- data.frame(
  # select mds coordinations for the balanced data set
  x = mds[c(selected_active_idx, inactive_idx),1],
  y = mds[c(selected_active_idx, inactive_idx),2],
  activity = balanced_df$activity
)
```

```
vis_mds(mds=mds_selected, df=balanced_df, group_col="activity")
```



According to the Figure above, the cluster-based undersampling method avoids redundancy, preserves chemical diversity, and produces a balanced dataset suitable for building robust predictive models.

```
# save the balanced data set
save(balanced_df, file = "data/balanced_df.RData")
```

2.2 Preparing training and test set

```
set.seed(20022001)
# randomly select 80% of molecules from the balanced data set
training_idx <- sample(1:nrow(balanced_df), size = 8/10 * nrow(balanced_df))
# get the training and test sets
training_df <- balanced_df[training_idx,]
test_df <- balanced_df[-training_idx,]
```

```
# check the data points for each partition
print(dim(balanced_df))
```

```
## [1] 685 5
```



```
print(dim(training_df))
```

```
## [1] 548 5
```

```
print(dim(test_df))
```

```
## [1] 137 5
```

```
# Create a new variable to indicate whether each molecule belongs to the training or the  
↪ test set  
balanced_df[-training_idx,"partition"] <- "Test"  
table(balanced_df$partition)
```

```
##  
##      Test Training  
##      137      548
```

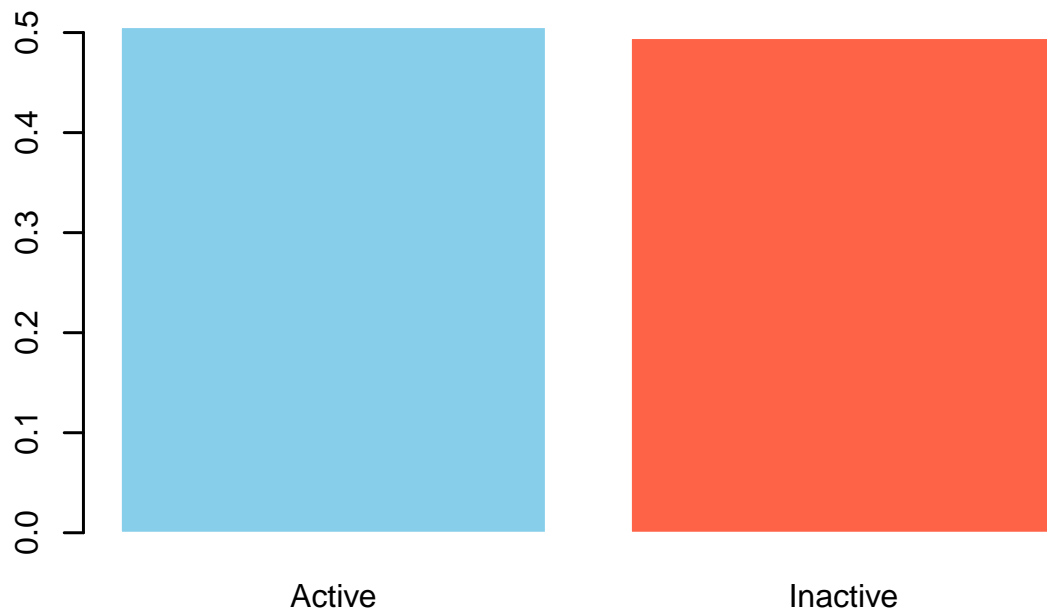
```
# check the distribution of Active and Inactive molecules in the training set  
train_ratio <- table(training_df$activity)/nrow(training_df)  
train_ratio
```

```
##  
##      Active Inactive  
## 0.5054745 0.4945255
```

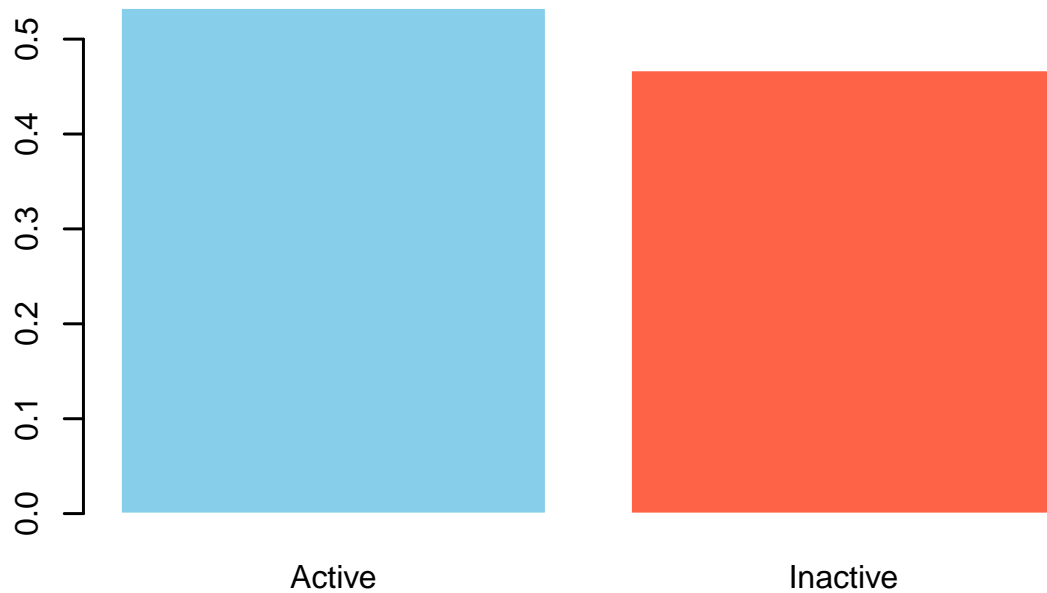
```
# check the distribution of Active and Inactive molecules in the test set  
test_ratio <- table(test_df$activity)/nrow(test_df)  
test_ratio
```

```
##  
##      Active Inactive  
## 0.5328467 0.4671533
```

```
barplot(train_ratio,  
        col = c("skyblue", "tomato"),  
        border = "white",  
        lwd = 1.5)
```

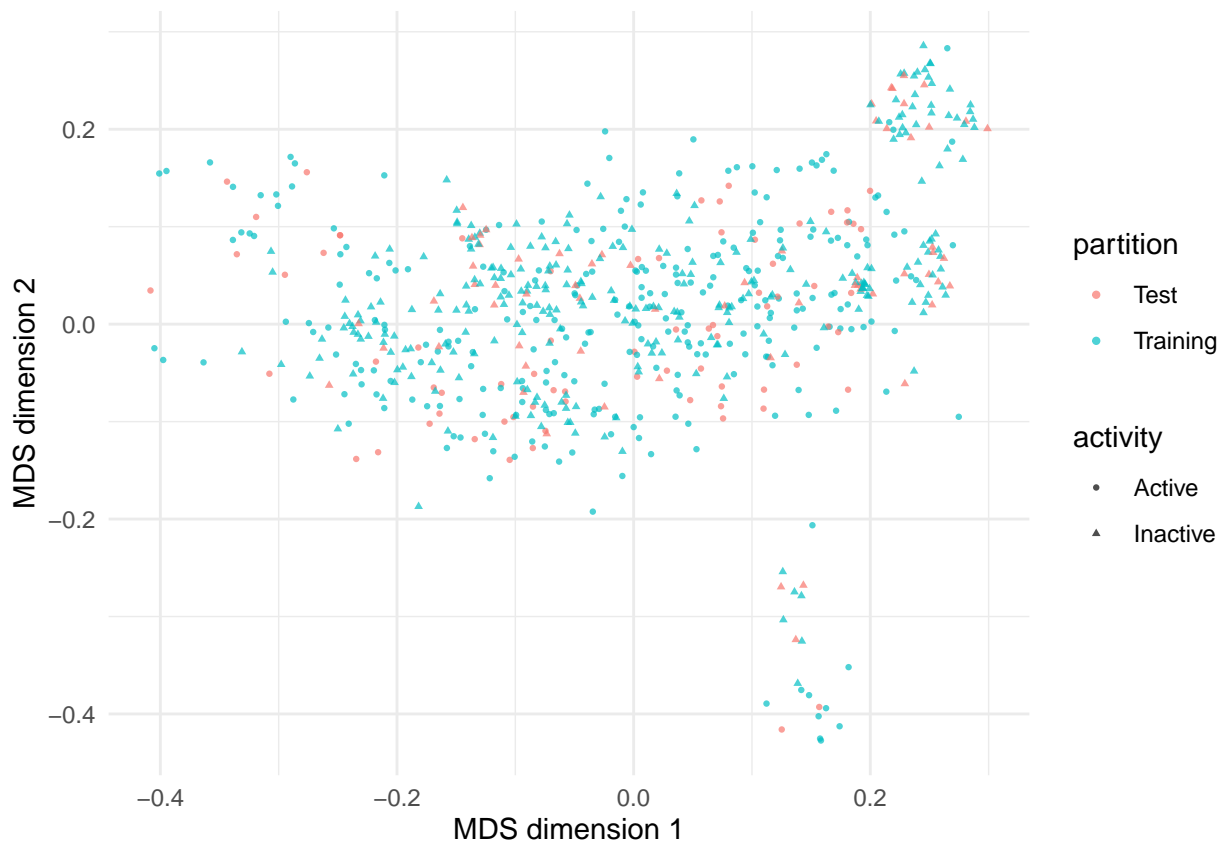


```
barplot(test_ratio,
  col = c("skyblue", "tomato"),
  border = "white",
  lwd = 1.5)
```



```
# create data frame for visualization
mds_selected <- data.frame(
  # select mds coordinations for the balanced data set
  x = mds[c(selected_active_idx, inactive_idx),1],
  y = mds[c(selected_active_idx, inactive_idx),2],
  color = balanced_df$partition,
  shape = balanced_df$activity
)
```

```
# plot
ggplot(mds_selected, aes(x = x, y = y, color = color, group = shape)) +
  # scatter plot
  geom_point(size = 0.8, alpha = 0.7, aes(shape = shape)) +
  labs(
    x = "MDS dimension 1",
    y = "MDS dimension 2",
    color = "partition",
    shape = "activity"
  ) +
  # set theme
  theme_minimal()
```



```
# save the training and test sets
save(training_df, file = "data/training.RData")
save(test_df, file = "data/test.RData")
```

3 Prediction of molecule class (Active/Inactive) Using a k-NN Approach based on chemical similarity

3.1 First steps with the k-NN algorithm

3.1.1 Manual computation

```
# Extract the training-training submatrix
# get training-training indices
training_idx <- strtoi(row.names(training_df))
# get submatrix
training_simi_matrix <- simi_matrix[training_idx, training_idx]
```

```
# Assign NA to the diagonal of this submatrix
diag(training_simi_matrix) = NA
```

```
# get the index of three molecules, which have the highest similarity compared to
↪ molecule 5
man_idx <- order(training_simi_matrix[5,], decreasing = TRUE)[1:3]
```

```
# Identify the activity classes of the 3 neighbors of molecule 5
votes <- training_df$activity[man_idx]
votes
```

```
## [1] "Inactive" "Inactive" "Inactive"
```

```
# Determine the majority class
names(which.max(table(votes)))
```

```
## [1] "Inactive"
```

3.1.2 Building a k-NN model on the training set

```
# function for training the knn model
knn_train <- function(train, simi_matrix, k){
  pred <- c()
  for (i in 1:nrow(train)){
    # get the indices of k neighbors
    neighbor_idx <- order(simi_matrix[i,], decreasing = TRUE)[1:k]
    # majority voting
    votes <- train$activity[neighbor_idx]
    pred_label <- names(which.max(table(votes)))
    # update the label of the prediction vector
    pred <- c(pred, pred_label)
  }
  # compute the confusion matrix
  confusion_matrix <- table(True = train$activity, Predicted = pred)
```

```

    return (confusion_matrix)
}

# function for evaluating the knn model
knn_eval <- function(train, test, simi_matrix, k){
  pred <- c()
  for (i in 1:nrow(test)){
    # get the indices of k neighbors
    neighbor_idx <- order(simi_matrix[i,], decreasing = TRUE)[1:k]
    # majority voting
    votes <- train$activity[neighbor_idx]
    pred_label <- names(which.max(table(votes)))
    # update the label of the prediction vector
    pred <- c(pred, pred_label)
  }
  # compute the confusion matrix
  confusion_matrix <- table(True = test$activity, Predicted = pred)

  return (confusion_matrix)
}

# function to extract classification metrics
class_metrics <- function(confusion_matrix){
  # extract TP, FN, FP, TN values
  TP <- confusion_matrix["Active", "Active"]
  FN <- confusion_matrix["Active", "Inactive"]
  FP <- confusion_matrix["Inactive", "Active"]
  TN <- confusion_matrix["Inactive", "Inactive"]

  # calculate metrics
  accuracy <- (TP + TN) / sum(confusion_matrix)
  recall <- TP / (TP + FN)
  precision <- TP / (TP + FP)
  specificity <- TN / (TN + FP)
  balance_accuracy <- mean(recall, specificity)
  f1_score <- (2 * recall * precision)/(recall + precision)

  return (
    list(
      accuracy=accuracy,
      recall=recall,
      precision=precision,
      specificity=specificity,
      balance_accuracy = balance_accuracy,
      f1_score = f1_score
    )
  )
}

```

3.2 Implementing the k-NN model: selection of the optimal number of neighbors with cross-validation

```
set.seed(20022001)

cross_validation <- function(df, simi_matrix, K_folds){
  # create 1-10 vectors and then randomly change their positions
  fold_id <- sample(rep(1:K_folds, length.out = nrow(df)))

  k_values <- 2:20

  # to store per-fold metrics
  cv_raw <- data.frame()

  for (k in k_values) {
    for (fold in 1:K_folds) {
      # get train and valid indices for this fold
      valid_idx_local <- which(fold_id == fold)
      train_idx_local <- which(fold_id != fold)

      # split data
      train_df_cv <- df[train_idx_local, ]
      valid_df_cv <- df[valid_idx_local, ]

      # map to global indices to get subsimilarity matrix
      train_row_idx <- strtoi(rownames(train_df_cv))
      valid_row_idx <- strtoi(rownames(valid_df_cv))

      # similarity matrices for this fold
      train_simi_matrix <- simi_matrix[train_row_idx, train_row_idx]
      diag(train_simi_matrix) <- NA # can not be neighbored by themselves

      # always get similarity between valid and train data points
      valid_simi_matrix <- simi_matrix[valid_row_idx, train_row_idx]

      # evaluate on validation fold
      conf_mat <- knn_eval(
        train = train_df_cv,
        test = valid_df_cv,
        simi_matrix = valid_simi_matrix,
        k = k
      )

      metrics <- class_metrics(conf_mat)

      # save this fold result
      cv_raw <- rbind(
        cv_raw,
        data.frame(
          k = k,
          fold = fold,
          accuracy = metrics$accuracy,
          recall = metrics$recall,
          precision = metrics$precision,

```

```

        specificity = metrics$specificity,
        balance_accuracy = metrics$balance_accuracy,
        f1_score = metrics$f1_score
    )
}
}

return (cv_raw)
}

cv_raw <- cross_validation(df=training_df, simi_matrix=simi_matrix, K_folds=10)

```

```

# # select the matrix for evaluating the knn model
# training_idx <- strtoi(row.names(training_df))
# train_simi_matrix <- simi_matrix[training_idx, training_idx]
# diag(train_simi_matrix) <- NA
#
# confusion_matrix <- knn_train(
#   train=training_df,
#   simi_matrix=train_simi_matrix,
#   k=4
# )
# metrics = class_metrics(confusion_matrix)
#
# metrics

```

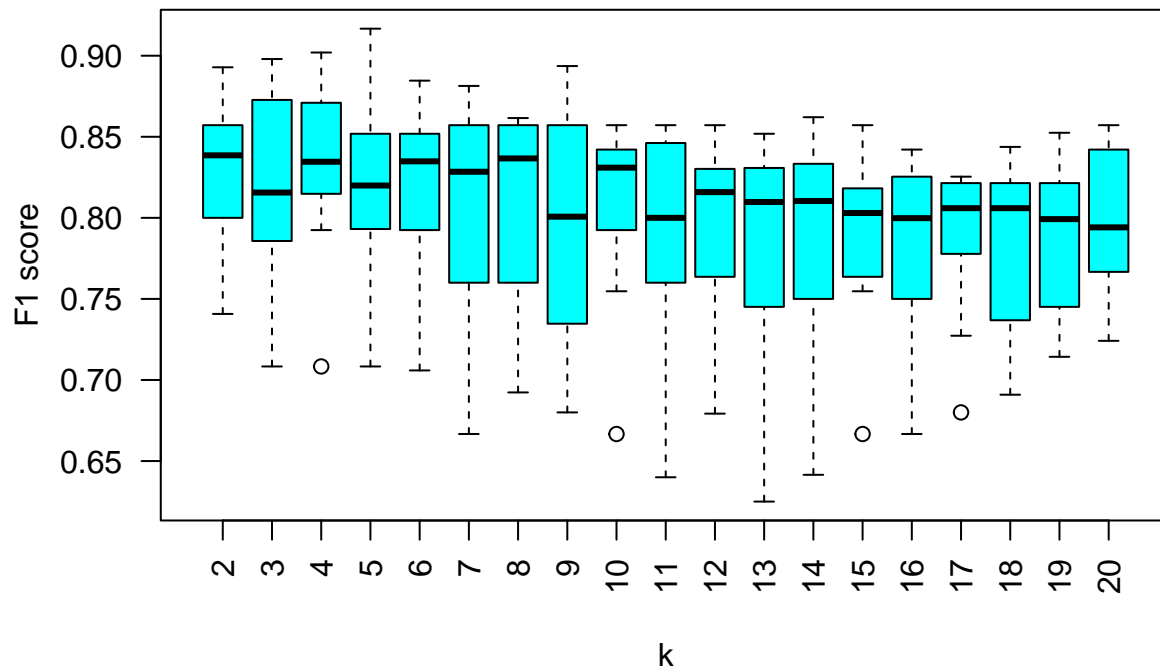
```

library(tidyr)

# long format: one row per (k, fold, metric)
cv_long <- cv_raw |>
  pivot_longer(
    cols = c(accuracy, recall, precision, specificity, balance_accuracy, f1_score),
    names_to = "metric",
    values_to = "value"
  )

# pdf("figs/cross_f1.pdf", width = 10, height = 6)
# draw boxplot
boxplot(
  value ~ k,
  data = cv_long[cv_long$metric == "f1_score", ],
  ylab = "F1 score",
  col = "cyan",
  las = 2
)

```



```
# dev.off()
```

3.3 Evaluating the performance of the model on the test set

```
# select the matrix for evaluating the knn model
subtest_idx <- strtoi(row.names(test_df))
training_idx <- strtoi(row.names(training_df))

test_simi_matrix <- simi_matrix[subtest_idx, training_idx]

confusion_matrix <- knn_eval(
  train=training_df,
  test = test_df,
  simi_matrix=test_simi_matrix,
  k=4
)
metrics = class_metrics(confusion_matrix)

metrics
```

```
## $accuracy
## [1] 0.9051095
##
## $recall
## [1] 0.9178082
##
## $precision
## [1] 0.9054054
##
```



```
## $specificity
## [1] 0.890625
##
## $balance_accuracy
## [1] 0.9178082
##
## $f1_score
## [1] 0.9115646
```