classification_experiement_v3

2022-09-30

```
# library(knitr)
# purl("classification_experiment_v5.Rmd", output = 'clf_v5.R')
```

```
library(missMDA)
library(scales)
library(softImpute)
library(mice)
library(missForest)
library(caret)
library(caTools) # to create train/test split
library(e1071)
library(future.apply)
plan(multisession)
```

IF HASNOT DOWNLOAD THE FILE YET THEN

```
library(dslabs)

#getting the path to save

curr_dir = getwd()
path = '../data'
mnist_path = file.path(curr_dir, path)

if (!file.exists(file.path(mnist_path, "train-images-idx3-ubyte")) |
  !file.exists(file.path(mnist_path, "t10k-images-idx3-ubyte")) |
  !file.exists(file.path(mnist_path, "train-labels-idx1-ubyte")) |
  !file.exists(file.path(mnist_path, "t10k-labels-idx1-ubyte"))
  ){

  # getting the data
  mnist <- read_mnist(
    path = NULL,
    destdir = mnist_path,
    download = TRUE,
    url = "https://www2.harvardx.harvard.edu/courses/IDS_08_v2_03/",
    keep.files = TRUE
  )

  # clear folder data (avoid wrong zipping)
  list_files = list.files(path=mnist_path)
  for (x in 1:length(list_files)){
    file_path = file.path(mnist_path, list_files[x])
    if (substring(file_path, nchar(file_path)-2, nchar(file_path)) == '.gz'){
      R.utils::gunzip(file_path, overwrite=TRUE, remove=FALSE)
    }
```

```
    }
}
```

IF FILE IS ALREADY DOWNLOADED AND UNZIP THEN JUST READ

```
# load image files
load_image_file = function(filename) {
  ret = list()
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n    = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  x = readBin(f, 'integer', n = n * nrow * ncol, size = 1, signed = FALSE)
  close(f)
  data.frame(matrix(x, ncol = nrow * ncol, byrow = TRUE))
}

# load label files
load_label_file = function(filename) {
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  y = readBin(f, 'integer', n = n, size = 1, signed = FALSE)
  close(f)
  y
}
```

```
# load images
processing_mnist_data <- function (){
  train = load_image_file(file.path(mnist_path, "train-images-idx3-ubyte"))
  test  = load_image_file(file.path(mnist_path,"t10k-images-idx3-ubyte"))

  train$label =  as.factor(load_label_file(file.path(mnist_path,"train-labels-idx1-ubyte")))

  test$label = as.factor(load_label_file(file.path(mnist_path,"t10k-labels-idx1-ubyte")))
  result = list('train'=train, 'test'=test)
  return(result)
}
```

```
processed_data = processing_mnist_data()
train = processed_data$train
test = processed_data$test
X.train = train[, -785]
X.test = test[, -785]
y.train = train[, 785, drop=F]
y.test = test[, 785, drop=F]
```

```
find_cov_ij <- function(Xij, Sii, Sjj){
  # Xij: the i, j column of the original matrix
  # sii, sjj = \hat{Sigma}_{ii}, \hat{Sigma}_{jj}
  # s11 = sum(Xij[,1]**2, na.rm = TRUE)
  # s12 = sum(Xij[,1]*Xij[,2], na.rm = TRUE)
  # s22 = sum(Xij[,2]**2, na.rm = TRUE)
  #start edited--------------------------
```

```
  comlt_Xij = Xij[complete.cases(Xij), ]
  s11 = sum(comlt_Xij[,1]**2)
  s12 = sum(comlt_Xij[,1]*comlt_Xij[,2])
  s22 = sum(comlt_Xij[,2]**2)
  #end edited--------------------------
  m = sum(complete.cases(Xij))
  coef = c(s12*Sii*Sjj,
                m*Sii*Sjj-s22*Sii-s11*Sjj,
                s12, -m)
  sol = polyroot(z = coef)
  sol = Re(sol)
  scond = Sjj - sol^2/Sii

  #Sii >0
  #start edited--------------------------
  #etas = suppressWarnings(-m*log(sol) - (Sjj-2*sol/Sii*s12+sol^2/Sii^2*s11)/scond)
  etas = suppressWarnings(-m*log(scond) - (Sjj-2*sol/Sii*s12+sol^2/Sii^2*s11)/scond)
  #end edited--------------------------
  return(sol[which.max(etas)])
}

dpers <- function(Xscaled){
  # Xscaled: scaled input with missing data
  # THE INPUT MUST BE NORMALIZED ALREADY
  shape = dim(Xscaled) # dimension
  S = matrix(0, shape[2],shape[2])
  diag(S) = apply(Xscaled, 2, function(x) var(x, na.rm=TRUE))
  # Get the index of the upper triangular matrix (row, column)
  Index<-which(upper.tri(S,diag=FALSE),arr.ind=TRUE)
  # compute the covariance and assign to S based on Index
  #start edited--------------------------
  total = nrow(Index)
  pb <- txtProgressBar(min = 0, max = total, style = 3)
  find_cov_upper_triag = function(i) {
    setTxtProgressBar(pb, i)
    if (S[Index[i,1], Index[i,1]] == 0 | S[Index[i,2], Index[i,2]] == 0){
      return(NA)
    }
    else{
      return (
        find_cov_ij(
            Xscaled[,c(Index[i,1],Index[i,2])],
            S[Index[i,1], Index[i,1]],
            S[Index[i,2], Index[i,2]]
            )
      )
    }
  }


  S_diag_calc = unlist(future_lapply(1:total, find_cov_upper_triag))

  print(length(S_diag_calc))
```

```
  print(length(S[Index]))

  stopifnot(length(S_diag_calc) == length(S[Index]))
  #end edited--------------------------
  S[Index]  = S_diag_calc
  S = S + t(S)
  diag(S) = diag(S)/2
  return(S)
}
```

```
impDi <- function(S, Xtest){
  Xpred = Xtest
  #not to apply the algorithm on the ZERO VARIANCE columns, just fill the with 0 / or mean value (which
  diag(S) = apply(Xtest, 2, function(x) var(x, na.rm=TRUE))
  zero_var_col_indices = (which(diag(S)!=0))

  pool = 1: nrow(Xtest)
  pool = setdiff(pool, zero_var_col_indices)
  # first ignore the sample with no missing entries
  # because we don't need to do anything on them
  pool = setdiff(pool, which(rowSums(is.na(Xtest)) == 0))
  while (length(pool)>0){
  #------ edited
    #x = Xtest[pool[1], ]
    x = as.numeric(Xtest[pool[1], , drop=F])
  #------
    oId = which(!is.nan(x))
    mId = which(is.nan(x))
    # find the samples with features in oId available
    id = which(rowSums(is.nan(Xpred[,oId, drop=F]))==0)

 #------ edited
    #id = which(rowSums(is.nan(Xpred[id, mId, drop=F])) == length(mId))
    filter_rownames = (rowSums(is.nan(Xpred[id, mId, drop=F])) == length(mId))
    d = data.frame(filter_rownames)
    id = which(rownames(as.data.frame(Xpred)) %in% rownames(d[which(d==TRUE), ,drop=F]))
 #------

    pool = setdiff(pool, id)
    So = S[oId, oId]
    Smo = S[oId, mId]
    beta = solve(So) %*% Smo
    Xpred[id, mId] = t(beta) %*% t(Xtest[id, oId, drop=F])
  }
  Xpred[which(is.na(Xpred))] = 0
  return(Xpred)
}
```

IMPUTE WITH IMPDI

```
impDi_run <- function(X.train, y.train, X.test, y.test){
  #a) on training set
  X.train[is.na(X.train)] <- NaN
```

```
  sigmaDper = dpers(X.train)
  print("S is done")

  X_imp.train = impDi(sigmaDper, X.train)[,, drop=F]
  #b) on testing set
  print("imp train is done")
  X.test[is.na(X.test)] <- NaN
  X_imp.test = impDi(sigmaDper, X.test)[,, drop=F]

  result = list("train" = X_imp.train, "test" = X_imp.test)
  return(result)
}
```

SOFTIMPUTE

```
softImpute_run <- function(X.train, y.train, X.test, y.test){
  #a) on training set
  fit_train = softImpute(as.matrix(X.train) , type = 'als')
  X_imp.train = softImpute::complete(
                              as.matrix(X.train),
                              fit_train)[,, drop=F]
  #b) on testing set
  fit_test = softImpute(as.matrix(X.test) , type = 'als')
  X_imp.test = softImpute::complete(
                             as.matrix(X.test),
                             fit_test)[,, drop=F]
  result = list("train" = X_imp.train, "test" = X_imp.test )
  return(result)
}
# impted = softImpute_run(Xnorm.train, y.train, Xnorm.test, y.test)
```

DATA PREPARATION

READING DATA :

CREATE NON RANDOM MISSING VALUE

```
get_image_position_spatial_to_flatten<- function(delImgPosWidth, delImgPosHeight){
  # delImgPosHeight: row
  # delImgPosWeight : col
  tmp = c(1:784)
  im <- matrix(unlist(tmp),nrow = 28,byrow = T)
  idxs = im[delImgPosHeight, delImgPosWidth]
  return(matrix(idxs,nrow = 1,byrow = T)[, ])
}
```

```
image_edge_deleting <- function(
    data,
    delete_type, #by_percent, by_pixel_number
    percents_of_data,
    image_width,
    image_height,
```

```
    width_del_percent=0,
    height_del_percent=0,
    from_pixel_width=None,
    from_pixel_height=None
    ){
  if (delete_type =='by_percent'){
    n = dim(data)[2]
    from_pixel_width = ceiling((1-width_del_percent)*image_width)
    from_pixel_height = ceiling((1-height_del_percent)*image_height)
  }
  if (delete_type=='by_pixel_number'){
    from_pixel_width = from_pixel_width
    from_pixel_height = from_pixel_height
  }

flatten_columns_removed = get_image_position_spatial_to_flatten(
  from_pixel_width:image_width,
  from_pixel_height: image_height
)

  flatten_rows_removed = sample.int(nrow(data), as.integer(nrow(data)*percents_of_data))
  missing_data = data
  missing_data[flatten_rows_removed, flatten_columns_removed] <- NA
  result = list(
    'missing_data'=missing_data,
    'flatten_columns_removed'=flatten_columns_removed,
    'flatten_rows_removed'=flatten_rows_removed
  )
  return(result)
}
```

```
# visualize the deleted images
visualize_digit <- function(missing_X, y, train_removed_rows, per_col, per_row){
  par(mfcol=c(per_col, per_row))
  par(mar=c(0, 0, 3, 0), xaxs='i', yaxs='i')
  for (idx in 1:(per_col*per_row)) {
      im <- matrix(unlist(missing_X[train_removed_rows, ][idx, ]),nrow = 28,byrow = T)
      im <- t(apply(im, 2, rev))
      image(1:28, 1:28, im, col=gray((0:255)/255), xaxt='n', main=paste(y[train_removed_rows,,drop=F][i
  }
}
```

```
#data normalization
sampling_data <- function(data, y_col_name, sample_perc){
  data$label= data[, y_col_name]
  sample_indices <- sample.split(Y=data[, 'label'], SplitRatio = sample_perc)
  sample_data <- as.data.frame(subset(data, sample_indices == TRUE))
  result = list(
    'sample'=sample_data,
    'X'=as.matrix(sample_data[, 1:(28*28)]),
    'y'=sample_data[,'label', drop=F]
  )
  return(result)
}
```

Normalizing train and test using train 's parameters

```
normalizing <- function(x=None, Xtrain=None){
  na_mask = is.na(x)
  mean = apply(Xtrain, 2, mean, na.rm=TRUE)
  sd = apply(Xtrain, 2, sd, na.rm=TRUE)
  sd_equal_zero_mask = which(sd==0)
  subtract_mean = sweep(x, 2, mean, '-')
  X_normed = sweep(subtract_mean, 2, sd, "/")
  X_normed[is.na(X_normed)] = 0
  X_normed[is.infinite(X_normed)] = 0
  X_normed[na_mask] = NA
  result = list('X_normed'=X_normed, 'mean'=mean, 'sd'=sd, 'sd_equal_zero_mask'=sd_equal_zero_mask)
  return (result)
}

reconstructingNormedMatrix <- function(X_norm, mean, std){
  mult = sweep(X_norm, 2, std, '*')
  reconstrc = sweep(mult, 2, mean, '+')
  return (reconstrc)
}
```

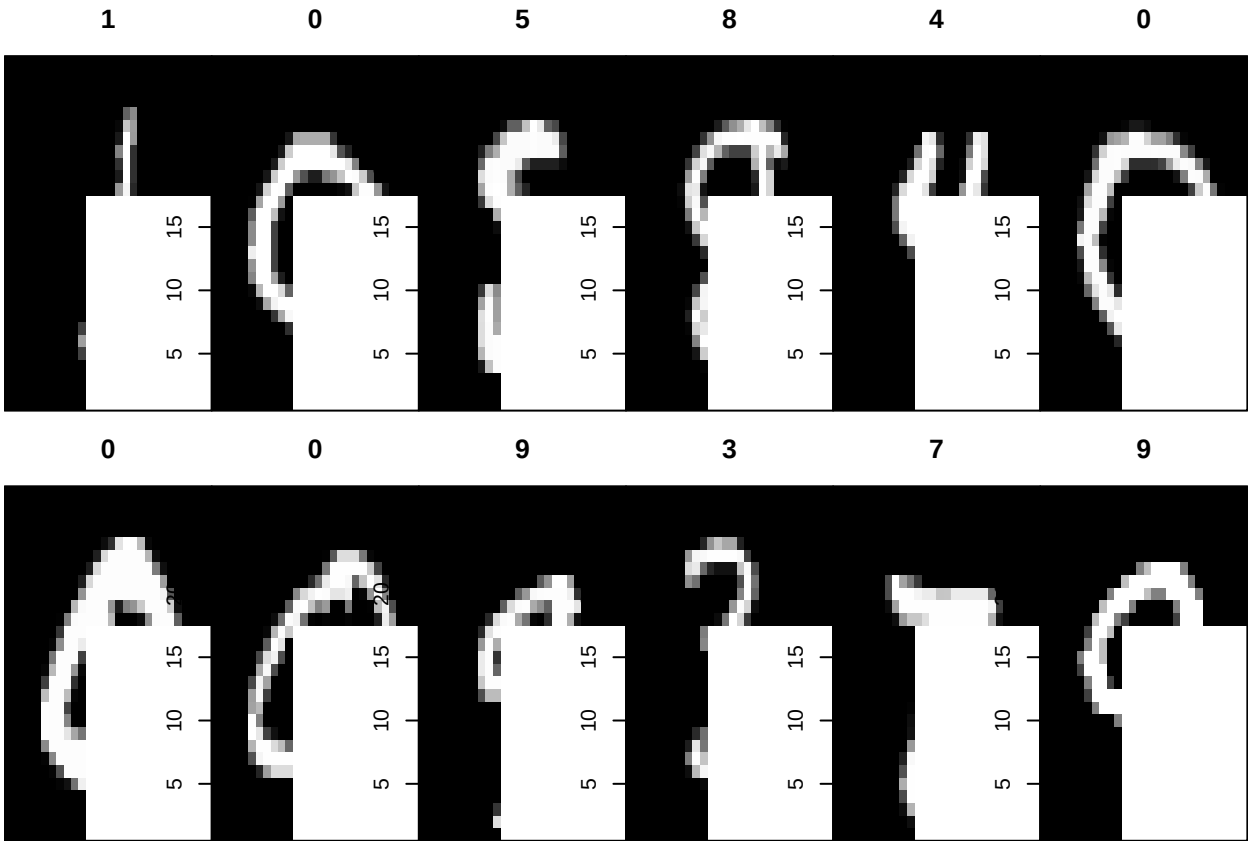
#prediction using svm (this option take more than 1hours to run)

```
#run model
predicting_svm <- function(imputed.X.train, y.train, imputed.X.test, y.test){
  start = Sys.time()
  y_train = as.factor(y.train[, ])
  y_train = sapply(y.train, as.character)
  y_test = as.factor(y.test[, ])

  model <- svm(y = y_train,
               x = imputed.X.train,
               type = 'C-classification',
               kernel = "linear",
               scale = FALSE )

  pred <- predict(model, imputed.X.test)
  pred <- as.factor(pred)

  acc = mean(pred == y_test)
  duration = Sys.time() - start

  #softImpute_acc = confusionMatrix(pred, as.factor(y.test))
  result = list('accuracy' = acc, 'model_run_time'= duration)

  return(result)
  }
```

FULL PIPELINE ON FULL DATASET MNIST:

```
# REMOVE SOME PIECES IN THE IMAGES
# traintest
X_train = as.matrix(X.train)
```

```
X_test = as.matrix(X.test)
y_train = as.matrix(y.train)
y_test = as.matrix(y.test)

#cut a piece of image
removed_train = image_edge_deleting(
  X_train,
  'by_percent',
  0.2, 28, 28, width_del_percent=0.6,height_del_percent=0.6)

removed_test= image_edge_deleting(
  X_test,
  'by_percent',
  0.2, 28,28,width_del_percent=0.6,height_del_percent=0.6)

#THIS IS FOR THE VISUALIZATION
train_removed_rows = removed_train$flatten_rows_removed
test_removed_rows = removed_test$flatten_rows_removed


missing.X_train = removed_train$missing_data
missing.X_test =  removed_test$missing_data
# normalization
train_normed = normalizing(x=missing.X_train,Xtrain=missing.X_train)
missing.X_train_normed = train_normed$X_normed
missing.X_train_mean = train_normed$mean
missing.X_train_sd = train_normed$sd

test_normed = normalizing(x=missing.X_test, Xtrain=missing.X_train)
missing.X_test_normed = test_normed$X_normed
missing.X_test_mean = test_normed$mean
missing.X_test_sd = test_normed$sd

#visualize after cutting

visualize_digit(missing.X_train, y_train, train_removed_rows, 2, 6)
```



impute -> reconstruct -> visualize

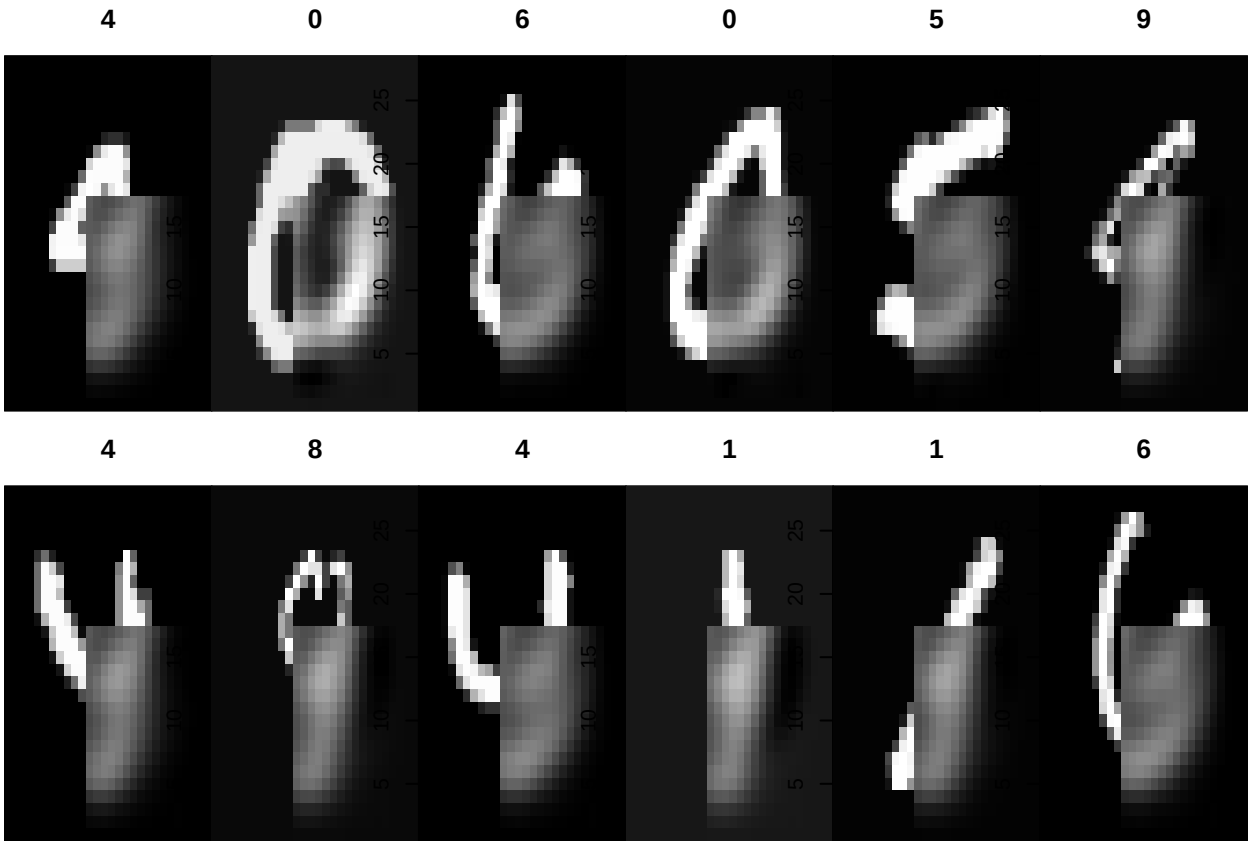
SOFT IMPUTE

```
## Warning in simpute.als(x, J, thresh, lambda, maxit, trace.it, warm.start, :
## Convergence not achieved by 100 iterations
```

```
## Time difference of 25.47634 secs
```

softImpute: reconstruct imputed and visualize

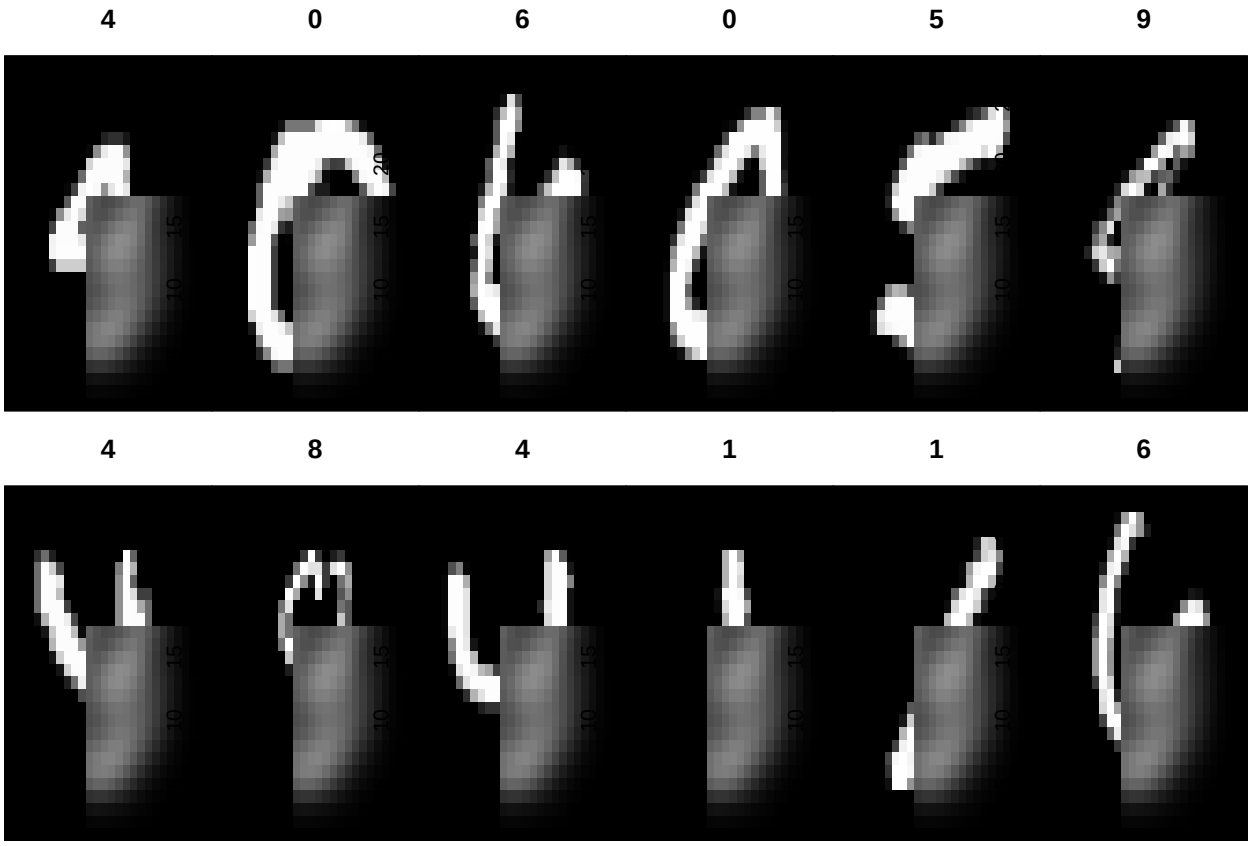
```
softImpute.imp_train = result_softImpute$train
softImpute.imp_test = result_softImpute$test


# resconstructing the original scaled
softImpute.Xrecon = reconstructingNormedMatrix(
  softImpute.imp_test,
  missing.X_test_mean,
  missing.X_test_sd
  )


visualize_digit(softImpute.Xrecon , y_test, test_removed_rows, 2, 6)
```



DIMV

ImpDi: reconstruct imputed and visualize

```
impDi.imp_train = result_impDi$train
impDi.imp_test = result_impDi$test
# resconstructing the original scaled
impDi.Xrecon = reconstructingNormedMatrix(
  impDi.imp_test,
  missing.X_test_mean,
  missing.X_test_sd
  )


visualize_digit(impDi.Xrecon , y_test, test_removed_rows, 2, 6)
```



#PREDICTION

APPLY MODEL (SVM) (NOT TUNNED WITH GRID YET)

```
# softImpute_result = predicting_svm(
#   softImpute.imp_train,
#   y_train,
#   softImpute.imp_test,
#   y_test
# )
# softImpute_result
# ```
#
#
# ```{r}
# impDi_result = predicting_svm(
#   impDi.imp_train,
#   y_train,
#   impDi.imp_test,
#   y_test
# )
# impDi_result
# ```
#
#
# #######  CODE IS ALREADY ENDED  ##########################################################################
```

```
# ######################  THE BELLOW IS JUST RUN ON SAMPLE           ##################################
# ##########################################################################################################
# ##########################################################################################################
# ```{r, cache=TRUE}
# # sampling some data (30%)
# sampled_train = sampling_data(train, 'label', 0.3)
# sample.train = sampled_train$sample
# sample.X_train = sampled_train$X
# sample.y_train = sampled_train$y
#
# sampled_test = sampling_data(test, 'label', 0.3)
# sample.test = sampled_test$sample
# sample.X_test = sampled_test$X
# sample.y_test = sampled_test$y
#
# # REMOVE SOME PIECES IN THE IMAGES
#
# sample.removed_train = image_edge_deleting(
#   sample.X_train,
#   'by_percent',
#   0.2, 28, 28, width_del_percent=0.6,height_del_percent=0.6)
#
# sample.removed_test= image_edge_deleting(
#   sample.X_test,
#   'by_percent',
#   0.2, 28,28,width_del_percent=0.6,height_del_percent=0.6)
#
#
# sample.missing.X_train = sample.removed_train$missing_data
# sample.missing.X_test =  sample.removed_test$missing_data
# #THIS IS FOR THE VISUALIZATION
# sample_train_removed_rows = sample.removed_train$flatten_rows_removed
# sample_test_removed_rows = sample.removed_test$flatten_rows_removed
#
#
# sample_train_normed = normalizing(sample.missing.X_train, sample.missing.X_train)
# sample.missing.X_train_normed = sample_train_normed$X_normed
# sample.missing.X_train_mean = sample_train_normed$mean
# sample.missing.X_train_sd = sample_train_normed$sd
# # sample= sample.missing.X_train_normed[, sample_train_normed$sd_equal_zero_mask, drop=F]
#
# sample_test_normed = normalizing(sample.missing.X_test, sample.missing.X_train)
# sample.missing.X_test_normed = sample_test_normed$X_normed
# sample.missing.X_test_mean = sample_test_normed$mean
# sample.missing.X_test_sd = sample_test_normed$sd
#
# visualize_digit(sample.missing.X_train, sample.y_train, sample_train_removed_rows, 2, 6)
```

```
# sample.softImpute.imp_train = sample_result_softImpute$train
# sample.softImpute.imp_test = sample_result_softImpute$test
# # resconstructing the original scaled
# sample.softImpute.Xrecon = reconstructingNormedMatrix(
#   sample.softImpute.imp_test,
#   apply(sample.missing.X_test, 2, mean, na.rm=TRUE),
```

```
#   apply(sample.missing.X_test, 2, sd, na.rm=TRUE)
#   )
# visualize_digit(sample.softImpute.Xrecon , sample.y_test, sample_test_removed_rows, 2, 6)
```

#USING DIMV

```
# sample.impDi.imp_train = sample_result_impDi$train
# sample.impDi.imp_test = sample_result_impDi$test
# # resconstructing the original scaled
# sample.impDi.Xrecon = reconstructingNormedMatrix(
#   sample.impDi.imp_test,
#   apply(sample.missing.X_test, 2, mean, na.rm=TRUE),
#   apply(sample.missing.X_test, 2, sd, na.rm=TRUE)
#   )
#
#
# visualize_digit(sample.impDi.Xrecon , sample.y_test, sample_test_removed_rows, 2, 6)
```

APPLY MODEL (SVM) (NOT TUNNED WITH GRID YET)

```
# softImpute_result = predicting_svm(
#   sample.softImpute.imp_train,
#   sample.y_train,
#   sample.softImpute.imp_test,
#   sample.y_test
# )
#
```

```
# impDi_result = predicting_svm(
#   sample.impDi.imp_train,
#   sample.y_train,
#   sample.impDi.imp_test,
#   sample.y_test
# )
```