HW1

Ravenclaws

2023-09-24

Q1 - drop NA from me1 and me2

dependencies and env. settings

```
library(reticulate)
use_condaenv('jiaxin')

library(dplyr)
library(tidyr)
library(ggplot2)
library(GGally)
library(e1071)
library(caret)

library(parallel)
numCores = detectCores()

library(doParallel)
registerDoParallel(numCores-1)

library(keras)
library(tensorflow)
tf <- import('tensorflow')
seed = 0590
set.seed(seed)
py_set_seed(seed, disable_hash_randomization = TRUE)

set.a.seed = function(){
  set_random_seed(
    seed,
    disable_gpu = TRUE
  )
}
```

preprocessing

```
## read in data

data = read.csv('sdss-all-c.csv', header = F)
colname = c('id','class','brightness','size','texture','me1','me2')
colnames(data) <- colname
data$class[data$class == 3] <- 0 # galaxy
data$class[data$class == 6] <- 1 # star
data$me1 = as.numeric(data$me1)
data$me2 = as.numeric(data$me2)

## drop NA observations
data_complete7 = data %>% drop_na()
data_complete5 = data[,1:5]

## standardized
data_complete7[,3:7] = scale(data_complete7[,3:7])
data_complete5[,3:5] = scale(data_complete5[,3:5])

## pack datasets
packing = function(data){

  set.a.seed()
  ## randomly partition the dataset (.75 training)
  num = dim(data)[1]
  training_idx = sample(1:num, num*0.75, replace = F)
  data.train <-  data[training_idx,]
  data.test <-  data[-training_idx,]

  ## gathering data
  train.inputs <- data.train[,3:dim(data.train)[2]]
  train.targets = as.matrix(data.train$class)
  test.inputs <- data.test[, 3:dim(data.test)[2]]
  test.targets = as.matrix(data.test$class)

  dataset = list()
  dataset$train = list(input = train.inputs, targets = train.targets)
  dataset$test = list(input = test.inputs, targets = test.targets)
  dataset$trainsvm = data.train
  dataset$testvsm = data.test

  return(dataset)
}

## check classification
table(data_complete7$class)
```

```
##
##    0    1
## 1178  287
```

```
## get dataset
dataset = packing(data_complete7)
str(dataset)
```

```
## List of 4
##  $ train   :List of 2
##   ..$ input  :'data.frame':  1098 obs. of  5 variables:
##   .. ..$ brightness: num [1:1098] -0.188 0.104 0.999 0.704 0.509 ...
##   .. ..$ size      : num [1:1098] 1.6645 -0.0572 0.2818 -0.7612 -0.3632 ...
##   .. ..$ texture   : num [1:1098] -0.118 -0.121 -0.122 -0.12 -0.12 ...
##   .. ..$ me1       : num [1:1098] 0.126 -0.19 -0.47 -0.341 0.18 ...
##   .. ..$ me2       : num [1:1098] 0.506 -0.016 0.377 -0.103 0.234 ...
##   ..$ targets: num [1:1098, 1] 0 0 0 0 0 0 0 0 0 0 ...
##  $ test    :List of 2
##   ..$ input  :'data.frame':  367 obs. of  5 variables:
##   .. ..$ brightness: num [1:367] -2.526 -3.001 -1.565 -0.718 -2.354 ...
##   .. ..$ size      : num [1:367] -0.584 -0.582 -0.568 0.124 -0.582 ...
##   .. ..$ texture   : num [1:367] 0.38 2.391 -0.1 -0.118 0.392 ...
##   .. ..$ me1       : num [1:367] -0.006889 0.000529 -0.007486 -0.016787 -0.022757 ...
##   .. ..$ me2       : num [1:367] -0.0155 -0.0197 -0.0215 0.1036 -0.0266 ...
##   ..$ targets: num [1:367, 1] 1 1 1 0 1 1 1 1 1 1 ...
##  $ trainsvm:'data.frame':    1098 obs. of  7 variables:
##   ..$ id        : chr [1:1098] "77-446" "77-739" "76-1239" "75-592" ...
##   ..$ class     : num [1:1098] 0 0 0 0 0 0 0 0 0 0 ...
##   ..$ brightness: num [1:1098] -0.188 0.104 0.999 0.704 0.509 ...
##   ..$ size      : num [1:1098] 1.6645 -0.0572 0.2818 -0.7612 -0.3632 ...
##   ..$ texture   : num [1:1098] -0.118 -0.121 -0.122 -0.12 -0.12 ...
##   ..$ me1       : num [1:1098] 0.126 -0.19 -0.47 -0.341 0.18 ...
##   ..$ me2       : num [1:1098] 0.506 -0.016 0.377 -0.103 0.234 ...
##  $ testvsm :'data.frame':    367 obs. of  7 variables:
##   ..$ id        : chr [1:367] "75-32" "75-38" "75-42" "75-49" ...
##   ..$ class     : num [1:367] 1 1 1 0 1 1 1 1 1 1 ...
##   ..$ brightness: num [1:367] -2.526 -3.001 -1.565 -0.718 -2.354 ...
##   ..$ size      : num [1:367] -0.584 -0.582 -0.568 0.124 -0.582 ...
##   ..$ texture   : num [1:367] 0.38 2.391 -0.1 -0.118 0.392 ...
##   ..$ me1       : num [1:367] -0.006889 0.000529 -0.007486 -0.016787 -0.022757 ...
##   ..$ me2       : num [1:367] -0.0155 -0.0197 -0.0215 0.1036 -0.0266 ...
```

linear classification by tensorflow

wrapped functions

```
sq_loss = function(targets, pred){
  tf$reduce_mean(tf$square(tf$subtract(targets, pred)))
}

# model
model = function(inputs){tf$matmul(inputs, w) + b}

## run step
run = function(t_input, targets, lr){
  with(tf$GradientTape() %as% tape, {
    pred <- model(t_input)
    loss <- sq_loss(targets, pred)
  })
  update <- tape$gradient(loss, list(w=w, b=b))

  w$assign_sub(update$w * lr)
  b$assign_sub(update$b * lr)
  loss
}

## training
model_train = function(t_input, targets, lr, printwidth=5){
  step = 0
  thres = 1e-6
  curr = 0

  repeat{

    step <- step + 1
    t_loss <- run(t_input, targets, lr)
    loss <- as.numeric(sprintf("%.3f\n", t_loss))

    if (step %% printwidth == 0) cat(sprintf("Loss at step %s: %.6f\n", step, t_loss))
    updates <- abs(loss - curr)

    if (step > 0 & updates <= thres) {
      cat(sprintf('Finish after %s steps.\n', step))
      break
    }
    curr <- loss
  }
}

## evaluation
eval = function(targets, pred){
  mse = sum(targets - pred)^2 / dim(targets)[1]
  pred = ifelse(pred >= 0.5, 1, 0)
```

```
  acc = sum(targets == pred)
  cat(sprintf('Accuracy: %.6f(%s/%s)  MSE: %.6f\n\n', acc/dim(targets)[1],acc, dim(targets)[1], mse))
  targets = as.factor(targets)
  pred = as.factor(round(pred))
  print(confusionMatrix(pred, targets)$table)
}
```

training result (Qa.i)

```
lr_list = c(0.01, 0.05, 0.1, 0.15, 0.2)

input_dim = dim(dataset$train$input)[2]
output_dim = dim(dataset$train$targets)[2]

for (lr in lr_list){
  cat('Learning rate: ', lr,'\n')

  # initialize hidden layer parameters
  set.a.seed()
  w <-  tf$Variable(tf$random$uniform(shape(input_dim, output_dim)))
  b <-  tf$Variable(tf$zeros(shape(output_dim)))
  train.t_input = as_tensor(dataset$train$input, dtype='float32')

  # train
  model_train(train.t_input, dataset$train$targets, lr)

  # eval
  test.t_input = as_tensor(dataset$test$input, dtype='float32')
  prediction = as.vector(model(test.t_input))
  eval(dataset$test$targets, prediction)
  cat('----------------------------------\n')
}
```

```
## Learning rate:  0.01
## Loss at step 5: 0.924639
## Loss at step 10: 0.747607
## Loss at step 15: 0.609703
## Loss at step 20: 0.501660
## Loss at step 25: 0.416504
## Loss at step 30: 0.348972
## Loss at step 35: 0.295077
## Loss at step 40: 0.251789
## Loss at step 45: 0.216799
## Loss at step 50: 0.188336
## Loss at step 55: 0.165038
## Loss at step 60: 0.145852
## Loss at step 65: 0.129960
## Loss at step 70: 0.116724
## Loss at step 75: 0.105641
## Loss at step 80: 0.096316
## Loss at step 85: 0.088432
## Loss at step 90: 0.081739
## Loss at step 95: 0.076034
## Loss at step 100: 0.071153
## Finish after 103 steps.
## Accuracy: 0.888283(326/367)  MSE: 0.127740
##
##           Reference
## Prediction   0   1
##          0 292  39
##          1   2  34
## ----------------------------------
## Learning rate:  0.05
## Loss at step 5: 0.464817
## Loss at step 10: 0.199318
## Loss at step 15: 0.108626
## Loss at step 20: 0.072033
## Loss at step 25: 0.055428
## Loss at step 30: 0.047318
## Finish after 33 steps.
## Accuracy: 0.959128(352/367)  MSE: 0.001036
##
##           Reference
## Prediction   0   1
##          0 294  15
##          1   0  58
## ----------------------------------
## Learning rate:  0.1
## Loss at step 5: 0.214873
## Loss at step 10: 0.073169
## Loss at step 15: 0.047319
## Loss at step 20: 0.040912
## Finish after 22 steps.
## Accuracy: 0.972752(357/367)  MSE: 0.018473
##
```

```
##           Reference
## Prediction   0   1
##          0 294  10
##          1   0  63
## ----------------------------------
## Learning rate:  0.15
## Loss at step 5: 0.116835
## Loss at step 10: 0.047304
## Loss at step 15: 0.039687
## Finish after 15 steps.
## Accuracy: 0.975477(358/367)  MSE: 0.021835
##
##           Reference
## Prediction   0   1
##          0 294   9
##          1   0  64
## ----------------------------------
## Learning rate:  0.2
## Loss at step 5: 0.075587
## Loss at step 10: 0.040731
## Finish after 13 steps.
## Accuracy: 0.980926(360/367)  MSE: 0.027474
##
##           Reference
## Prediction   0   1
##          0 294   7
##          1   0  66
## ----------------------------------
```

According to the evaluation result, the best linear classification model achieves an 98.1% accuracy on the test dataset with learning rate = 0.2 and updates only for 13 steps. According to the change in confusion matrix, increasing learning rate from 0.01 to 0.2 enables the model to recall some false negative predictions.

classification by svm (Qa.ii, Qa.iii)

```
formula = class~brightness+size+texture+me1+me2

set.seed(seed)
kernels = c('linear', 'radial','polynomial','sigmoid')
costs = c(0.001, 0.01, 0.1, 1,5,10,100)


# parallel tuning process
params = expand.grid(kernel = kernels, cost = costs)
results = list()
foreach(i = 1:nrow(params), .packages = 'e1071') %do% {
    current_cost = params$cost[i]
    current_kernel = params$kernel[i]
    curResult <- tune(method = svm,
                      train.x = formula,
                      data = dataset$trainsvm,
                      kernel = as.character(current_kernel),
                      cost = current_cost)
    results[[i]] <- c(current_kernel, current_cost, curResult$performances$error)
}
```

```
## [[1]]
## [1] 1.000000 0.001000 0.112261
##
## [[2]]
## [1] 2.0000000 0.0010000 0.1670796
##
## [[3]]
## [1] 3.0000000 0.0010000 0.1600346
##
## [[4]]
## [1] 4.0000000 0.0010000 0.1658615
##
## [[5]]
## [1] 1.00000000 0.01000000 0.04292841
##
## [[6]]
## [1] 2.00000000 0.01000000 0.07274986
##
## [[7]]
## [1] 3.0000000 0.0100000 0.1088974
##
## [[8]]
## [1] 4.0000000 0.0100000 0.0922463
##
## [[9]]
## [1] 1.00000000 0.10000000 0.04047963
##
## [[10]]
## [1] 2.00000000 0.10000000 0.01899529
##
## [[11]]
## [1] 3.0000000 0.1000000 0.1124262
##
## [[12]]
## [1] 4.0000000 0.1000000 0.2156642
##
## [[13]]
## [1] 1.00000000 1.00000000 0.04031662
##
## [[14]]
## [1] 2.000000000 1.000000000 0.008682732
##
## [[15]]
## [1] 3.0000000 1.0000000 0.2578538
##
## [[16]]
## [1]  4.00000  1.00000 34.78067
##
## [[17]]
## [1] 1.00000000 5.00000000 0.04022732
##
## [[18]]
```

```
## [1] 2.000000000 5.000000000 0.006386856
##
## [[19]]
## [1] 3.0000000 5.0000000 0.3200779
##
## [[20]]
## [1]    4.000    5.000 1032.743
##
## [[21]]
## [1]  1.00000000 10.00000000  0.04054598
##
## [[22]]
## [1]  2.000000000 10.000000000  0.005846379
##
## [[23]]
## [1]  3.000000 10.000000  1.919723
##
## [[24]]
## [1]    4.000   10.000 5020.248
##
## [[25]]
## [1]   1.00000000 100.00000000   0.04057071
##
## [[26]]
## [1] 2.000000e+00 1.000000e+02 5.673254e-03
##
## [[27]]
## [1]   3.000000 100.000000   2.112995
##
## [[28]]
## [1]      4.0    100.0 428140.4
```

```
tuning_result = data.frame(Kernel = levels(params$kernel)[results[[i]]][1],
              Cost = results[[i]][2],
              Error = results[[i]][3])

# packing results
for (i in 2:nrow(params)){
  tuning_result = rbind(tuning_result,
    c(levels(params$kernel)[results[[i]]][1],
              results[[i]][2],
              results[[i]][3]))
}

tuning_result$Kernel = as.factor(tuning_result$Kernel)
tuning_result$Cost = as.factor(tuning_result$Cost)
tuning_result$Error = as.numeric(tuning_result$Error)

tuning_result[order(tuning_result$Error, decreasing = F),]
```

	Kernel <fct>	Cost <fct>	Error <dbl>
26	radial	100	5.673254e-03
22	radial	10	5.846379e-03
18	radial	5	6.386856e-03
14	radial	1	8.682732e-03
10	radial	0.1	1.899529e-02
17	linear	5	4.022732e-02
13	linear	1	4.031662e-02
9	linear	0.1	4.047963e-02
21	linear	10	4.054598e-02
25	linear	100	4.057071e-02

1-10 of 28 rows Previous **1** 2 3 Next

Use 10-fold cross validation to calculate the average error of each model (under different parameter combination). According to the tuning result, radial kernel seperates the data best. And the best cost parameter is 100.

```
best.svm = svm(formula, dataset$trainsvm, kernel = 'radial', cost = 100)
summary(best.svm)
```

```
##
## Call:
## svm(formula = formula, data = dataset$trainsvm, kernel = "radial",
##     cost = 100)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.2
##     epsilon:  0.1
##
##
## Number of Support Vectors:  286
```

```
pred = predict(best.svm, dataset$testvsm)
eval(dataset$testvsm$class, pred)
```

```
##           Reference
## Prediction   0   1
##          0 294   0
##          1   0  73
```

Using the best model to evaluate model performance on test dataset. By checking the confusion matrix, the model achieves 100% accuracy on the test dataset.

Q1 - drop column me1 and me2 (Qb)

```
dataset = packing(data_complete5)
```

classification by network

```
lr_list = c(0.01, 0.05, 0.1, 0.15, 0.2)
input_dim = dim(dataset$train$input)[2]
output_dim = dim(dataset$train$targets)[2]

  for (lr in lr_list){
    cat('Learning rate: ', lr,'\n')

    # initialize hidden layer parameters
    set.a.seed()
    w <-  tf$Variable(tf$random$uniform(shape(input_dim, output_dim)))
    b <-  tf$Variable(tf$zeros(shape(output_dim)))
    train.t_input = as_tensor(dataset$train$input, dtype='float32')

    # train
    model_train(train.t_input, dataset$train$targets, lr)

    # eval
    test.t_input = as_tensor(dataset$test$input, dtype='float32')
    prediction = as.vector(model(test.t_input))
    eval(dataset$test$targets, prediction)
    cat('-----------------------------------\n')
  }
```

```
## Learning rate:  0.01
## Loss at step 5: 0.847400
## Loss at step 10: 0.675107
## Loss at step 15: 0.543668
## Loss at step 20: 0.442800
## Loss at step 25: 0.364888
## Loss at step 30: 0.304282
## Loss at step 35: 0.256782
## Loss at step 40: 0.219258
## Loss at step 45: 0.189369
## Loss at step 50: 0.165362
## Loss at step 55: 0.145914
## Loss at step 60: 0.130027
## Loss at step 65: 0.116941
## Loss at step 70: 0.106078
## Loss at step 75: 0.096993
## Loss at step 80: 0.089340
## Loss at step 85: 0.082853
## Loss at step 90: 0.077321
## Loss at step 95: 0.072579
## Finish after 95 steps.
## Accuracy: 0.862069(325/377)  MSE: 0.469356
##
##           Reference
## Prediction   0   1
##          0 299  52
##          1   0  26
## ------------------------------------
## Learning rate:  0.05
## Loss at step 5: 0.407417
## Loss at step 10: 0.174044
## Loss at step 15: 0.099457
## Loss at step 20: 0.069361
## Loss at step 25: 0.055135
## Loss at step 30: 0.047836
## Finish after 34 steps.
## Accuracy: 0.949602(358/377)  MSE: 0.004470
##
##           Reference
## Prediction   0   1
##          0 299  19
##          1   0  59
## ------------------------------------
## Learning rate:  0.1
## Loss at step 5: 0.186121
## Loss at step 10: 0.070509
## Loss at step 15: 0.047907
## Loss at step 20: 0.041772
## Finish after 20 steps.
## Accuracy: 0.973475(367/377)  MSE: 0.000710
##
##           Reference
```

```
## Prediction   0   1
##          0 299  10
##          1   0  68
## ------------------------------------
## Learning rate:  0.15
## Loss at step 5: 0.106171
## Loss at step 10: 0.047969
## Loss at step 15: 0.040557
## Finish after 15 steps.
## Accuracy: 0.976127(368/377)  MSE: 0.004903
##
##           Reference
## Prediction   0   1
##          0 299   9
##          1   0  69
## ------------------------------------
## Learning rate:  0.2
## Loss at step 5: 0.073120
## Loss at step 10: 0.041610
## Finish after 13 steps.
## Accuracy: 0.978780(369/377)  MSE: 0.009468
##
##           Reference
## Prediction   0   1
##          0 299   8
##          1   0  70
## ------------------------------------
```

According to the evaluation result, the best linear classification model achieves a 97.9% accuracy on the test dataset with learning rate = 0.2 and updates only for 13 steps. According to the change in confusion matrix, increasing learning rate from 0.01 to 0.2 enables the model to recall some false negative predictions. Comparing with Qa's data cleaning strategy, utilizing the information in variable me1 and me2 in linear classification can increase model's accuracy by around 1%.

classification by svm

```
formula = class~brightness+size+texture

set.seed(seed)
kernels = c('linear', 'radial','polynomial','sigmoid')
costs = c(0.001, 0.01, 0.1, 1,5,10,100)


# parallel tuning process
params = expand.grid(kernel = kernels, cost = costs)
results = list()
foreach(i = 1:nrow(params), .packages = 'e1071') %do% {
    current_cost = params$cost[i]
    current_kernel = params$kernel[i]
    curResult <- tune(method = svm,
                      train.x = formula,
                      data = dataset$trainsvm,
                      kernel = as.character(current_kernel),
                      cost = current_cost)
    results[[i]] <- c(current_kernel, current_cost, curResult$performances$error)
}
```

```
## [[1]]
## [1] 1.0000000 0.0010000 0.1126463
##
## [[2]]
## [1] 2.000000 0.001000 0.155732
##
## [[3]]
## [1] 3.000000 0.001000 0.118142
##
## [[4]]
## [1] 4.0000000 0.0010000 0.1553271
##
## [[5]]
## [1] 1.00000000 0.01000000 0.04373643
##
## [[6]]
## [1] 2.0000000 0.0100000 0.0504816
##
## [[7]]
## [1] 3.00000000 0.01000000 0.08488147
##
## [[8]]
## [1] 4.00000000 0.01000000 0.09005643
##
## [[9]]
## [1] 1.00000000 0.10000000 0.04143525
##
## [[10]]
## [1] 2.00000000 0.10000000 0.01309013
##
## [[11]]
## [1] 3.0000000 0.1000000 0.1730617
##
## [[12]]
## [1] 4.0000000 0.1000000 0.7721243
##
## [[13]]
## [1] 1.00000000 1.00000000 0.04114575
##
## [[14]]
## [1] 2.000000000 1.000000000 0.005843049
##
## [[15]]
## [1] 3.0000000 1.0000000 0.6051436
##
## [[16]]
## [1]  4.00000  1.00000 56.58255
##
## [[17]]
## [1] 1.00000000 5.00000000 0.04159578
##
## [[18]]
```

```
## [1] 2.000000000 5.000000000 0.004194124
##
## [[19]]
## [1] 3.0000000 5.0000000 0.6953375
##
## [[20]]
## [1]    4.000    5.000 1419.663
##
## [[21]]
## [1]  1.00000000 10.00000000  0.04121756
##
## [[22]]
## [1]  2.000000000 10.000000000  0.004331692
##
## [[23]]
## [1]  3.0000000 10.0000000  0.5852891
##
## [[24]]
## [1]    4.000   10.000 5379.161
##
## [[25]]
## [1]   1.00000000 100.00000000   0.04113837
##
## [[26]]
## [1] 2.000000e+00 1.000000e+02 2.735516e-03
##
## [[27]]
## [1]   3.0000000 100.0000000   0.6034993
##
## [[28]]
## [1]      4.0    100.0 755142.7
```

```
tuning_result = data.frame(Kernel = levels(params$kernel)[results[[i]]][1],
              Cost = results[[i]][2],
              Error = results[[i]][3])

# packing results
for (i in 2:nrow(params)){
  tuning_result = rbind(tuning_result,
    c(levels(params$kernel)[results[[i]]][1],
              results[[i]][2],
              results[[i]][3]))
}

tuning_result$Kernel = as.factor(tuning_result$Kernel)
tuning_result$Cost = as.factor(tuning_result$Cost)
tuning_result$Error = as.numeric(tuning_result$Error)

tuning_result[order(tuning_result$Error, decreasing = F),]
```

	Kernel <fct>	Cost <fct>	Error <dbl>
26	radial	100	2.735516e-03
18	radial	5	4.194124e-03
22	radial	10	4.331692e-03
14	radial	1	5.843049e-03
10	radial	0.1	1.309013e-02
25	linear	100	4.113837e-02
13	linear	1	4.114575e-02
21	linear	10	4.121756e-02
9	linear	0.1	4.143525e-02
17	linear	5	4.159578e-02

1-10 of 28 rows Previous **1** 2 3 Next

```
best.svm = svm(formula, dataset$trainsvm, kernel = 'radial', cost = 100)
summary(best.svm)
```

```
##
## Call:
## svm(formula = formula, data = dataset$trainsvm, kernel = "radial",
##     cost = 100)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.3333333
##     epsilon:  0.1
##
##
## Number of Support Vectors:  184
```

```
pred = predict(best.svm, dataset$testvsm)
eval(dataset$testvsm$class, pred)
```

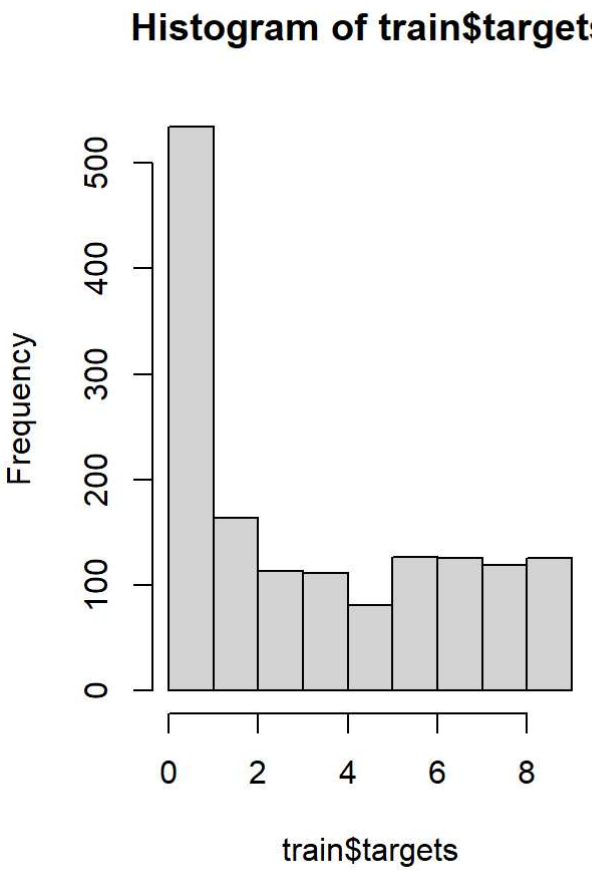
```
##           Reference
## Prediction   0   1
##          0 299   0
##          1   0  78
```

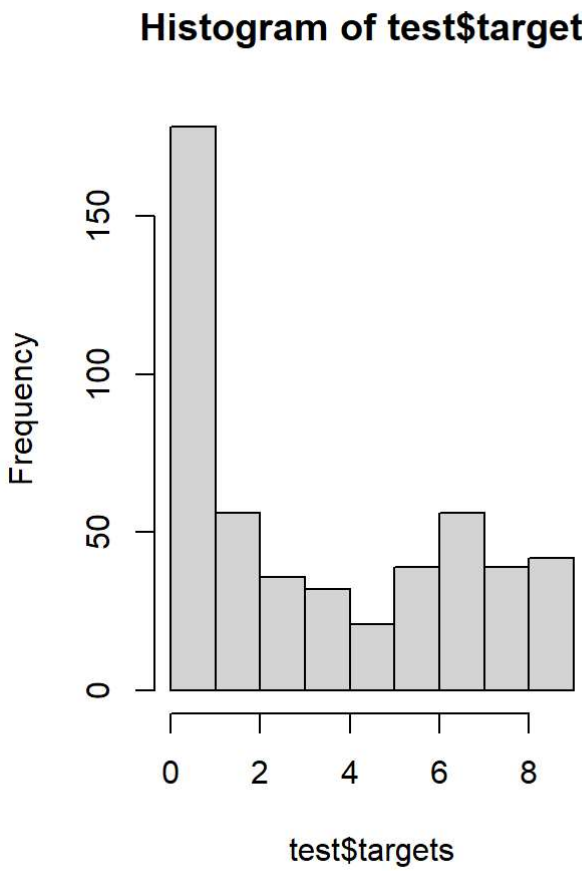
Use 10-fold cross validation to calculate the average error of each model (under different parameter combination). According to the tuning result, radial kernel seperates the data best. And the best cost parameter is 100. The best model also achieves 100% accuracy on test dataset. Two different data cleaning strategies do not influence the model performance of svm. Since under each strategy, svm can seperate the data perfectly.

Q2

preprocessing

```
x = read.table('ziptrain.dat')
y = read.table('zipdigit.dat')

## reformat input (2000 rows, 256 columns)
data <- cbind(y, x)



## create 75 25 split by class
set.seed(seed)
split <- createDataPartition(y = data[,1], p = 0.75) |> unlist()
train <- data[split,]
test <- data[-split,]
rm(split, data)

## set up data sets, scale training data set
train <- list(input = train[,2:ncol(train)], targets = train[,1])
train[['input']] <- train[['input']] |> sapply(as.numeric) |> as.matrix()
train[['targets']] <- train[['targets']] |> as.matrix()
scale <- train[['input']] |> preProcess(method = c('center', 'scale'))
train[['input']] <- predict(scale, train[['input']])

## optionally scale test data based on training data
test <- list(input = test[,2:ncol(test)], targets = test[,1])
test[['input']] <- test[['input']] |> sapply(as.numeric) |> as.matrix()
test[['targets']] <- test[['targets']] |> as.matrix()
test[['input']] <- predict(scale, test[['input']]) |> as.matrix()

## make sure distribution of digits is similar between training and test sets
par(mfrow = c(1,2))
hist(train$targets)
hist(test$targets)
```


Histogram of train$targets
Frequency
0
100
200
300
400
500
0
2
4
6
8
train$targets


Histogram of test$targets
Frequency
0
50
100
150
0
2
4
6
8
test$targets

##

PCA(Qa.i)

```
# wrapped function
summary_image_by = function(func){
  summary_img = array(dim = c(10, 256))
  for (i in 0:9){
  summary_img[i+1,] <- apply(x[which(y == i),], 2, func)
  }
  summary_img
}

mean_img = summary_image_by(mean)

# remove mean effect
mean_effect = array(dim = c(nrow(x), 256))
for (i in 0:9){
  nrows = length(which(y == i))
  mean_effect[which(y == i),] <- matrix(rep(mean_img[i+1,], nrows), nrow = nrows, byrow = T)
}

cleaned_zip_img = x - mean_effect

zip.pca = prcomp(cleaned_zip_img)

curr_var = 0
Divider = TRUE
total_var = sum((zip.pca$sdev)^2)
compress_idx = 1

cat("Cumulative Importance:\n")
```

```
## Cumulative Importance:
```

```
for (i in 1:length(zip.pca$sdev)){
  curr_var = curr_var + zip.pca$sdev[i]^2
  cat(sprintf("%s - %.2f\n", i, curr_var/total_var*100))
  if (curr_var / total_var > 0.8 & Divider){
    cat('----------------------------------------\n')
    Divider = FALSE
    compress_idx = i
    break
  }
}
```

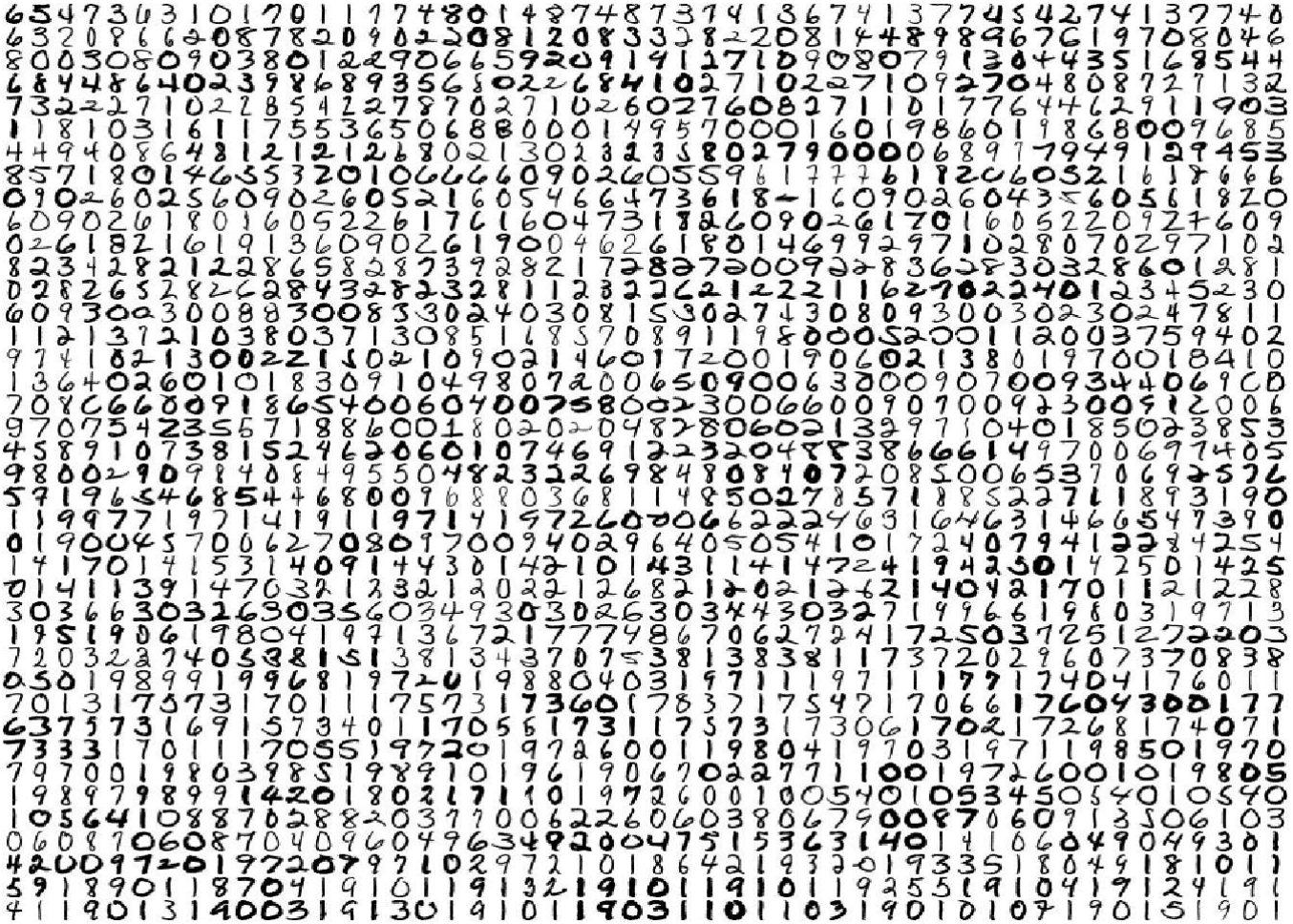
```
## 1 - 10.51
## 2 - 18.39
## 3 - 24.13
## 4 - 28.89
## 5 - 33.33
## 6 - 37.15
## 7 - 40.64
## 8 - 43.29
## 9 - 45.73
## 10 - 48.03
## 11 - 50.20
## 12 - 52.10
## 13 - 53.93
## 14 - 55.68
## 15 - 57.31
## 16 - 58.85
## 17 - 60.36
## 18 - 61.72
## 19 - 63.02
## 20 - 64.31
## 21 - 65.49
## 22 - 66.63
## 23 - 67.73
## 24 - 68.76
## 25 - 69.68
## 26 - 70.58
## 27 - 71.44
## 28 - 72.30
## 29 - 73.14
## 30 - 73.94
## 31 - 74.71
## 32 - 75.45
## 33 - 76.16
## 34 - 76.86
## 35 - 77.50
## 36 - 78.12
## 37 - 78.71
## 38 - 79.30
## 39 - 79.86
## 40 - 80.40
## ----------------------------------------
```

After removing the effect of digit-specific means and applying PCA, the fist 46 components can explain at least 80% of the total variance in the training dataset.

low-rank representation (Qa.ii)

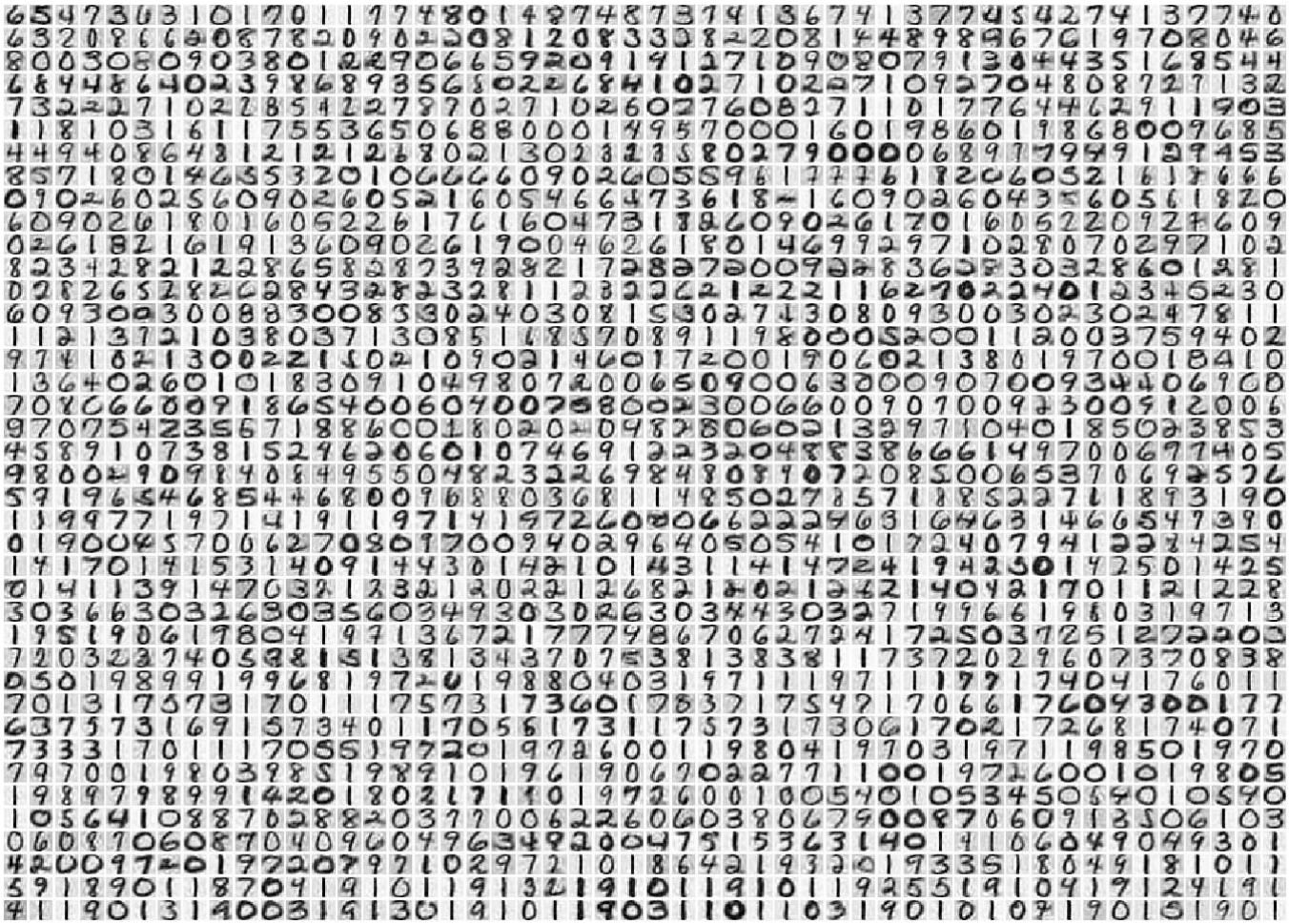
```
# wrapped function
replace_d = function(d,k){
  d_k = diag(0, nrow = length(d))
  d_k <- rep(0, length(d))
  d_k[1:k] <- d[1:k]
  diag(d_k)
}

renewed_img_of = function(d_k){
  decomp_img = svd_res$u %*% d_k %*% t(svd_res$v)
  renewed_img = decomp_img + mean_effect
  dim(renewed_img) <- c(nrow(x), 16, 16)
  renewed_img
}
```

```
# svd
svd_res = svd(cleaned_zip_img)

# display
d_k = replace_d(svd_res$d, 40)
new_img = renewed_img_of(d_k)

# raw form
par(mar = rep(0.05,4), mfrow = c(40,50))
raw_img = as.matrix(x)
dim(raw_img) <- c(nrow(x), 16, 16)
apply(raw_img, 1, function(x) {image(x[,16:1], axes = F, col = gray(31:0/31))})
```



```
## NULL
```

```
# low-rank form
par(mar = rep(0.05,4), mfrow = c(40,50))
apply(new_img, 1, function(x) {image(x[,16:1], axes = F, col = gray(31:0/31))})
```



```
## NULL
```

By applying PCA to obtain the low-rank representation of the digits, the comparison with raw digit images implys that the PCA can still maintain the shape of each digit, while it may lose some information about the boundary between the digit and its background, reflecting by the vagueness in low-rank representation.

classification performance (Qb.i)

```
# evaluation function for 10 classes
eval10 = function(targets, pred){
  mse = sum(targets - pred)^2 / dim(targets)[1]
  pred = ifelse(pred >= 9, 9, ifelse(pred <= 0, 0, pred))
  pred = round(pred)
  acc = sum(targets == pred)
  cat(sprintf('Accuracy: %.6f(%s/%s)  MSE: %.6f\n\n', acc/dim(targets)[1],acc, dim(targets)[1], ms
e))
  targets = as.factor(targets)
  pred = as.factor(round(pred))
  print(confusionMatrix(pred, targets)$table)
}

# packing data
dataset = list()
dataset$train = train
dataset$test = test
```

linear classification

```
lr_list = c(5e-4, 1e-3, 5e-3, 0.01, 0.02)

input_dim = dim(dataset$train$input)[2]
output_dim = dim(dataset$train$targets)[2]

for (lr in lr_list){
  cat('Learning rate: ', lr,'\n')

  # initialize hidden layer parameters
  set.a.seed()
  w <-  tf$Variable(tf$random$uniform(shape(input_dim, output_dim)))
  b <-  tf$Variable(tf$zeros(shape(output_dim)))
  train.t_input = as_tensor(dataset$train$input, dtype='float32')

  # train
  model_train(train.t_input, dataset$train$targets, lr, printwidth = 100)

  # eval
  test.t_input = as_tensor(dataset$test$input, dtype='float32')
  prediction = as.vector(model(test.t_input))
  eval10(dataset$test$targets, prediction)
  cat('----------------------------------\n')
}
```

```
## Learning rate:  5e-04
## Loss at step 100: 71.352928
## Loss at step 200: 31.162800
## Loss at step 300: 20.819736
## Loss at step 400: 16.445597
## Loss at step 500: 13.808612
## Loss at step 600: 11.920169
## Loss at step 700: 10.468922
## Loss at step 800: 9.318789
## Loss at step 900: 8.392312
## Loss at step 1000: 7.637970
## Loss at step 1100: 7.018681
## Loss at step 1200: 6.506660
## Loss at step 1300: 6.080601
## Loss at step 1400: 5.723929
## Loss at step 1500: 5.423608
## Loss at step 1600: 5.169303
## Loss at step 1700: 4.952754
## Loss at step 1800: 4.767328
## Loss at step 1900: 4.607664
## Loss at step 2000: 4.469415
## Loss at step 2100: 4.349038
## Loss at step 2200: 4.243637
## Finish after 2229 steps.
## Accuracy: 0.290581(145/499)  MSE: 66.860890
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 41 19 15  1  1  0  0  0  0  0
##          1 22 55  3  4  2  0  2  1  2  1
##          2 13 11  7 10  1  1  1  1  1  1
##          3 11  1 12  7  2  2  9  1  4  0
##          4  1  0  9  8  3  5 12  4  3  1
##          5  2  1  5  3  7  4 11  8  8  6
##          6  0  0  3  2  4  6  3 17 10  6
##          7  1  0  1  1  6  2  1 16  7 12
##          8  0  0  1  0  3  1  0  3  3  9
##          9  0  0  0  0  3  0  0  5  1  6
## ---------------------------------
## Learning rate:  0.001
## Loss at step 100: 31.223835
## Loss at step 200: 16.462408
## Loss at step 300: 11.930202
## Loss at step 400: 9.324646
## Loss at step 500: 7.641434
## Loss at step 600: 6.508751
## Loss at step 700: 5.725228
## Loss at step 800: 5.170140
## Loss at step 900: 4.767887
## Loss at step 1000: 4.469810
## Loss at step 1100: 4.243931
## Loss at step 1200: 4.068899
```

```
## Loss at step 1300: 3.930245
## Loss at step 1400: 3.818043
## Finish after 1432 steps.
## Accuracy: 0.300601(150/499)  MSE: 18.404907
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 36 14 12  1  0  0  0  0  0  0
##          1 24 57  6  2  3  0  2  0  1  0
##          2 16 13  5 11  1  0  1  2  1  2
##          3  9  2  8 10  2  3  6  0  3  0
##          4  3  0 14  5  1  3 15  4  5  1
##          5  2  1  4  4  8  6 11  6  6  4
##          6  0  0  5  2  5  5  4 14 10  5
##          7  1  0  1  1  5  3  0 20  7 13
##          8  0  0  1  0  4  1  0  5  5 11
##          9  0  0  0  0  3  0  0  5  1  6
## ------------------------------------
## Learning rate:  0.005
## Loss at step 100: 7.669304
## Loss at step 200: 4.472988
## Loss at step 300: 3.726601
## Loss at step 400: 3.427592
## Loss at step 500: 3.257137
## Loss at step 600: 3.142018
## Finish after 612 steps.
## Accuracy: 0.282565(141/499)  MSE: 0.274156
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 33  9  9  0  1  0  1  0  0  0
##          1 25 51  5  3  1  0  1  0  0  0
##          2 18 23  9 10  2  1  0  0  1  0
##          3  8  2  8 10  2  4  3  2  3  2
##          4  6  1 12  6  1  2 19  1  4  1
##          5  1  1  6  5  6  6 11 10  4  1
##          6  0  0  5  2  6  5  3 12 13  7
##          7  0  0  1  0  5  2  1 17  8 11
##          8  0  0  0  0  6  1  0  8  6 15
##          9  0  0  1  0  2  0  0  6  0  5
## ------------------------------------
## Learning rate:  0.01
## Loss at step 100: 4.476960
## Loss at step 200: 3.428589
## Loss at step 300: 3.142545
## Loss at step 400: 2.992941
## Finish after 453 steps.
## Accuracy: 0.310621(155/499)  MSE: 0.409410
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 34  5  8  0  1  0  2  0  0  0
```

```
##          1 30 55  5  3  1  0  0  0  0  0
##          2 12 23 10 10  2  1  1  0  1  0
##          3  8  2  9 10  2  5  4  2  1  1
##          4  6  1 11  6  3  1 16  1  5  2
##          5  1  1  5  5  4  7 11 11  4  1
##          6  0  0  6  2  6  5  4 10 11  6
##          7  0  0  1  0  5  1  1 18 10 12
##          8  0  0  0  0  5  1  0  6  7 13
##          9  0  0  1  0  3  0  0  8  0  7
## ------------------------------------
## Learning rate:  0.02
## Loss at step 100: 3.430590
## Loss at step 200: 2.993566
## Loss at step 300: 2.835768
## Finish after 318 steps.
## Accuracy: 0.314629(157/499)  MSE: 0.695320
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 35  7  7  0  1  0  2  0  0  0
##          1 29 54  7  4  2  0  0  0  0  0
##          2 13 22  9  8  1  2  1  0  1  0
##          3  8  3  9 12  2  3  5  2  1  1
##          4  4  0 11  7  4  2 16  2  6  2
##          5  2  1  4  3  3  6 10  9  3  2
##          6  0  0  7  2  8  6  5 12 10  5
##          7  0  0  1  0  3  1  0 17 10 12
##          8  0  0  0  0  5  1  0  8  8 13
##          9  0  0  1  0  3  0  0  6  0  7
## ------------------------------------
```

According to the evaluation result, the best linear classification model achieves a 31.5% accuracy on the test dataset with learning rate = 0.02 and updates for 318 steps (tuning learning rate from 5e-4 to 0.02, since training encountered gradient exploding with use lr =0.05). It implys that the linear classification model performs not well on digit dataset.

svm

```
colnames(dataset$train$targets) <- c('class')
colnames(dataset$test$targets) <- c('class')
dataset$trainsvm = data.frame(cbind(dataset$train$targets, dataset$train$input))
dataset$testsvm = data.frame(cbind(dataset$test$targets, dataset$test$input))

set.seed(seed)
kernels = c('linear', 'radial','polynomial','sigmoid')
costs = c(0.001, 0.01, 0.1, 1,5,10,100)



# parallel tuning process
## run each model type with generic cost = 10

results = list()
foreach(i = 1:4, .packages = 'e1071') %do% {
    current_kernel = kernels[i]
    curResult <- tune(method = svm,
                      train.x = class~.,
                      data = dataset$trainsvm,
                      kernel = as.character(current_kernel),
                      cost = 10)
    results[[i]] <- c(current_kernel, 10, curResult$performances$error)
}
```

```
## [[1]]
## [1] "linear"           "10"               "4.56010835336013"
##
## [[2]]
## [1] "radial"            "10"                "0.986574213712546"
##
## [[3]]
## [1] "polynomial"       "10"               "1.35209299278766"
##
## [[4]]
## [1] "sigmoid"          "10"               "8106.70239085516"
```

```
tuning_result = data.frame(Kernel = results[[1]][1],
              Cost = results[[1]][2],
              Error = results[[1]][3])

# packing results
for (i in 2:length(kernels)){
  tuning_result = rbind(tuning_result,
    c(results[[i]][1],
              results[[i]][2],
              results[[i]][3]))
}

tuning_result$Kernel = as.factor(tuning_result$Kernel)
tuning_result$Cost = as.factor(tuning_result$Cost)
tuning_result$Error = as.numeric(tuning_result$Error)

tuning_result[order(tuning_result$Error, decreasing = F),]
```

	Kernel <fct>	Cost <fct>	Error <dbl>
2	radial	10	0.9865742
3	polynomial	10	1.3520930
1	linear	10	4.5601084
4	sigmoid	10	8106.7023909

4 rows

```
## further tuning
svmfit  <- tune(method = svm,
                train.x = dataset$train$input,
                train.y = dataset$train$targets,
                ranges = list(cost = c(0.1,  1,5,10,100),
                              kernel = c('linear', 'radial', 'polynomial', 'sigmoid')),
                scale = FALSE)


summary(svmfit$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = dataset$train$input, train.y = dataset$train$targets,
##     ranges = list(cost = c(0.1, 1, 5, 10, 100), kernel = c("linear",
##         "radial", "polynomial", "sigmoid")), scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.00390625
##     epsilon:  0.1
##
##
## Number of Support Vectors:  1217
```

```
best.svm = svm(class~., dataset$trainsvm, kernel = 'radial', cost = 100)
pred = predict(best.svm, dataset$testsvm)
eval10(dataset$test$targets, pred)
```

```
## Accuracy: 0.609218(304/499)  MSE: 0.493060
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 67  0  0  0  0  0  0  0  0  0
##          1 18 82  2  0  0  0  0  0  0  0
##          2  4  4 24  3  0  0  0  0  0  0
##          3  2  1 18 18  4  0  1  0  0  0
##          4  0  0  7 11 16  7  5  1  2  0
##          5  0  0  4  3  9  5 11  6  2  1
##          6  0  0  1  1  1  7 19  6  5  3
##          7  0  0  0  0  2  2  3 37  9  2
##          8  0  0  0  0  0  0  0  6 15 15
##          9  0  0  0  0  0  0  0  0  6 21
```

By grid-search and 10-fold cross validation, the best svm classifier uses radial kernel with cost parameter equal to 100. The best model has 60.9% accuracy on test dataset.