Homework #7: SVM and Generative Classifiers

Ben Wilson

Due: Wed Oct 26 | 11:45am

Contents

DS 6030 | Fall 2022 | University of Virginia

Required R packages and Directories

```
data_dir = 'https://mdporter.github.io/DS6030/data/' # data directory
library(R6030)     # functions for DS-6030
library(tidyverse) # functions for data manipulation
library(e1071)     # svm functions

# Add other libraries here
library(plyr)
library(mvtnorm)
library(yardstick)
```

Problem 1: Handwritten Digit Recognition

a. Load the MNIST training and testing data.

The data are `.rds` format. Training data has 1000 samples from each class. The test data has only one sample from each class. Training Data Testing Data

```r
#load rds data
df_test <- readRDS("mnist_test.rds")
df_train <- readRDS("mnist_train.rds")
```

b. Quadratic Discriminant Analysis (QDA)

Implement quadratic discriminant analysis (QDA) step-by-step (i.e., manually).

```r
## create matrix to capture values for digits 0-9
val = matrix(ncol=10, nrow=nrow(df_test))

## function to iterate across digits 0-9 in data
for (i in 0:9){

  #create subset of train without label
  train_var = df_train %>%
    filter(label == i) %>%
    select(-label)

  #create subset of test without label
  test_var = df_test %>%
    select(-label)

  #calculate means for training subset
  means = colMeans(train_var)

  #calculate prior prob
    #use training subset and training data
  prior = nrow(train_var)/nrow(df_train)

  #calculate covariance values
  covariance = cov(train_var)

  #iterate over test data subset
  for (j in 1:nrow(test_var)){
    test_row = test_var[j,]

    #calculate QDA using mathematical equation and model inputs
    val[j,i+1] = det(covariance)^(-1/2)*(2*pi)^-(ncol(train_var)/2)*exp(-1/2*data.matrix(test_row - mea
  }

  #input values to matrix for capturing values
  colnames(val) <- c(0:9)

  #calculate normalized prob
  probs = val

  for (k in 1:nrow(val)){
    probs[k,] = val[k,]/sum(val[k,])
  }

  #identify acuracy
  colnames(probs)[max.col(probs,ties.method="first")]
```

```r
  analysis = cbind(df_test, "pred" = colnames(probs)[max.col(probs,ties.method="first")])
}

#calculate for misclassification rate
analysis$misclass = ifelse(analysis$label == analysis$pred, 0, 1)

## Misclass rate
sum(analysis$misclass)/nrow(analysis)
```

```
#> [1] 0
```

Problem 2: One-vs-Rest

a. Support Vector Machines (SVM) for 2-class problem

```r
#create copy of training data for manipulation
df_train_dig0 = df_train

#identify 0 digit for fitting model
df_train_dig0$label = ifelse(df_train_dig0$label == 0, 1, -1)
df_train_dig0$label = factor(df_train_dig0$label)

#fit training data for model
fit = svm(label ~ ., data = df_train_dig0,

          #radial basis svm turning
          kernel = "radial",

          #cost given for problem
          cost = 100,

          #probability given for problem
          probability = TRUE)

#perform predictions on test data based on trained data
pred = predict(fit, df_test%>% select(-label), probability=TRUE) %>%
  attr("probabilities")

#predict output
pred
```

```
#>          -1         1
#> 1  0.0000382 1.000e+00
#> 2  0.9999988 1.153e-06
#> 3  0.9999822 1.783e-05
#> 4  0.9999987 1.314e-06
#> 5  0.9999987 1.270e-06
#> 6  0.9998218 1.782e-04
#> 7  0.9987080 1.292e-03
#> 8  0.9984856 1.514e-03
```

```
#> 9  0.9999997 3.398e-07
#> 10 0.9999845 1.545e-05
```

b. Game time. Implement one-vs-rest for the MNIST data.

```
#create matrix for capturing values from svm function
val_probs = matrix(nrow = 10, ncol = 10)

for (i in 0:9){
  #create copy of training data for manipulation of all digits
  df_train_dig = df_train

  #identify 0 digit for fitting model
  df_train_dig$label = ifelse(df_train_dig$label == i, 1, -1)
  df_train_dig$label = factor(df_train_dig$label)

  #fit training data for model
  fit = svm(label ~ ., data = df_train_dig,

          #radial basis svm turning
          kernel = "radial",

          #cost given for problem
          cost = 100,

          #probability given for problem
          probability = TRUE)

  #perform predictions on test data based on trained data
  pred = predict(fit, df_test, probability = TRUE) %>%
    attr("probabilities")

  #print which digit is being predicted
  print(paste0('Digit Predicted : ', i))

  #print prediction
  print(pred)

  #identify prediction probability for digits 0-9
  val_probs[,i+1]= pred[,2]

}
```

```
#> [1] "Digit Predicted : 0"
#>           -1         1
#> 1  3.461e-05 1.000e+00
#> 2  1.000e+00 1.031e-06
#> 3  1.000e+00 1.633e-05
#> 4  1.000e+00 1.176e-06
#> 5  1.000e+00 1.137e-06
#> 6  9.998e-01 1.666e-04
#> 7  9.988e-01 1.229e-03
```

```
#> 8  9.986e-01 1.443e-03
#> 9  1.000e+00 3.006e-07
#> 10 1.000e+00 1.414e-05
#> [1] "Digit Predicted : 1"
#>           -1         1
#> 1  0.9997313 2.687e-04
#> 2  0.0007446 9.993e-01
#> 3  0.9999800 2.004e-05
#> 4  0.9999198 8.025e-05
#> 5  0.9997402 2.598e-04
#> 6  0.9999761 2.388e-05
#> 7  0.9996653 3.347e-04
#> 8  0.9998763 1.237e-04
#> 9  0.9999916 8.426e-06
#> 10 0.9999990 9.994e-07
#> [1] "Digit Predicted : 2"
#>          -1         1
#> 1  0.999759 2.410e-04
#> 2  0.999424 5.762e-04
#> 3  0.000295 9.997e-01
#> 4  0.999998 1.673e-06
#> 5  0.999994 6.200e-06
#> 6  0.999994 5.843e-06
#> 7  0.999711 2.886e-04
#> 8  0.999989 1.131e-05
#> 9  0.999988 1.201e-05
#> 10 0.999996 4.065e-06
#> [1] "Digit Predicted : 3"
#>           -1         1
#> 1  9.997e-01 2.754e-04
#> 2  9.997e-01 2.775e-04
#> 3  1.000e+00 4.945e-05
#> 4  3.076e-05 1.000e+00
#> 5  9.952e-01 4.801e-03
#> 6  9.907e-01 9.306e-03
#> 7  9.918e-01 8.227e-03
#> 8  9.990e-01 1.038e-03
#> 9  9.998e-01 2.298e-04
#> 10 9.999e-01 1.285e-04
#> [1] "Digit Predicted : 4"
#>           -1         1
#> 1  0.9994080 5.920e-04
#> 2  0.9977849 2.215e-03
#> 3  0.9999999 1.485e-07
#> 4  0.9997595 2.405e-04
#> 5  0.0004927 9.995e-01
#> 6  0.9999786 2.142e-05
#> 7  0.9999596 4.043e-05
#> 8  0.9999986 1.430e-06
#> 9  0.9999477 5.232e-05
#> 10 0.9999134 8.665e-05
#> [1] "Digit Predicted : 5"
#>           -1         1
#> 1  0.9999532 4.680e-05
```

```
#> 2  0.9999991 9.355e-07
#> 3  0.9999979 2.120e-06
#> 4  0.9999657 3.429e-05
#> 5  0.9996431 3.569e-04
#> 6  0.0002324 9.998e-01
#> 7  0.9999998 1.923e-07
#> 8  0.9996900 3.100e-04
#> 9  0.9998001 1.999e-04
#> 10 0.9997163 2.837e-04
#> [1] "Digit Predicted : 6"
#>          -1         1
#> 1  0.999999 9.649e-07
#> 2  0.999985 1.500e-05
#> 3  0.999949 5.104e-05
#> 4  0.999999 5.777e-07
#> 5  0.999715 2.848e-04
#> 6  0.999951 4.891e-05
#> 7  0.009563 9.904e-01
#> 8  0.999999 1.335e-06
#> 9  0.999966 3.371e-05
#> 10 0.999999 7.312e-07
#> [1] "Digit Predicted : 7"
#>            1        -1
#> 1  5.418e-04 0.9994582
#> 2  2.760e-04 0.9997240
#> 3  4.662e-03 0.9953377
#> 4  1.663e-06 0.9999983
#> 5  3.633e-04 0.9996367
#> 6  9.822e-05 0.9999018
#> 7  3.120e-05 0.9999688
#> 8  9.998e-01 0.0001586
#> 9  4.373e-06 0.9999956
#> 10 5.255e-05 0.9999475
#> [1] "Digit Predicted : 8"
#>           -1         1
#> 1  9.989e-01 1.074e-03
#> 2  9.997e-01 3.292e-04
#> 3  1.000e+00 8.774e-06
#> 4  9.993e-01 6.503e-04
#> 5  9.997e-01 3.305e-04
#> 6  9.996e-01 3.579e-04
#> 7  9.984e-01 1.634e-03
#> 8  9.995e-01 4.636e-04
#> 9  2.121e-05 1.000e+00
#> 10 9.999e-01 1.108e-04
#> [1] "Digit Predicted : 9"
#>          -1         1
#> 1  0.999958 4.249e-05
#> 2  0.999661 3.394e-04
#> 3  0.999963 3.671e-05
#> 4  0.999847 1.530e-04
#> 5  0.999035 9.652e-04
#> 6  0.999982 1.803e-05
#> 7  0.999961 3.921e-05
```

```
#> 8  0.999008 9.919e-04
#> 9  0.999311 6.895e-04
#> 10 0.000137 9.999e-01
```