Stat 246 Project

Jerry Chee

Setup

```
setwd("/home/jerry/GoogleDrive/Documents/College/3rd Year/Spring Quarter/Pattern Recognition/")
load("digits.RData")

num.class <- dim(training.data)[1] # Number of classes
num.training <- dim(training.data)[2] # Number of training data per class
d <- prod(dim(training.data)[3:4]) # Dimension of each training image (rowsxcolumns)
num.test <- dim(test.data)[2] # Number of test data
dim(training.data) <- c(num.class * num.training, d) # Reshape training data to 2-dim matrix
dim(test.data) <- c(num.class * num.test, d) # Same for test.
training.label <- rep(0:9, num.training) # Labels of training data.
test.label <- rep(0:9, num.test) # Labels of test data
D = 400 # length of feature vector
```

2.1 LDA

Parameter Estimates from Training Data I first have a function which generates the parameter estimates $\hat{\mu_k}, \hat{\Sigma}$ from a subset of the training data. Note that $\hat{\Sigma}$ is not yet smoothed. We do not need to compute $\hat{\pi_k}$ because we have equal number of training samples from each class, and when we perform cross-validation, we ensure that there is equal number of training samples from each class. Thus $\hat{\pi_k}$ becomes a constant which we do not need to consider.

```
gen_training_param <- function(training.data_subset, training.label_subset) {
  # split training data subset by digit classification
  training_0 = training.data_subset[training.label_subset == 0,]
  training_1 = training.data_subset[training.label_subset == 1,]
  training_2 = training.data_subset[training.label_subset == 2,]
  training_3 = training.data_subset[training.label_subset == 3,]
  training_4 = training.data_subset[training.label_subset == 4,]
  training_5 = training.data_subset[training.label_subset == 5,]
  training_6 = training.data_subset[training.label_subset == 6,]
  training_7 = training.data_subset[training.label_subset == 7,]
  training_8 = training.data_subset[training.label_subset == 8,]
  training_9 = training.data_subset[training.label_subset == 9,]

  # get mean for each digit
  m = dim(training_0)[1]
  mu_est_0    = .colMeans(training_0, m, D)
  mu_est_1    = .colMeans(training_1, m, D)
  mu_est_2    = .colMeans(training_2, m, D)
  mu_est_3    = .colMeans(training_3, m, D)
  mu_est_4    = .colMeans(training_4, m, D)
  mu_est_5    = .colMeans(training_5, m, D)
  mu_est_6    = .colMeans(training_6, m, D)
  mu_est_7    = .colMeans(training_7, m, D)
```

```r
  mu_est_8    = .colMeans(training_8, m, D)
  mu_est_9    = .colMeans(training_9, m, D)

  # generate matrix mu_est, each row a mu_k
  mu_est = matrix( c(mu_est_0, mu_est_1, mu_est_2, mu_est_3,
                     mu_est_4, mu_est_5, mu_est_6, mu_est_7,
                     mu_est_8, mu_est_9),
                   nrow=10, ncol=D, byrow=TRUE)

  # generate matrix cov_est
  N = dim(training.data_subset)[1]
  cov_est = matrix(rep.int(0, D*D), nrow=D, ncol=D)
  for (i in 1:N) {
    x_i = training.data_subset[i,]
    dim(x_i) = c(D,1)
    mu_i = mu_est[training.label_subset[i]+1,]
    dim(mu_i) = c(D,1)
    cov_est = cov_est + (x_i - mu_i) %*% t(x_i - mu_i)
  }
  cov_est = cov_est / N

  # return
  training_param = list("mu_est"=mu_est, "cov_est"=cov_est)
  return(training_param)
}
```

Bayes' Classifier I now have a function which serves as the Bayes' classifier derived in 2.1 LDA a). The $\omega_k s$ and $\omega_{0k} s$ are each kept in a matrix ω and ω_0. $\omega^T x + \omega_0$ results in a vector. Note that the param variable holds $\hat{\mu_k}, \hat{\Sigma_\lambda}$. Any covariance matrix passed as a parameter to bayes_classifier() will have been smoothed. The decision of the Bayes' classifier amounts to returning the (index - 1) of the element with the greatest value.

```r
bayes_classifier <- function(param, x){
  mu_est   = param$mu_est
  cov_inv  = param$cov_inv
  w_t      = mu_est %*% cov_inv # transpose of w
  w_0      = -0.5 * diag(mu_est %*% cov_inv %*% t(mu_est))
  decision = which.max(w_t %*% x + w_0) - 1 # convert 1...10 index to 0...9
  return(decision)
}
```

The eval_classifier() function outputs the count of miss-classifications on a test data set, given parameters $\hat{\mu_k}, \hat{\Sigma_\lambda}$.

```r
eval_classifier <- function(test, labels, param){
  miss_classify = 0
  num_test = dim(test)[1]
  for (i in 1:num_test) {
    if (bayes_classifier(param, test[i,]) != labels[i]) {
      miss_classify = miss_classify + 1
    }
  }
```

```
  return(miss_classify)
}
```

Train Smoothing Parameter To train λ, I compute the miss-classification error of a random subset of 400 training samples per class, testing on the remaining 100 training samples per class. I do this 5 times on range of values of λ. We only need to keep track of the miss-classification counts and not their averages because each training set and test set in the the cross-validation is of the same size.

```
train_lambda <- function(training, labels, ls_lambda) {
  # get training subsets by digit
  training_0 = training[labels == 0,]
  training_1 = training[labels == 1,]
  training_2 = training[labels == 2,]
  training_3 = training[labels == 3,]
  training_4 = training[labels == 4,]
  training_5 = training[labels == 5,]
  training_6 = training[labels == 6,]
  training_7 = training[labels == 7,]
  training_8 = training[labels == 8,]
  training_9 = training[labels == 9,]

  # array to hold lambda missclassification
  n_lambda    = length(ls_lambda)
  miss_lambda = rep(0, n_lambda)
  class_size  = dim(training_0)[1]
  train_size  = class_size * 0.8
  test_size   = class_size - train_size

  # 5x cross validation
  for (i in 1:5){
    # for each class training is random 400 out of 500
    idx = sample(1:class_size, train_size)
    train_0 = training_0[idx,]
    train_1 = training_1[idx,]
    train_2 = training_2[idx,]
    train_3 = training_3[idx,]
    train_4 = training_4[idx,]
    train_5 = training_5[idx,]
    train_6 = training_6[idx,]
    train_7 = training_7[idx,]
    train_8 = training_8[idx,]
    train_9 = training_9[idx,]

    # for each class test is remaining 100
    test_0  = training_0[-idx,]
    test_1  = training_1[-idx,]
    test_2  = training_2[-idx,]
    test_3  = training_3[-idx,]
    test_4  = training_4[-idx,]
    test_5  = training_5[-idx,]
    test_6  = training_6[-idx,]
    test_7  = training_7[-idx,]
    test_8  = training_8[-idx,]
```

```
    test_9  = training_9[-idx,]

    # form matrix subsets and labels
    train_subset = rbind(train_0, train_1, train_2, train_3,
                         train_4, train_5, train_6, train_7,
                         train_8, train_9)
    train_label  = c(rep(0,train_size), rep(1,train_size), rep(2,train_size),
                     rep(3,train_size), rep(4,train_size), rep(5,train_size),
                     rep(6,train_size), rep(7,train_size), rep(8,train_size),
                     rep(9, train_size))
    test_subset  = rbind(test_0, test_1, test_2, test_3,
                        test_4, test_5, test_6, test_7,
                        test_8, test_9)
    test_label   = c(rep(0,test_size), rep(1,test_size), rep(2,test_size),
                     rep(3,test_size), rep(4,test_size), rep(5,test_size),
                     rep(6,test_size), rep(7,test_size), rep(8,test_size),
                     rep(9, test_size))

    # get parameter estimates from training subset
    param = gen_training_param(train_subset, train_label)

    # go over lambda values
    for (l in 1:n_lambda){
      # smooth by lambdaS
      cov_smooth = (1 - ls_lambda[l]) * param$cov_est + (ls_lambda[l]/4) * diag(D)
      cov_inv    = solve(cov_smooth)
      param_smooth = list("mu_est "=param$mu_est, "cov_inv"=cov_inv)

      # missclassification
      miss_lambda[l] = miss_lambda[l] + eval_classifier(test_subset, test_label, param_smooth)
    }
  }

  return(miss_lambda)
}
```

I now run the code to train λ, choosing λ that has the lowest miss-classification count over the 5 cross-validation trials.

```
ls_lambda = seq(0.02, 0.50, 0.02)
misscount_lambda = train_lambda(training.data, training.label, ls_lambda)
```

0.02	0.04	0.06	0.08	0.1	0.12	0.14	0.16	0.18	0.2	0.22	0.24	0.26
763.00	750.00	742.00	738.00	736.00	725.00	722.00	724.00	723.00	724.00	725.00	726.00	723.00

0.28	0.3	0.32	0.34	0.36	0.38	0.4	0.42	0.44	0.46	0.48	0.5
723.00	726.00	729.00	730.00	729.00	727.00	733.00	748.00	747.00	750.00	754.00	756.00

And now we find the λ which minimizes the miss-classification error:

```r
lambda_hat = ls_lambda[which.min(misscount_lambda)]
print(lambda_hat)
```

```
## [1] 0.14
```

Miss-Classification Error on Test Data Set Finally I retrain the estimates $\hat{\mu_k}$ and $\hat{\Sigma_k}$ given $\hat{\lambda}$ on the entire training data set, and compute the miss-classification error on the test data set.

```r
param_hat = gen_training_param(training.data, training.label)
#smooth covariance
param_hat$cov_est = (1 - lambda_hat) * param_hat$cov_est + (lambda_hat/4) * diag(D)
# add matrix inverse to param
param_hat$cov_inv = solve(param_hat$cov_est)

#miss-classification count
num_test = dim(test.data)[1]
error_count = eval_classifier(test.data, test.label, param_hat)
print(error_count / num_test)
```

```
## [1] 0.1436
```

2.2 Bernoulli Mixture

E-Step I compute $\gamma(z_{im})$ by the method described in 2.2 c) to prevent issues of scaling.

```r
log_mixture_component <- function(x_i, pi_m, mu_m) {
  tmp = 0.0
  for (j in 1:D) {
    tmp = tmp + x_i[j] * log(mu_m[j]) + (1 - x_i[j]) * log(1 - mu_m[j])
  }
  return(log(pi_m) + tmp)
}

gamma_im <- function(x_i, pi, mu, m) {
  M = length(pi)
  ls_gamma = rep(0, M)
  for (alpha in 1:M) {
    ls_gamma[alpha] = log_mixture_component(x_i, pi[alpha], mu[alpha,])
  }
  component_max = which.max(ls_gamma)

  denom = 0.0
  for (alpha in 1:M) {
    denom = denom + exp(ls_gamma[alpha] - component_max)
  }
  return(exp(ls_gamma[m] - component_max) / denom)
}
```

This function computes all $\gamma(z_{im})$ and stores them in a matrix, each row for a different sample x_i.

```r
gamma_all <- function(X, pi, mu) {
  #n x M matrix
  n = dim(X)[1]
  M = length(pi)
  gamma = matrix(rep(0, n*M), nrow=n, ncol=M)

  for (i in 1:n) {
    for (m in 1:M) {
      gamma[i,m] = gamma_im(X[i,], pi, mu, m)
    }
  }
  return(gamma)
}
```

M-Step Here are functions which compute $\hat{pi_m}$ and $m\hat{u}_{mj}$ in terms of $\gamma(z_{im})$. the pi_all() and mu_all() functions compute all $\hat{pi_m}$ and $\hat{\mu_{mj}}$ for a given θ^{old}, storing them in a matrix.

```r
pi_m_hat <- function(X, gamma_m) {
  n = dim(X)[1]
  M = length(pi)
  numerator = sum(gamma_m) + 1
  return(numerator / (n+M))
}
mu_mj_hat <- function(X, gamma_m, j) {
  n = dim(X)[1]
  numerator = 1.0
  denom = 0
  # sum to get numerator
  for (i in 1:n) {
    numerator = numerator + gamma_m[i] * X[i,j]
  }
  # sum to get denominator
  denom = sum(gamma_m) + 2
  return(numerator / denom)
}
pi_all <- function(X, gamma) {
  M = dim(gamma)[2]
  pi = rep(0, M)
  for (m in 1:M) {
    pi[m] = pi_m_hat(X, gamma[,m])
  }
  return(pi)
}
mu_all <- function(X, gamma) {
  M = dim(gamma)[2]
  mu = matrix(rep(0, M*D), nrow=M, ncol=D)
  for (m in 1:M) {
    for (j in 1:D) {
      mu[m,j] = mu_mj_hat(X, gamma[,m], j)
    }
  }
  return(mu)
}
```

Evaluation of Q() + lnp(theta) This function evaluates the value of $Q(\theta, \theta^{old}) + \ln(p(\theta))$

```
Q_theta_eval <- function(X, pi_new, pi_old, mu_new, mu_old) {
  n = dim(X)[1]
  M = length(pi_new)
  gamma_old = gamma_all(X, pi_old, mu_old)
  val = 0.0
  # Q(theta, theta_old)
  for (i in 1:n) {
    for (m in 1:M) {
      tmp = 0
      for (j in 1:D) {
        tmp = tmp + X[i,j]*log(mu_new[m,j])+ (1-X[i,j])*log(1-mu_new[m,j])
      }
      val = val + gamma_old[i,m] * (log(pi_new[m]) + tmp)
    }
  }
  # ln(p(theta))
  for (m in 1:M) {
    tmp = 0
    for (j in 1:D) {
      tmp = tmp + log(mu_new[m,j]) + log(1-mu_new[m,j])
    }
    val = val + log(pi_new[m]) + tmp
  }
  return(val)
}
```

EM Algorithm I initialize the EM algorithm by randomly assinging values to the latent variables z_{im}. For each z_i, chose a single $m \in 1 \ldots M$ such that $z_{im} = 1$, all other $z_{i\alpha} = 0$ such that $\alpha ! = m$. I store these latent variables in a matrix Z, each row per sample x_i. I then compute the vector π holding all π_m values and the matrix μ holding all μ_{mj} values, using the latent variables found in Z. z_{im} serves as a replacement for $\gamma_(z_{im})$ when calculating these initial π, μ.

```
init_EM <- function(X, M) {
  n = dim(X)[1]
  # Z
  Z = rep(0, n*M)
  dim(Z) = c(n, M)
  for (i in 1:n) {
    j = sample(1:M, 1)
    Z[i,j] = 1
  }
  # init pi
  pi = pi_all(X, Z)
  # init mu
  mu = mu_all(X, Z)
  out = list("pi"=pi, "mu"=mu)
  return(out)
}
```

EM algorithm runs for a specified number of iterations. At each iteration the error is printed, which I write as the percent difference of of the new optimal $Q(\theta, \theta^{old}) + \ln(p(\theta))$ and the previous optimal $Q(\theta', \theta'^{old}) + \ln(p(\theta'))$. I also print the new value of the optimal $Q(\theta, \theta^{old}) + \ln(p(\theta))$.

```
EM <- function(X, M, iter) {
  init = init_EM(X,M)
  pi_old = init$pi
  mu_old = init$mu
  Q_old = 0
  count = 0

  while (count < iter) {
    count = count + 1
    # E-step
    gamma_new = gamma_all(X, pi_old, mu_old)
    # M-step
    pi_new = pi_all(X, gamma_new)
    mu_new = mu_all(X, gamma_new)
    # Eval Q+lnp(theta)
    Q_new = Q_theta_eval(X, pi_new, pi_old, mu_new, mu_old)
    error = abs((Q_new - Q_old) / mean(c(Q_new, Q_old)))
    cat("error: ", error, " ,Q_new: ", Q_new, "\n")
    # switch variables
    pi_old = pi_new
    mu_old = mu_new
    Q_old = Q_new
  }
  out = list("pi"=pi_new, "mu"=mu_new)
  return(out)
}
```

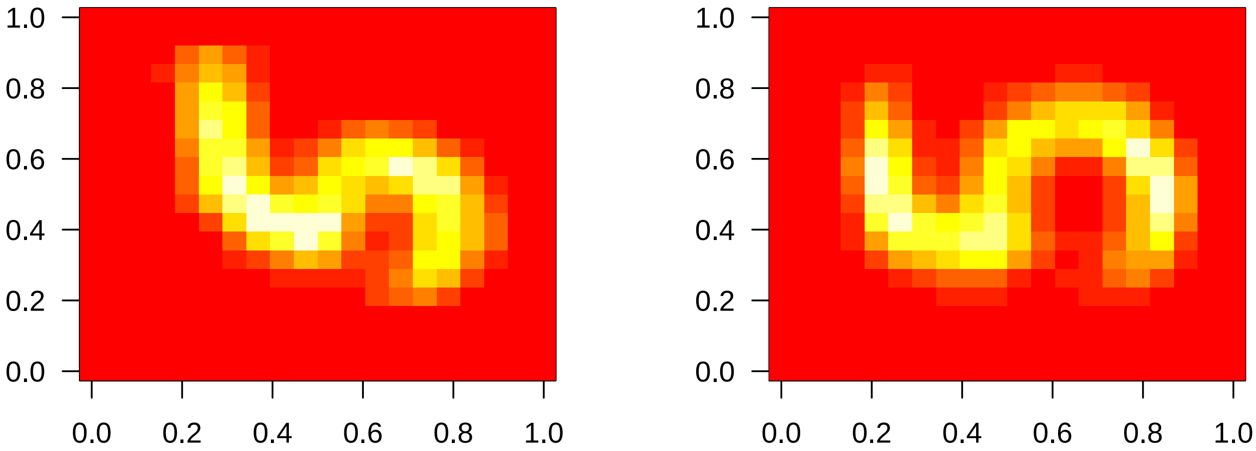
2.d) I look at the digit class for 5.

```
image_class <- function(train, label, class, M) {
  train_class = train[label == class,]
  iter = 5
  param = EM(train_class, M, iter)
  for (m in 1:M) {
    mu_m = param$mu[m,]
    dim(mu_m) = c(20,20)
    image(mu_m)
  }
}
```

For M = 2

```
par(mfrow=c(1,2), las=1)
image_class(training.data, training.label, 5, 2)
```

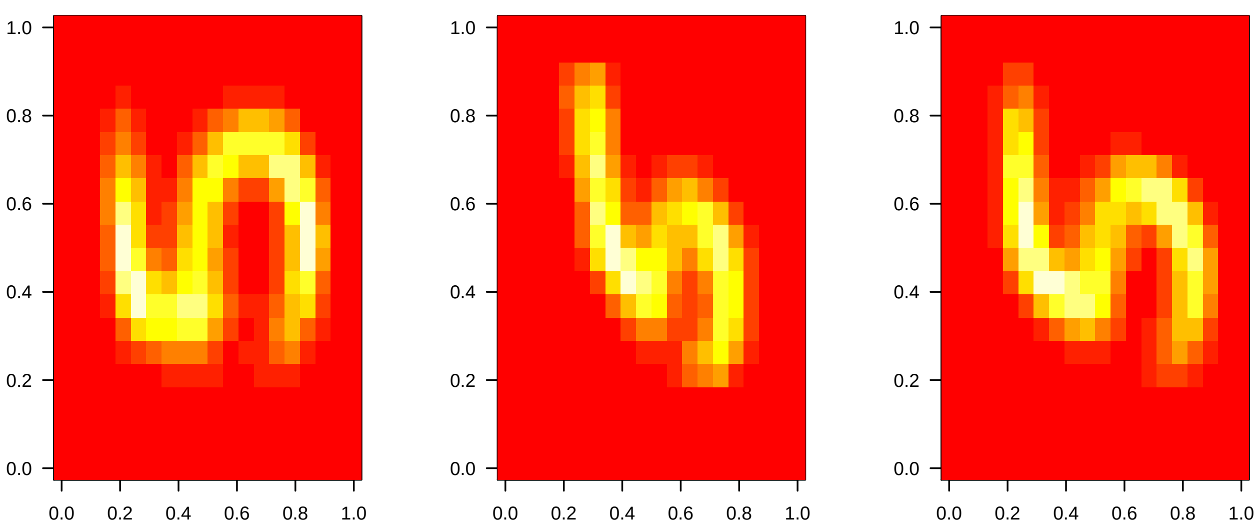
```
## error:  2  ,Q_new:  -48556.16
## error:  0.03483709  ,Q_new:  -46893.57
## error:  0.001660963  ,Q_new:  -46815.74
## error:  0.0004728888  ,Q_new:  -46793.61
## error:  0.000590659  ,Q_new:  -46765.98
```



M = 3

```r
par(mfrow=c(1,3), las=1)
image_class(training.data, training.label, 5, 3)
```

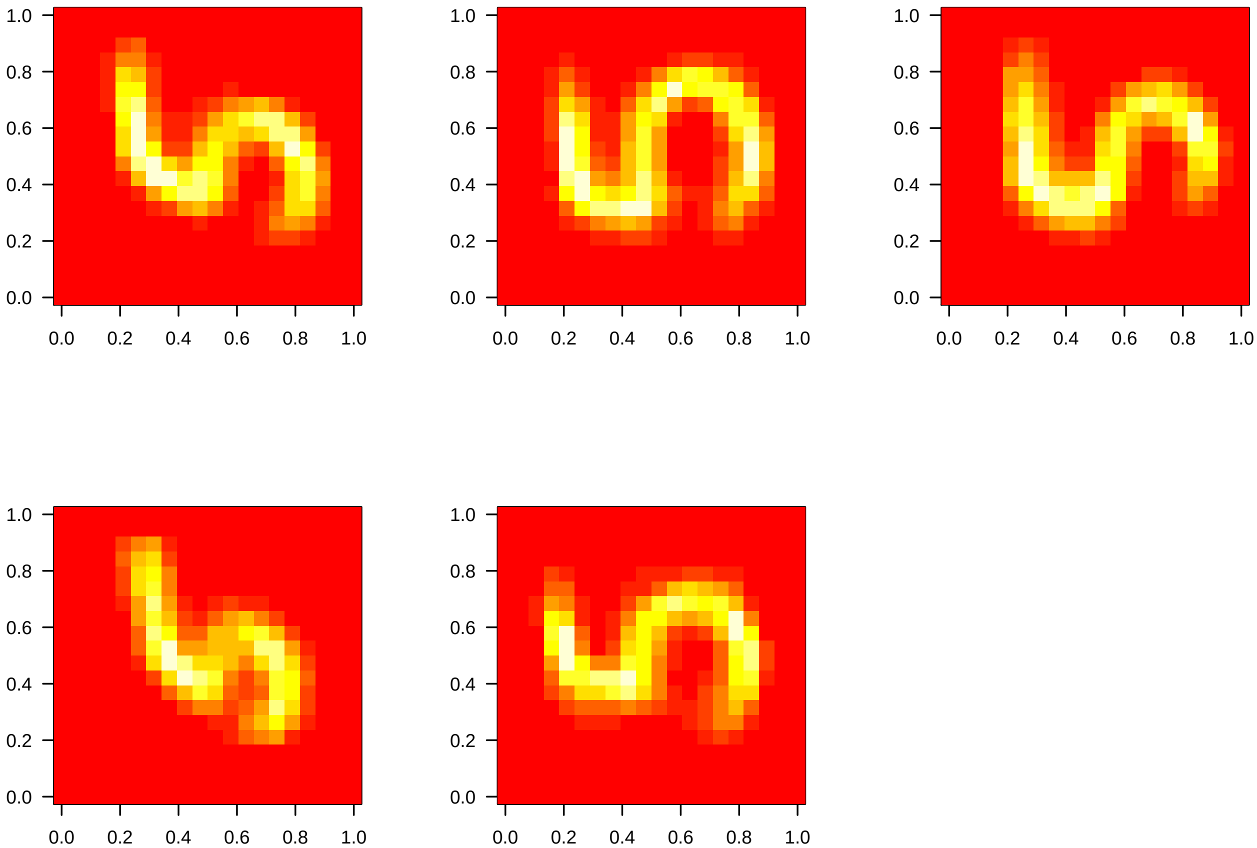
```
## error:  2  ,Q_new:  -49186.81
## error:  0.06062398  ,Q_new:  -46292.63
## error:  0.004382016  ,Q_new:  -46090.22
## error:  0.001436385  ,Q_new:  -46024.07
## error:  0.0006176509  ,Q_new:  -45995.65
```



M = 5

```r
par(mfrow=c(2,3), las=1)
image_class(training.data, training.label, 5, 5)
```

```
## error:  2  ,Q_new:  -51391.02
## error:  0.07105088  ,Q_new:  -47864.91
## error:  0.01220968  ,Q_new:  -47284.04
## error:  0.003945606  ,Q_new:  -47097.85
## error:  0.001236615  ,Q_new:  -47039.64
```



We see that each mixture componenet looks to embody a different handwriting style or way in which the digit can be written.

2.e) Classification I take M = 3, and fit a mixture model for digit classes 0 and 9. To classify, I use the Bayes' Classifier.

$$h(x) = \text{argmax}_k\{p(C_k|x)\} = \text{argmax}_k\{p(x, C_k)\}$$

$$h(x) = \text{argmax}_k\{\sum_{m=1}^{M} \pi_m p(x|\mu_m) \prod_{j=1}^{D} \mu_m^{x_j}(1-\mu_m)^{1-x_j}\}$$

```
eval_mixture <- function(x_i, pi, mu) {
  M = length(pi)
  out = 0
  for (m in 1:M) {
    likelihood = 1
    for (j in 1:D) {
      likelihood = likelihood * mu[m,j]^x_i[j] * (1-mu[m,j])^(1-x_i[j])
    }
    out = out + pi[m] * likelihood
  }
  return(out)
}

classify <- function(test_i, pi_a, pi_b, mu_a, mu_b, class_a, class_b) {
  p_a = eval_mixture(test_i, pi_a, mu_a)
```

```
  p_b = eval_mixture(test_i, pi_b, mu_b)
  if (p_a > p_b) {
    return(class_a)
  } else {
    return(class_b)
  }
}
```

The eval_classifier() function outputs the miss-classification counts, given class a and b of interest.

```
eval_classifier <- function(test, label, pi_a, pi_b, mu_a, mu_b, class_a, class_b) {
  n = dim(test)[1]
  miss = 0
  for (i in 1:n) {
    if (classify(test[i,], pi_a, pi_b, mu_a, mu_b, class_a, class_b) != label[i]) {
      miss = miss + 1
    }
  }
  return(miss)
}
```

I now compute the miss-classification error of digits 0 and 9 on the test data set.

```
# get training subsets for each class
train_0 = training.data[training.label  == 0,]
train_9 = training.data[training.label  == 9,]

# train parameters with EM
M = 3
iter = 5
param_0 = EM(train_0, M, iter)
```

```
## error:  2  ,Q_new:  -51441.99
## error:  0.06108347  ,Q_new:  -48392.86
## error:  0.01279665  ,Q_new:  -47777.53
## error:  0.0028084  ,Q_new:  -47643.54
## error:  0.0007887449  ,Q_new:  -47605.98
```

```
param_9 = EM(train_9, M, iter)
```

```
## error:  2  ,Q_new:  -43237.31
## error:  0.0279537  ,Q_new:  -42045.33
## error:  0.01381113  ,Q_new:  -41468.62
## error:  0.01968546  ,Q_new:  -40660.25
## error:  0.007815094  ,Q_new:  -40343.72
```

```
# get test and label subsets
test_0   = test.data[test.label == 0,]
test_9   = test.data[test.label == 9,]
test_09  = rbind(test_0, test_9)
num_0    = dim(test_0)[1]
```

```r
num_9    = dim(test_9)[1]
label_09 = c(rep(0,num_0), rep(9,num_9))

# compute miss-classification error
miss_count = eval_classifier(test_09, label_09, param_0$pi, param_9$pi, param_0$mu, param_9$mu, 0, 9)
print(miss_count / (num_0+num_9))
```

```
## [1] 0.015
```