DATA630013 Final Project

Jichen Yang

18210980017

Problem 1

(a)

Suppose that $Y \in \mathcal{Y} = \{1, \dots, K\}$. The optimal rule is

$$\begin{aligned} h(x) &= argmax_k \ P(Y = k|X = x) \\ &= argmax_k \ \frac{f_k(x)\pi_k}{\sum_r f_r(x)\pi_r} \\ &= argmax_k \ f_k(x)\pi_k \end{aligned}$$

Consider the LDA model: the observations $(X_i, y_i) \in R^p \times \{1, \dots, K\}$ are i.i.d. sample from a Gaussian mixture model, i.e., $P(y_i = k) = \pi_k$ and $X_i|(y_i = k) \sim N(\mu_k, \Sigma)$. Thus,

$$\begin{aligned} h(x) &= argmax_k \ f_k(x)\pi_k \\ &= argmax_k \ log(f_k(x)) + log\pi_k \\ &= argmax_k \ log\pi_k - 0.5(x - \mu_k)^T\Sigma^{-1}((x - \mu_k)) \\ &= argmax_k \ x^T\Sigma^{-1}\mu_k + log\pi_k - 0.5\mu_k^T\Sigma^{-1}\mu_k \end{aligned}$$

If we use $\beta_k = (\Sigma^{-1}\mu_k)$ and $\beta_{0k} = log\pi_k - 0.5\mu_k^T\Sigma^{-1}\mu_k$, we can get following form,

$$h(x) = argmax_k \ x^T\beta_k + \beta_{0k}$$

(b)

This is exactly the same as problem2 in assignment1. So here I simply repeat the proof process, omiting some details of the derivative.

$$f(x, y = k) = f(x|G = k)f(G = k) = \frac{\pi_k}{\sqrt{(2\pi)^p|\Sigma|}}exp(-0.5(x - \mu_k)^T\Sigma^{-1}(x - \mu_k))$$

assume $y_1 = k_1, \dots, y_n = k_n$

$$l = logL = \sum_{i=1}^{n} log(\pi_{ki}) - (\frac{pn}{2}log(2\pi) + \frac{n}{2}log|\Sigma|) + \sum_{i=1}^{n}(-\frac{1}{2}(x_i - \mu_{ki})^T\Sigma^{-1}(x_i - \mu_{ki}))$$

First, we calculate the $\hat{\mu}_k$:

$$\frac{\partial l}{\partial \mu_k} = \sum_{i=1}^{n} I_{\{k_i=k\}}\Sigma^{-1}(x_i - \mu_{ki}) = 0$$

because $\frac{\partial^2 l}{\partial \mu_k^2} \leq 0$, we can get the MLE of μ_k,

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} I_{\{k_i=k\}} x_i}{\sum_{i=1}^{N} I_{\{k_i=k\}}}$$

Use the similar method, we can get the MLE of Σ

$$\hat{\Sigma} = \frac{1}{n} \sum_{k=1}^{K} \sum_{\{i:y_i=k\}} (x_i - \mu_k)(x_i - \mu_k)^T$$

(c)

In this problem I will show you how to make $\hat{\Sigma}$ invertible.

$$\hat{\Sigma} = \frac{1}{n} \sum_{k=1}^{K} \sum_{\{i:y_i=k\}} (x_i - \mu_k)(x_i - \mu_k)^T = \frac{n_1 \hat{\Sigma}_1 + n_2 \hat{\Sigma}_2 + .. + n_K \hat{\Sigma}_K}{n}$$

each $\hat{\Sigma}_i$ is positive semidefinire matrix, thus $\hat{\Sigma}$ is positive semidefinire matrix, we can get

$$\nu^T \hat{\Sigma}_\lambda \nu = \nu^T ((1-\lambda)\hat{\Sigma} + \lambda/4)\nu = (1-\lambda)\nu^T \Sigma \nu + \lambda/4 * \nu^T \nu > 0$$

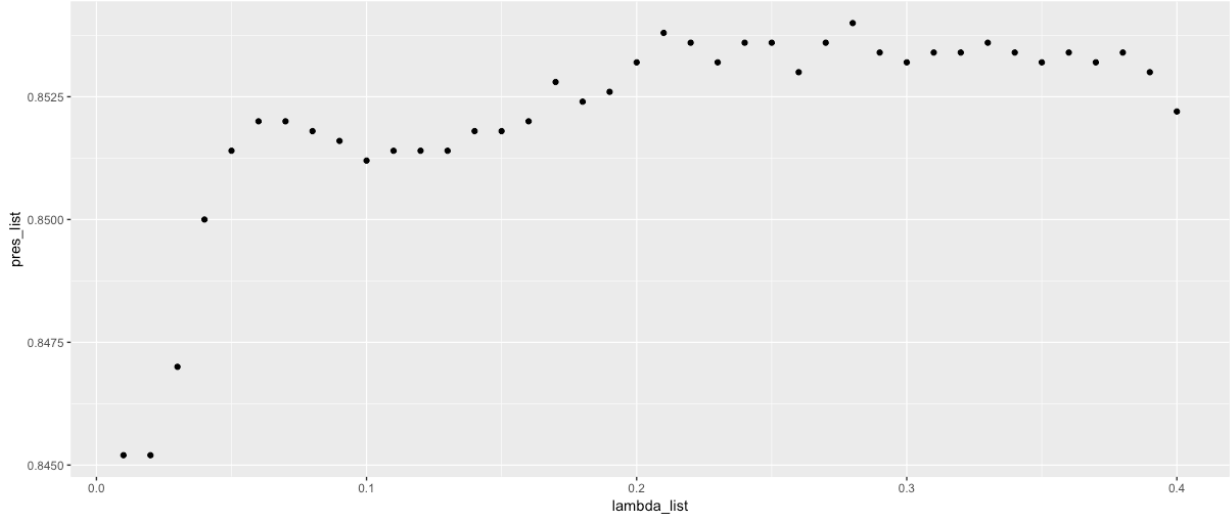
this means $\hat{\Sigma}_\lambda$ is positive definite matrix, each singular value is greater than 0, so it is invertible.

(d)

In this problem I use LDA as a classification method. I combine the **crossvalidation** and **grid search** to find the best smoothing parameter λ.

For each λ, I calculate the error rate by 5-fold-crossvalidation: run the algorithm 5 times on a random subset of 400 training examples per class, and evaluate the error on the remaining 100 examples per class.

Grid search on $\lambda = 0.01, 0.02, \ldots, 0.40$, I get the following result. (Y-label is accuracy rate.)



So I finally choose $\lambda = 0.28$, and then I check the algorithm on the test set, here I only show the crucial code for fitting the method on the training set.

```
# Find best lambda
final_lambda = lambda_list[which.max(pres_list)]
# learning parameter
mu  = matrix(0,10,400)
for (i in 0:9){
  idx = training_label==i
  mu[i+1,] = apply(training.data[idx,],2,mean)
}
#sigma
sigma = matrix(0,400,400)
for (i in 0:9){
  idx = training_label==i
  sigma = sigma + (dim(training.data)[1]/10)*var(training.data[idx,])
}
sigma = sigma/dim(training.data)[1]
sigma = (1-final_lambda)*sigma + final_lambda/4*diag(400)
#pi_k
prior = as.numeric(table(training_label))/length(training_label)
beta = solve(sigma) %*% t(mu)

beta0=c()
for (i in 1:10){
  beta0k = log(prior[i])-0.5* mu[i,] %*% solve(sigma) %*% mu[i,]
  beta0 = append(beta0, beta0k)
}
# Here for predict and calculate accuracy rate
pred_fun = function(x){
  pred = rep(0,10)
  for (i in 1:10){
    pred[i] = as.numeric(x) %*% beta[,i] + beta0[i]
  }
  final = which.max(pred)-1
  return(final)
}

acc = function(x_pred,x_label){
  ac = sum(as.numeric(x_pred==x_label))/length(x_pred)
  return(ac)
}
pred_test = apply(test.data,1,pred_fun)
acc_test = acc(pred_test,test_label)
```

The parameter for this model is $\beta \in R^{10\times 400}$ and $\beta_0 \in R^{10\times 1}$, it's too large to show here, you can check it by using my code, it runs in a few seconds.

I get the accuracy rate is **0.8534**.

Problem 2

(a)

First let's derive the formula for $Q(\theta, \theta^{old})$,

$$P(X, Z|\theta) = \prod_{i=1}^{n}[\prod_{m=1}^{M}(\pi_m \prod_{j=1}^{D} {\mu_{mj}}^{x_{ij}}(1-\mu_{mj})^{1-x_{ij}})^{1_{\{Z=m\}}}]$$

Then,

$$logP(X, Z|\theta) = \sum_{i=1}^{n}\{1_{\{z=m\}}[\sum_{m=1}^{M}(log\pi_m + logf_m(x_i))]\}$$

where $f_m(x_t) = \prod_{j=1}^{D} \mu_{mj}^{x_{ij}}(1-\mu_{mj})^{1-x_{ij}}$,

Thus,

$$\begin{aligned} Q(\theta, \theta^{old}) &= E_{Z|X}(logP(X, Z, \theta)) \\ &= \sum_{i=1}^{n}\sum_{m=1}^{M}\gamma(Z_{im})(log\pi_m + logf_m(x_i)) \\ &= \sum_{i=1}^{n}\sum_{m=1}^{M}[\gamma(Z_{im})(log\pi_m + \sum_{j=1}^{D}x_{ij}log\mu_{mj} + (1-x_{ij})log(1-\mu_{mj}))] \end{aligned}$$

Where,

$$\begin{aligned} \gamma(Z_{im}) &= P(Z_i = m|X_i) \\ &= \frac{\pi_m^{old} f_m^{old}(x_i)}{\sum_{m=1}^{M}\pi_m^{old} f_m^{old}(x_i)} \end{aligned}$$

next we derive the formula for $P(\theta)$, because of the **conjucate prior**, we can derive the posterior easily,

$$log\, p(\theta) = \sum_{m=1}^{M} log\pi_m + \sum_{m=1}^{M}\sum_{j=1}^{D}[log(\mu_{mj}) + log(1-\mu_{mj})] + C$$

For convenience, $K = Q(\theta, \theta^{old}) + log\, p(\theta)$,

$$\frac{\partial K}{\partial \mu_{mj}} = \sum_{i=1}^{n}\gamma(Z_{im})(\frac{x_{ij}}{\mu_{mj}} + \frac{x_{ij}-1}{1-\mu_{mj}}) + \frac{1}{\mu_{mj}} - \frac{1}{1-\mu_{mj}}$$

so we can get the MAP of μ is

$$\hat{\mu}_{mj} = \frac{1 + \sum_{i=1}^{n}\gamma(Z_{im})x_{ij}}{2 + \sum_{i=1}^{n}\gamma(Z_{im})}$$

MAP of π_m is more difficult, we need use the KKT conditions to solve an optimization problem,

$$max\ log\pi_m \sum_{i=1}^{n}\gamma(Z_{im}) + log\pi_m$$

$$s.t. \sum_{i=1}^{M} \pi_m = 1$$

Thus, the Lagrangian function is

$$log\pi_m \sum_{i=1}^{n} \gamma(Z_{im}) + log\pi_m + \lambda(1 - \sum_{i=1}^{n} \pi_m)$$

combine with the constrain $\sum \pi_m$=1, we can get the MAP of π_m

$$\hat{\pi}_m = \frac{1 + \sum_{i=1}^{n} \gamma(Z_{im})}{M + \sum_{i=1}^{n} \sum_{m=1}^{M} \gamma(Z_{im})}$$

Finally, we get the EM algorithm of this problem:

1. E-step: $\gamma(Z_{im}) = \frac{\pi_m^{old} f_m^{old}(x_i)}{\sum_{m=1}^{M} \pi_m^{old} f_m^{old}(x_i)}$
2. M-step: $\hat{\mu}_{mj} = \frac{1+\sum_{i=1}^{n} \gamma(Z_{im}) x_{ij}}{2+\sum_{i=1}^{n} \gamma(Z_{im})}$, $\hat{\pi}_m = \frac{1+\sum_{i=1}^{n} \gamma(Z_{im})}{M+\sum_{i=1}^{n} \sum_{m=1}^{M} \gamma(Z_{im})}$

(b)

1. Assign each example at random to one of the M components

```
train_mix_component = sample(1:M, size = dim(train_3)[1],replace = TRUE)
```

2. Use the mix_component initialize γ with 0 and 1. If the i-th sample is belong to m-th component, then $\gamma_{im} = 1$, others in this row equal to 0.

```
Gamma = matrix(0, dim(train_3)[1], M)
for (i in 1:dim(train_3)[1]){
  idx = train_mix_component[i]
  Gamma[i,idx] = 1
}
```

3. M-step: use γ_{im} to initialize μ_{mj} and π_m.

```
#mu M*D
mu = matrix(0, M, D)
for (m in 1:M){
  for(j in 1:D){
    mu[m,j] = (1+Gamma[, m] %*%train_3[, j]) / (2+sum(Gamma[,m]))
  }
}
#pi_prior
pi_prior = rep(0,M)
for (m in 1:M){
  pi_prior[m] = (1+sum(Gamma[, m])) / (M+sum(Gamma))
}
```

(c)

First of all, I want to prove that the two methods are equivalent.

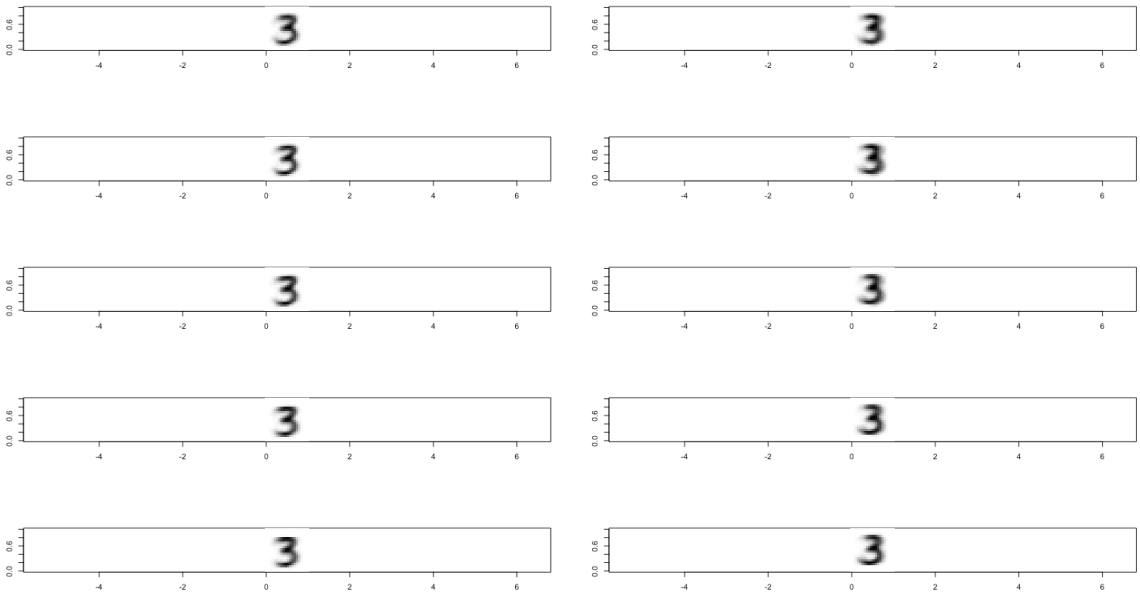
$$\begin{aligned}\gamma(z_{im}) &= \frac{\frac{exp(l_m)}{exp(l^*)}}{\sum_{j=1}^{M}\frac{exp(l_j)}{exp(l^*)}} \\ &= \frac{exp(l_m)}{\sum_{j=1}^{M} exp(l_j)} \\ &= \frac{\pi_m f_m(x_i)}{\sum_{j=1}^{M}\pi_j f_j(x_i)}\end{aligned}$$

where $f_m(x_i) = \prod_{j=1}^{D}\mu_{mj}^{x_{ij}}(1-\mu_{mj})^{1-x_{ij}}$. This is mainly to avoid loss of precision, because too small numbers will be recorded as 0 in R, $f_m(x_i)$ is very close to 0, so we need to use $log(f_m)$. Logarithm can avoid this phenomenon without loss of precision.

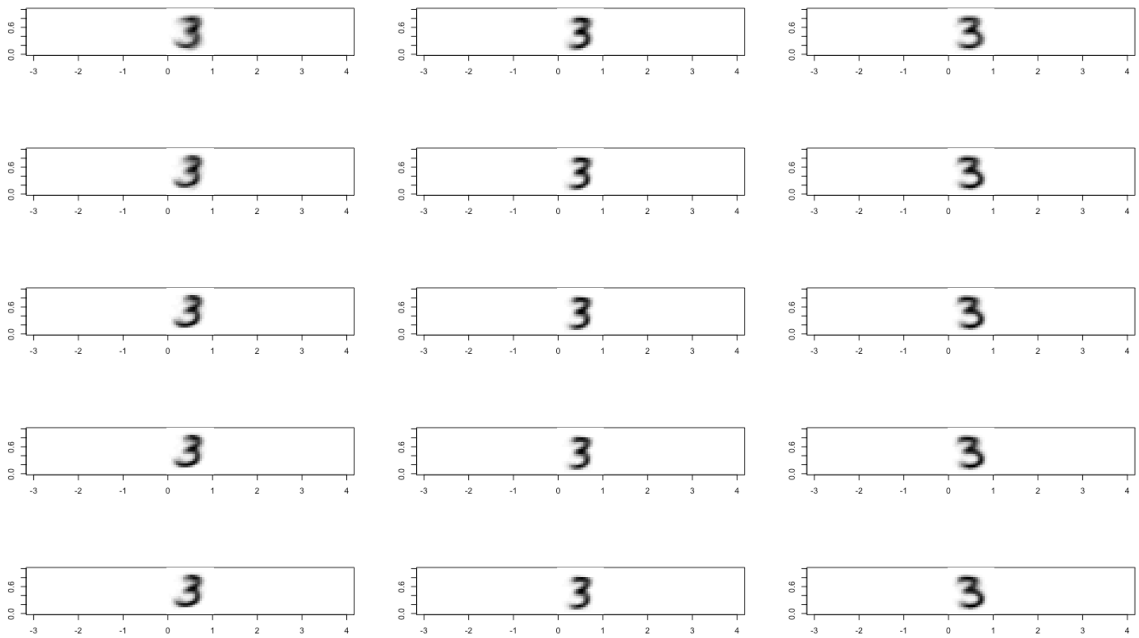
(d)

I choose digit-3 for algorithms in this problem. I run the EM-algorithms for **5 times** and plot the image for each of the M components in every iteration. Each row for each iteration.

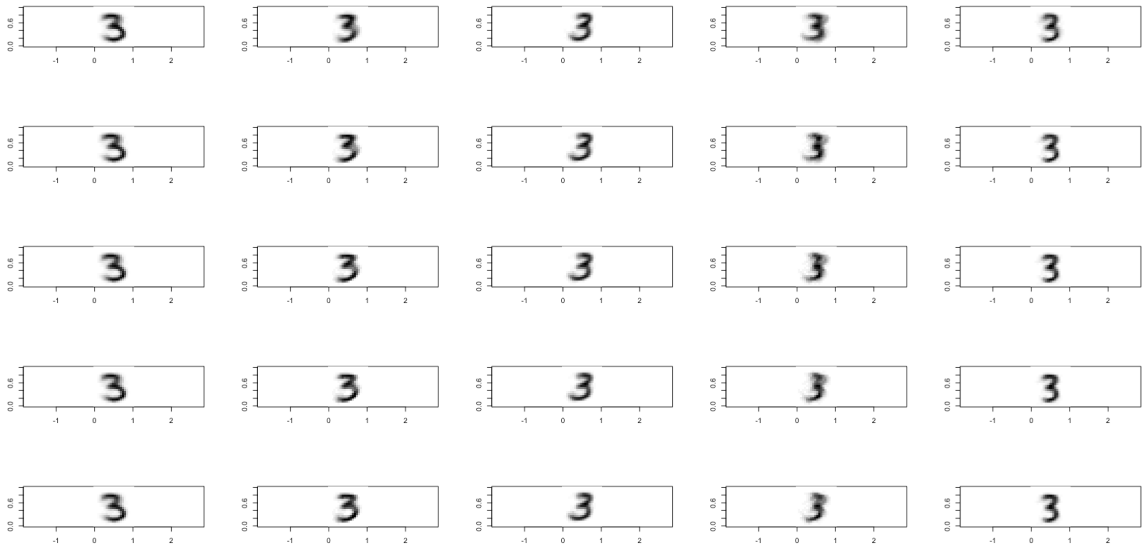
1. M = 2



2. M = 3

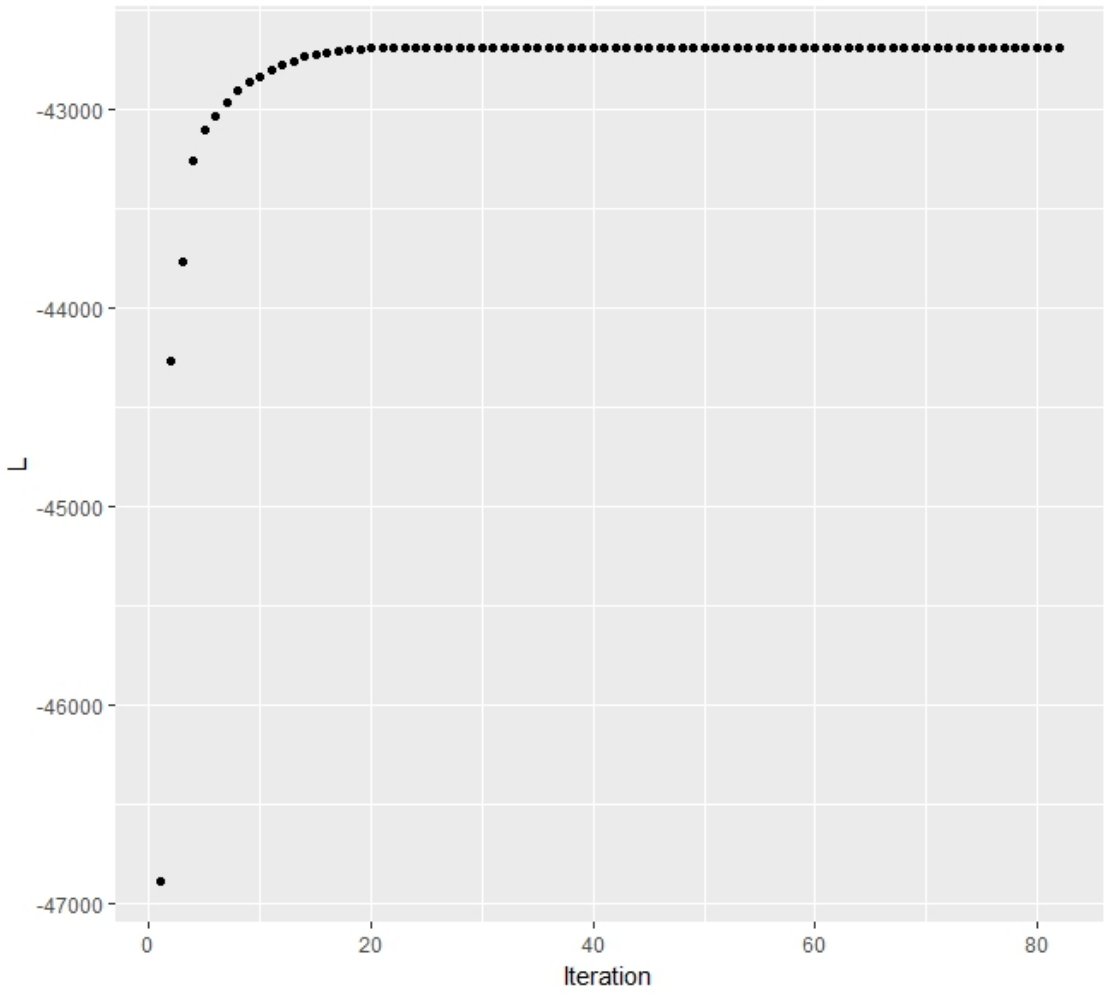


3. M = 5



We can see that the algorithm converges very fast, the first iteration has a good performance, and then each iteration makes the shape from fuzzy to concrete. As you can see, there are two types of patterns that are very similar when M=5, so I will choose M=3 when I fit the model in **problem (e)**.

The project says that show the value of the log-likelihood at each iteration. I think this is not very important in the EM algorithm, so only the case of M=3 is drawn here. It converge after 82 iterations.



(e)

In this problem, we need choose the training data of two digit classes and fit a mixture model for each of them. I choose the digit 1 and 8 to build two models separately, I choose the M = 3 heuristically.

Run the EM-algorithm for **100 times**, then we can get $\mu^{[1]}, \pi^{[1]}$ for model-1 and $\mu^{[2]}, \pi^{[2]}$ for model-2. For test sample we need to calculate the probability
$P^{[1]}(X_i) = \sum_{m=1}^{M} \pi_m^{[1]} (\prod_{j=1}^{D} \mu_{mj}^{[1]x_{ij}} (1-\mu_{mj}^{[1]})^{1-x_{ij}})$ and
$P^{[2]}(X_i) = \sum_{m=1}^{M} \pi_m^{[2]} (\prod_{j=1}^{D} \mu_{mj}^{[2]x_{ij}} (1-\mu_{mj}^{[2]})^{1-x_{ij}})$. If $P^{[1]} \geq P^{[2]}$, we predict this sample as digit 1. If $P^{[1]} < P^{[2]}$, we predict this sample as digit 8.

Here I only show the crucial code.

```
#I wrote a function myself, too long, so I omitted the details of this
function here.
em_alg = function(M = 2, digit = 8, step = 2){...}
#fit on training set
digit = c(1,8)
parameter1 = em_alg(M=3,digit[1],step = 100)
parameter2 = em_alg(M=3,digit[2],step = 100)
#test set
```

```
test1 = test.data[test_label==1,]
test2 = test.data[test_label==8,]
test = rbind(test1,test2)
testlabel = c(test_label[test_label==1],test_label[test_label==8])
#predict on test set
predd = matrix(0, 2, dim(test)[1])
for (iter in 1:dim(test)[1]){
  p1 = pred(test[iter,],parameter1$mu,parameter1$pi_prior,3)
  p2 = pred(test[iter,],parameter2$mu,parameter2$pi_prior,3)
  predd[,iter] = c(p1,p2)
}
compare_idx = predd[1,]-predd[2,]>=0

pred_value = rep(0,dim(test)[1])
pred_value[compare_idx] = digit[1]
pred_value[!compare_idx] = digit[2]

#Test accuracy
acc = function(x_pred,x_label){
  ac = sum(as.numeric(x_pred==x_label))/length(x_pred)
  return(ac)
}
presision= acc(pred_value,testlabel)
```

Then we get the accuracy on test set of 2 selected digit classes is **0.984**.

However, please note that this is only the accuracy of the two classifications and is not comparable to the results of the previous question.

Problem 3

Finally, I tested several other classification algorithms. I didn't spend too much time tuning the parameters, just picking some of the better models I tested.

(因为已经完成了project中的内容，所以这一部分仅仅是测试下其他模型性能，调用了一些现成的R-package)

1. Softmax Regression(logistic regression for Multi-classification)

```
library(softmaxreg)
model  = softmaxReg(training.data, training_label, hidden = c(), funName =
'sigmoid', maxit =100 ,rang = 0.1, type = "class", algorithm = "adagrad",
rate = 0.01, batch = 100)

pred = predict(model,test.data )
acc_test = acc(pred-1,test_label)
acc_test
```

This model is very similar with 1-layer neural network. There are a lot of hyperparameters and it is necessary to use random gradient descent during training. Training process is very slow, maybe 5 minutes for following code. Accuracy rate is **0.8817**.

2. XGBoost model

```
library(xgboost)
#train
model <- xgboost(data = training.data, label= training_label,nrounds = 100,
nthread = 64, nfold = 5, max_depth = 10, eta = 0.1,objective =
"multi:softmax",num_class = 10)
#test
preds = predict(model,test.data)
acc_test = acc(preds,test_label)
acc_test
```

It takes only 2 minutes on my mac, and the accuracy rate is pretty high, **0.9217**. This model is very popular in data science competitions in recent years, and the accuracy is very high. If the parameters are carefully adjusted, the accuracy rate can be higher.

3. KNN model

```
library(class)
pred<-knn(train=training.data,test=test.data,cl=training_label,k=10)
acc_test = acc(pred,test_label)
acc_test
```

Accuracy rate is **0.8957**.

Finally, I summarize the accuracy of these methods.

LDA	Softmax Regression	XGBoost	KNN
0.8534	0.8817	0.9217	0.8957