Introduction to Machine Learning 732A95

Lab 1 Block 2

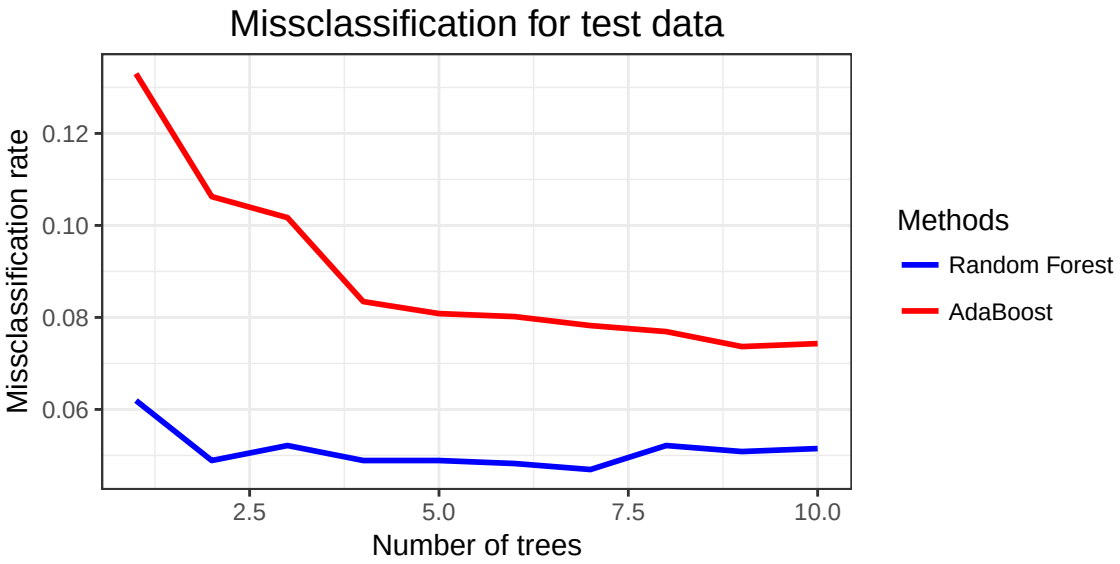
Eric Herwin
erihe068

Contents

1. Ensemble Methods

For this task I will use AdaBoost classification trees and random forests to classify spam mails. I will evaluate both of the models with the missclassification rate for the test data. I have splitted up the data as 2/3 are training and 1/3 are test. The code used for this is provided in the appendix.

```
n <- dim(spambase)[1]
set.seed(12345)
id <- sample(1:n, floor(n*2/3))
train <- spambase[id,]
test <- spambase[-id,]
```



We can see that the more trees I am using, the lesser the missclassification rate is. From this plot the conclusion is that random forest performs better for this data set then AdaBoost and this might be because the data are more appropriate for a random forest.

Generally the AdaBoost would perform better, due to the penalization to improve and the random forest just grows trees at random to improve.

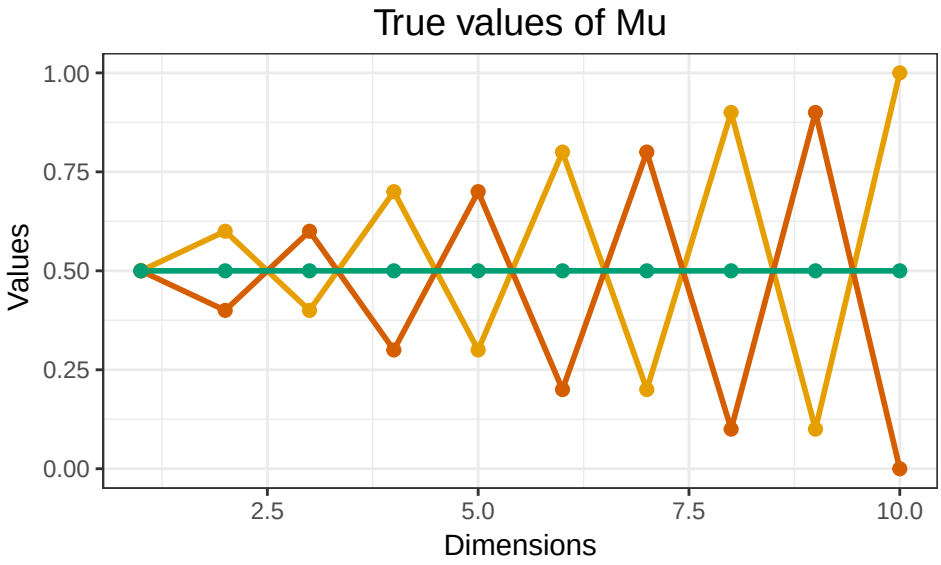
2. Mixture Models

For this task I will implement the expectation-maximization (EM) algorithm for mixtures of multivariate Benouilli distributions. I will perform this with 2,3 and 4 amount of components (K). Theese components are the amount of clusters of the data. For simplicity I have made the code to a function: `EM(K)` and the code is in the appendix.

Before the initliazation of the EM-algorithm I need to "guess" the parameters (μ) and some π where the π:s are the probablilities of a observation to belong to a cluster. Theese paramters and probablilities are evaluated and updated in the algorithm. The algorithm consists of two steps:

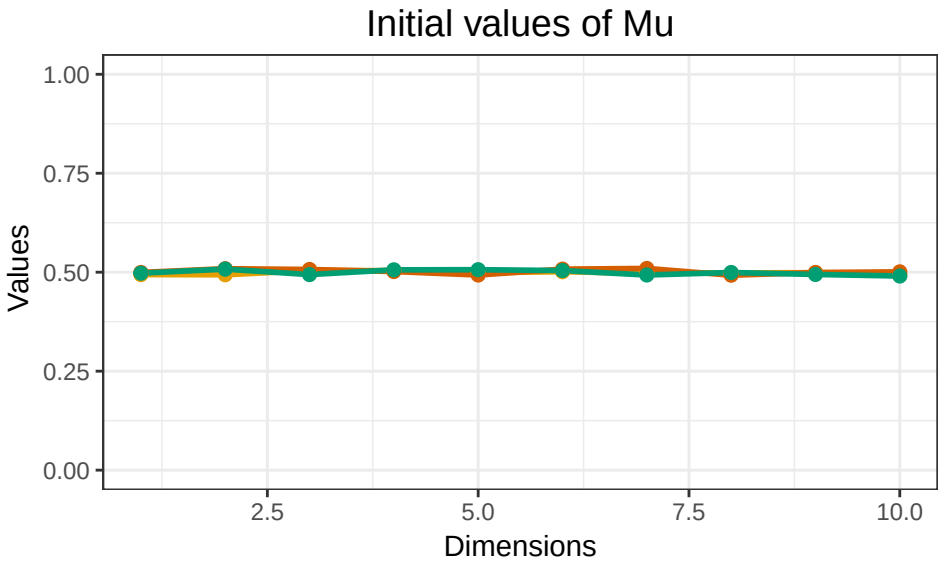
- Step E (expectation) uses the data and paramters (the initial "guesses" for the first iteration) to calculate the posterior probablilities.
- Step M (maximization) uses the posterior probablilities to update the parameters.

This two steps are repeated until the log-likelihood have converged some true value and some parameters are estimated. The true values for this assignment can be vizulized as:

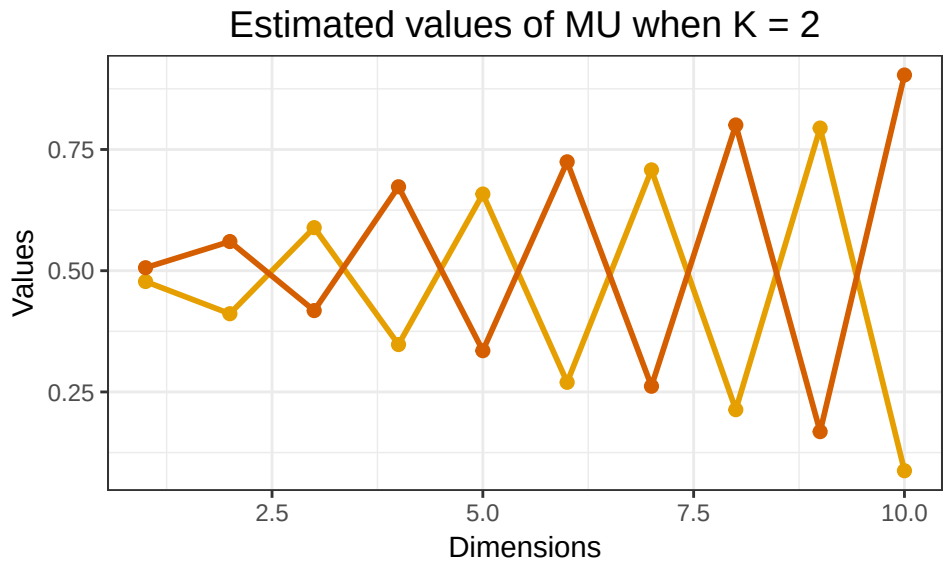


From this plot we can see that there are three true components (clusters) where each line is one compnent and each point is the values of the μ parameters in the Benouilli distribution. For the π in the distribution, they are the probablilities of a observation to belong to a specific cluster. The initial π:s for the first itteration are then "guessed" as well.

The initial values for μ are choosen uniformly such as:



I will now compute the algorithm to estimate theese values and running the algorithm with K = 2 we get the following output:



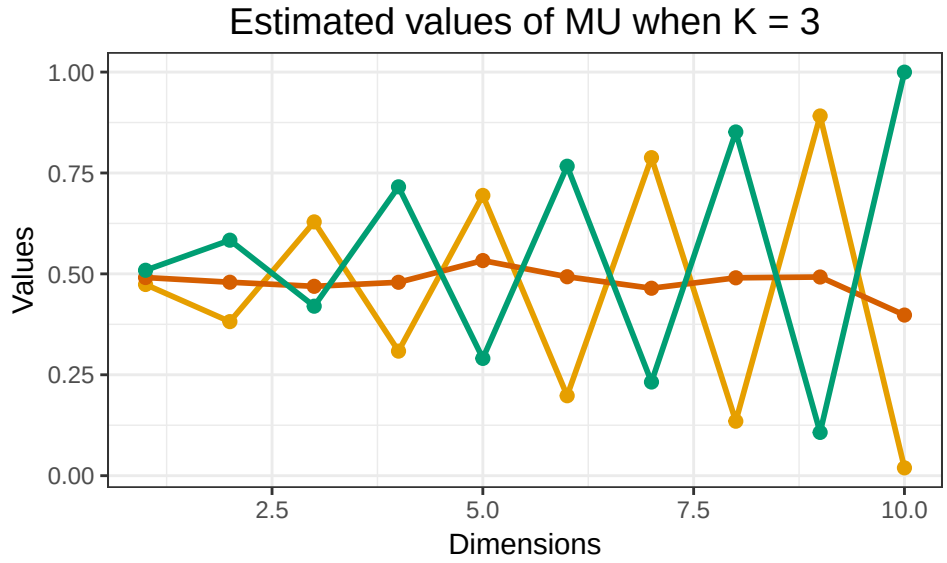
We can see that the two components have converged to the same shape as the true values, but are missing one component. This fit is good though, because the two compnents are both estimated opposites to each other. The π:s are:

```
ext$pi_fit
```

```
## [1] 0.4981919 0.5018081
```

The estimates are very equaly distributed so we can conclude that they are a good fit.

The algorithm for K = 3:



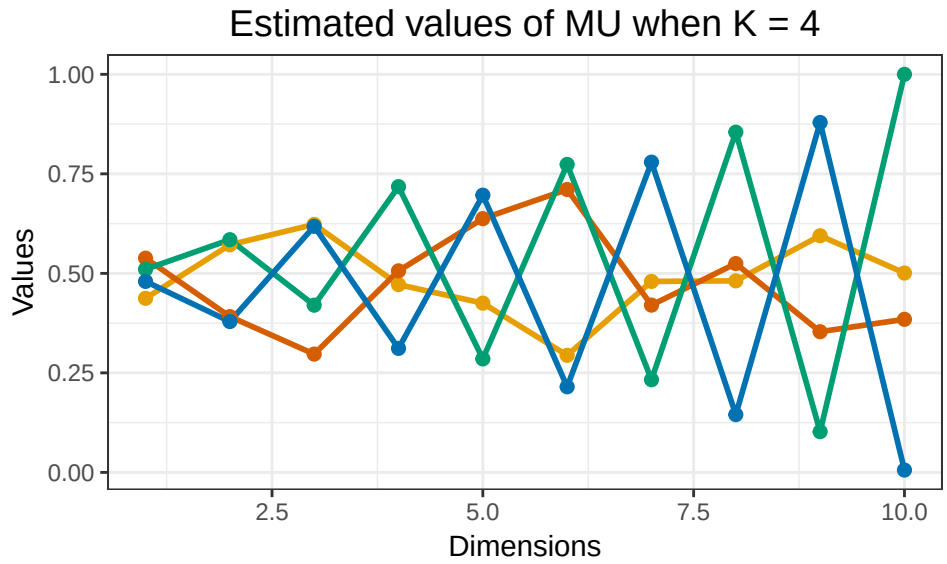
We can see that this estimation of the amount of parameters is a better estmation of the amount of components (clusters) in the data. The π:s are:

```
ext1$pi_fit
```

```
## [1] 0.3259592 0.3044579 0.3695828
```

The estimates are fairly uniformly distributed so we can conclude that they are a good fit. Compared to when K = 2, we can see that the estimates are more different but are still good.

I will also use the algorithm to perform when K = 4:



We can see that with four components that the two compnents in the middle are not good estimates of the true μ. This might be because I am trying to find four clusters but it does only exist three. The π:s are:

```
ext$pi_fit
```

```
## [1] 0.1614155 0.1383613 0.3609912 0.3392319
```

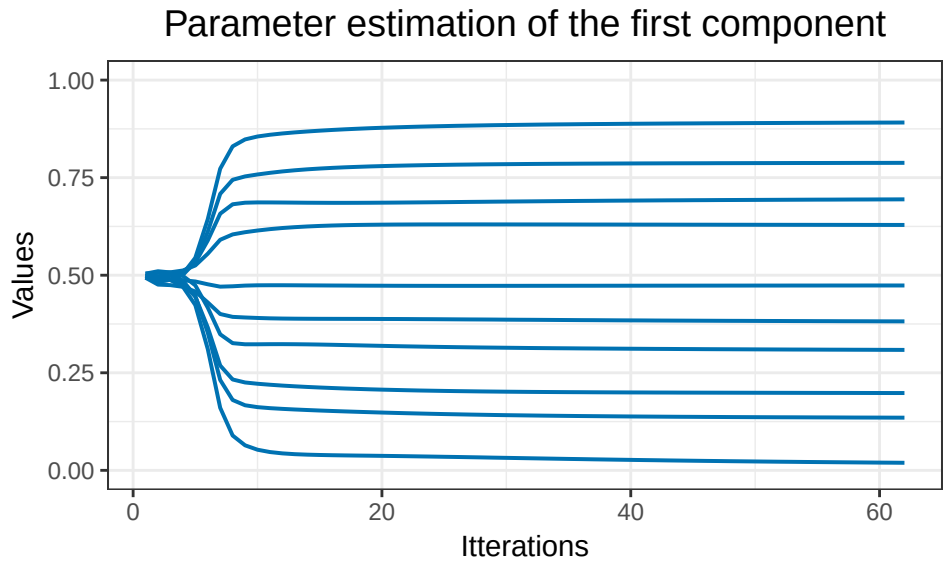
We can see that the estimates are not equally distributed and we can also note that the sum of the two first π:s are almost equal to 1/3. Compared to when K = 2 and K = 3 this on is the worse of the fits due to that we are trying to estimate too many compnents for the data.

The conclusion is that the best and most logical amount of components is when K = 3. Investigating the log-likelihood, when K = 3, we can see if the algorithm are converging:

We can see that the log-likelihood have converged, so the algorithm has found the optimal values for the parameters. The algorithm converges fast so there is no need to run the algorithm too long. We can also investigare if the actual μ:s are converging as well.

I will plot the estimate of the parameters for each itteration to investigate if the paramters have converged.

This will only be shown for the first component:



Each line in the plot represent the parameters for each variable. We can see that the paramters are converging fast in the start, which we also saw in the log-likelihood plot.

Appendix

1

```
tr_miss_ADA <- c()
tst_miss_ADA <- c()

tr_miss_skog <- c()
tst_miss_skog <- c()
n <-0
for(i in seq(10,100,10)){
  n <- n+1
  #Adaboost
  intr <- blackboost(as.factor(Spam) ~ . ,data = data_tr, family = AdaExp(),
                     control = boost_control(mstop = i))
  preed_tst <- predict(intr, newdata = data_tst)
  tabbies <- table(data_tst$Spam, sign(preed_tst))

  tst_miss_ADA[n] <- (tabbies[1,2] + tabbies[2,1])/sum(tabbies)

  #Random forest
  skog <- randomForest(formula = as.factor(Spam) ~ . ,data = train, ntree = i)

  preddie <- predict(skog, newdata = test)
  taubble <- table(test$Spam, preddie)
  tst_miss_skog[n] <- (taubble[1,2] + taubble[2,1])/sum(taubble)
}
```

2

```
EM+ <- function(K){
  set.seed(1234567890)
  max_it <- 100 # max number of EM iterations
  min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
  N=1000 # number of training points
  D=10 # number of dimensions
  x <- matrix(nrow=N, ncol=D) # training data

  true_pi <- vector(length = 3) # true mixing coefficients
  true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
  true_pi=c(1/3, 1/3, 1/3)
  true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
  true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
  true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)

  # Producing the training data
  for(n in 1:N) {
    k <- sample(1:3,1,prob=true_pi)
    for(d in 1:D) {
      x[n,d] <- rbinom(1,1,true_mu[k,d])
    }
```

```
}

K=K # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}

#####################
llik <- c()
for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  temp_z <- z
  for(pis in 1:length(pi)){
    for(n in 1:nrow(x)){

      #the binomial calc
      binom_my <- c(mu[pis,]^x[n,]*(1-mu[pis,])^(1-x[n,]), pi[pis])
      prob_bin <- prod(binom_my)
      temp_z[n, pis] <- prob_bin
    }
  }
  sumsum <- rowSums(temp_z)
  for(i in 1:length(pi)){
    p_Zn_Xn <- temp_z[,i]/sumsum
    z[,i] <- p_Zn_Xn
  }

  ## Log lik
  sum_k <- matrix(nrow=N, ncol=K)
  for(k in 1:ncol(z)){
    sum_i <- c()
    for(row in 1:nrow(x)){
      #calculate the inner sum
      sum_i[row] <- sum(x[row,]*log(mu[k,]) + (1-x[row,])*log(1-mu[k,]))
    }
    #calculate ncol=K amount of vectros with the summs of the dimensions (10)
    sum_k[,k] <- z[,k]*(log(pi[k]) + sum_i)
  }
  sum_k <- rowSums(sum_k)
  llik[it] <- sum(sum_k)

  # Stop if the log likelihood has not changed significantly
  if(it >= 2){
    if(abs(llik[it]-llik[it-1]) <= min_change ){
      break
    }
```

```
    }

    ####
    ##Updating the pis an mu. M-step!
    for(i in 1:ncol(z)){
      mu_v <- c()
      for(j in 1:ncol(x)){
        mu_v[j] <- sum(x[,j]*z[,i])/sum(z[,i])
      }
      #updt the mu
      mu[i,] <- mu_v

      #updt the pi
      pi[i] <- sum(z[,i])/nrow(z)
    }

  }
  return(list(Log_lik = llik, mu_fit = mu, pi_fit = pi, true_mu = true_mu))
}
```