Ensemble Methods and Mixture Models

Maximilian Pfundstein (maxpf364)

27 November 2018

Table of Contents

# 1 Ensemble Methods

Let’s load the dataset and have a look at it. The dataset is truncated to only show the last 10 columns.

spambase.csv

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Word48 | Char1 | Char2 | Char3 | Char4 | Char5 | Char6 | Capitalrun1 | Capitalrun2 | Capitalrun3 | Spam |
| 0 | 0.00 | 0.000 | 0 | 0.778 | 0.000 | 0.000 | 3.756 | 61 | 278 | 1 |
| 0 | 0.00 | 0.132 | 0 | 0.372 | 0.180 | 0.048 | 5.114 | 101 | 1028 | 1 |
| 0 | 0.01 | 0.143 | 0 | 0.276 | 0.184 | 0.010 | 9.821 | 485 | 2259 | 1 |
| 0 | 0.00 | 0.137 | 0 | 0.137 | 0.000 | 0.000 | 3.537 | 40 | 191 | 1 |
| 0 | 0.00 | 0.135 | 0 | 0.135 | 0.000 | 0.000 | 3.537 | 40 | 191 | 1 |
| 0 | 0.00 | 0.223 | 0 | 0.000 | 0.000 | 0.000 | 3.000 | 15 | 54 | 1 |

The source code calls the Random Forest and AdaBoost implementation and uses the predict function of each for getting the error rates for the training and the validation data set. The functions are called for 10, 20, …, 100 trees.

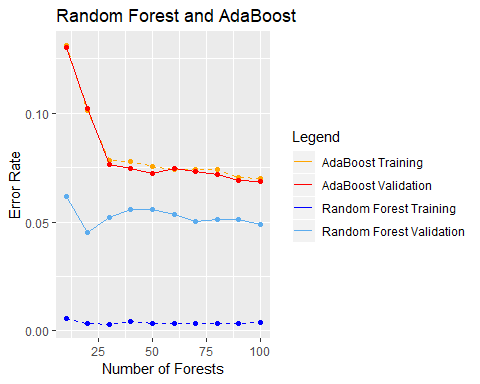
The following tables show the error rates for Random Forst and AdaBoost. The plot visualizes this data, the dashed lines represent the performance on the training data set.

Error rates for Random Forest

|  |  |  |
| --- | --- | --- |
| n | error\_rate\_training | error\_rate\_validation |
| 10 | 0.0055159 | 0.0618828 |
| 20 | 0.0035691 | 0.0454246 |
| 30 | 0.0029202 | 0.0520079 |
| 40 | 0.0042180 | 0.0559579 |
| 50 | 0.0035691 | 0.0559579 |
| 60 | 0.0032446 | 0.0533246 |
| 70 | 0.0035691 | 0.0500329 |
| 80 | 0.0032446 | 0.0513496 |
| 90 | 0.0035691 | 0.0513496 |
| 100 | 0.0038936 | 0.0487163 |

Error rates for AdaBoost

|  |  |  |
| --- | --- | --- |
| n | error\_rate\_training | error\_rate\_validation |
| 10 | 0.1310837 | 0.1303489 |
| 20 | 0.1012330 | 0.1020408 |
| 30 | 0.0781960 | 0.0763660 |
| 40 | 0.0778715 | 0.0743910 |
| 50 | 0.0756003 | 0.0724161 |
| 60 | 0.0743024 | 0.0743910 |
| 70 | 0.0743024 | 0.0730744 |
| 80 | 0.0743024 | 0.0717577 |
| 90 | 0.0704088 | 0.0691244 |
| 100 | 0.0700844 | 0.0684661 |



We can observe that the Random Forest is performing way better than AdaBoost. Still AdaBoost seems to perform way better on the training with respect to the validation data set than compared to the Random Forest which has a big gap between the training and validation data set. The only thing which seems to be weird is that Adaboost is actually performing better on the validation data set compared to the training data set. This behavior changes when the seed is changed, so it might be just an occasion.

# 2 Mixture Models

For the EM-algorithm one must calculate the matrix, the likelihood and the new and for each iteration. These can be done with matrix multiplications and thus in the followingit will be explained which formulas were used and what the values , , , and mean.

## Explanation Of Involved Values

We have which holds entries with dimensions In the real world we will not know from how many components our data is derived nor how many entries belong to each component.

In this example holds our estimate how many datapoints from belong to (which is for each, but let’s suppose we don’t know.)

represents the probability for each to be head or tail for each .

The likelihood is basically the expected value of our parameters that our model derived from the data. The goal is to maximize tis value iteratively. Due to [Jensen’s inequality](https://en.wikipedia.org/wiki/Jensen%27s_inequality) it can be proofed that the likelihood is always greater or equal in the next iteration step of the EM-algorithm [Gently Building Up the EM Algorithm](https://abidlabs.github.io/EM-Algorithm/#mjx-eqn-eq5). Thus we will at least not get worste with each iteration, we shold just care if the likelihood doesn’t improved much with an itration and then stop. The is set to (note that we speak about the ln-likelihood).

holds the probability for each set of (row in ) to belong to each . We always choose the one which most likely fits to the observed data and use that to update ouzr priors.

## Mathematical Equations

Let’s have a look at the mathematical equations and how we can derive our formulas for the matrix multiplication from that. Formulas without a source are either taken from the lecture slides or derived by previous formulas.

The first step is to calculate . We will divide that in first calculating which holds and afterwards calculating which we can use to calculate . The formales will reduce the left side to single letters which you will find in the source code.

For using matrix multiplication we need to get rid of the product and the exponents, so we use on both sides:

Now let’s get rid of the on the left side:

This craves to be put into a neat matrix multiplication. Okay, let’s look for .

We will use later to calculate the likelihood but for now let’s calculate Z:

The likelihood is given by the following equation which can be found in (Bishop 2006) on page 433 equation 9.14.

As we already have the value inside the curly braces () it’s basically just the sum of the logarithms over .

For calculating we use the following.

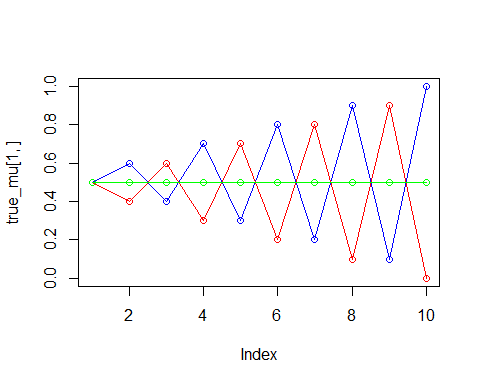
And finally we use the following for calculating . Note that the nominator of and the denominator of are the same.

Voilà we’re done, now the coding is actually just a few lines of code, you’ll find in in the appendix.

## Execution

### True Values for and

These are the true values for and :



true\_pi

|  |
| --- |
| x |
| 0.3333333 |
| 0.3333333 |
| 0.3333333 |

true\_mu

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0.5 | 0.6 | 0.4 | 0.7 | 0.3 | 0.8 | 0.2 | 0.9 | 0.1 | 1.0 |
| 0.5 | 0.4 | 0.6 | 0.3 | 0.7 | 0.2 | 0.8 | 0.1 | 0.9 | 0.0 |
| 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |

### K = 2

pi

|  |
| --- |
| x |
| 0.4985558 |
| 0.5014442 |

mu

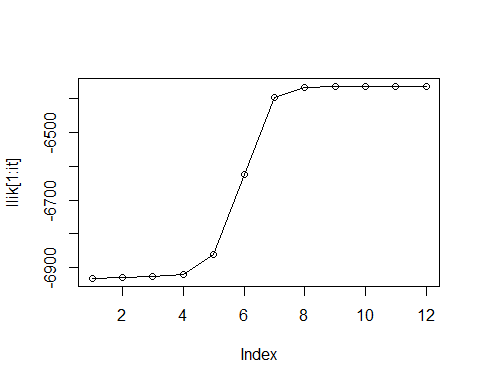
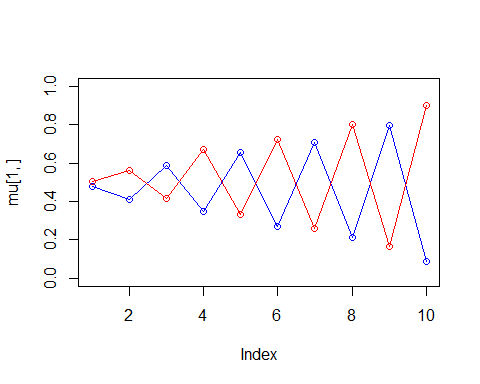
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0.4776212 | 0.4115474 | 0.5888350 | 0.3476584 | 0.657974 | 0.2694835 | 0.7080466 | 0.2129201 | 0.7946025 | 0.0895949 |
| 0.5062960 | 0.5600237 | 0.4176594 | 0.6734008 | 0.334959 | 0.7252002 | 0.2612440 | 0.8013802 | 0.1672096 | 0.9020584 |

Number of Iterations

|  |
| --- |
| x |
| 12 |

Ln-Likelihood

|  |
| --- |
| x |
| -6362.891 |



### K = 3

pi

|  |
| --- |
| x |
| 0.3272335 |
| 0.3300263 |
| 0.3427403 |

mu

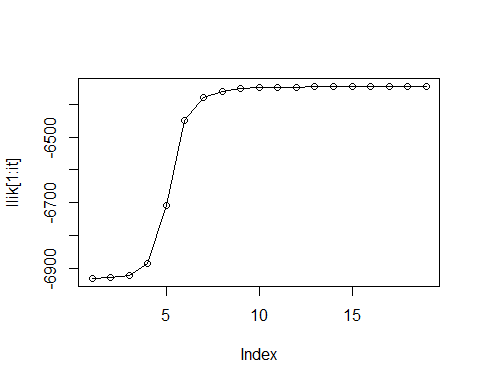
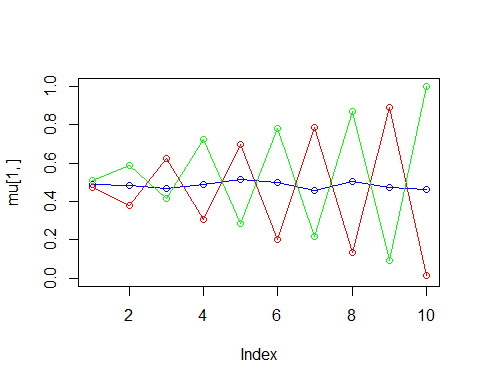
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0.4891351 | 0.4855618 | 0.4686094 | 0.4907012 | 0.5136872 | 0.5001229 | 0.4581237 | 0.5064250 | 0.4737710 | 0.4605612 |
| 0.4748950 | 0.3814730 | 0.6255717 | 0.3087514 | 0.6973317 | 0.2008993 | 0.7846864 | 0.1360473 | 0.8901803 | 0.0131365 |
| 0.5112058 | 0.5870679 | 0.4178097 | 0.7251266 | 0.2852498 | 0.7820528 | 0.2191731 | 0.8676588 | 0.0909826 | 0.9977047 |

Number of Iterations

|  |
| --- |
| x |
| 19 |

Ln-Likelihood

|  |
| --- |
| x |
| -6344.83 |



### K = 4

pi

|  |
| --- |
| x |
| 0.2451226 |
| 0.2474470 |
| 0.2545552 |
| 0.2528752 |

mu

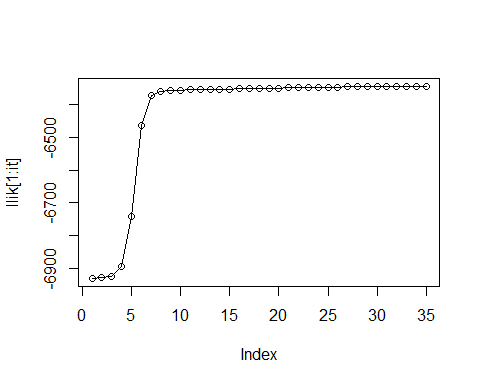
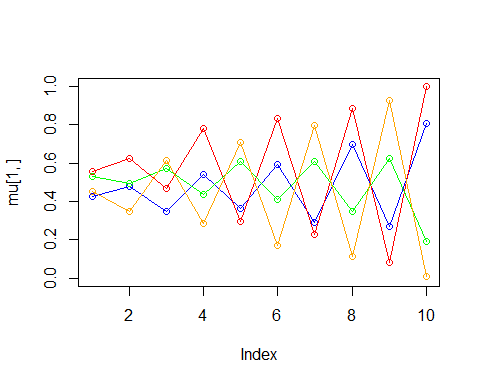
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0.4275214 | 0.4794378 | 0.3490801 | 0.5424050 | 0.3629693 | 0.5917624 | 0.2926573 | 0.6986396 | 0.2698976 | 0.8092181 |
| 0.5575234 | 0.6254826 | 0.4669151 | 0.7816688 | 0.2979063 | 0.8334707 | 0.2280754 | 0.8828056 | 0.0834857 | 0.9985014 |
| 0.5293161 | 0.4936496 | 0.5735814 | 0.4394978 | 0.6072886 | 0.4086347 | 0.6072466 | 0.3502206 | 0.6255901 | 0.1926651 |
| 0.4528209 | 0.3481721 | 0.6164610 | 0.2876763 | 0.7067658 | 0.1688016 | 0.7958422 | 0.1152724 | 0.9251067 | 0.0099744 |

Number of Iterations

|  |
| --- |
| x |
| 35 |

Ln-Likelihood

|  |
| --- |
| x |
| -6343.522 |



### Conclusions

We can see that with the model is underfitting as the line for is completely missing while with the model is overfitting. For the model is acutally not performing that bad while for we can see that the green line (from the real) data is missing and that the *new* green and blue line are dancing around the *invisible* line at . It looks like the average of the blue and the green line would form the true missing line.

For , so with the true number of components, the models converges fastest.

# Appendix

knitr::opts\_chunk$set(echo = TRUE)  
library(mboost)  
library(randomForest)  
library(ggplot2)  
library(knitr)  
set.seed(1234567890)  
  
################################################################################  
# 1. Ensemble Methods  
################################################################################  
  
spambase = read.csv("spambase.csv", sep=";", dec = ",")  
spambase$Spam = as.factor(spambase$Spam)  
  
n = dim(spambase)[1]  
id = sample(1:n, floor(n\*0.67))  
train\_spambase = spambase[id,]  
val\_spambase = spambase[-id,]  
  
kable(head(spambase[,48:58]), caption = "spambase.csv")  
  
  
# General Information  
c\_formula = Spam ~ .  
tree\_sizes = seq(from = 10, to = 100, by = 10)  
  
# Random Forest  
rf\_errors = data.frame(n = numeric(), error\_rate\_training = numeric(),  
 error\_rate\_validation = numeric())  
  
for (i in tree\_sizes) {  
   
 # Create the forest  
 c\_randomForest =  
 randomForest(formula = c\_formula, data = train\_spambase, ntree = i)  
   
 # Do the prediction on the validation dataset  
 c\_prediction\_training =  
 predict(object = c\_randomForest, newdata = train\_spambase)  
 c\_prediction\_validation =  
 predict(object = c\_randomForest, newdata = val\_spambase)  
   
 # Get the error rate  
 c\_error\_rate\_training = 1 - sum(c\_prediction\_training ==  
 train\_spambase$Spam)/nrow(train\_spambase)  
 c\_error\_rate\_validation = 1 - sum(c\_prediction\_validation ==  
 val\_spambase$Spam)/nrow(val\_spambase)  
   
 rf\_errors = rbind(rf\_errors,  
 list(n = i,  
 error\_rate\_training = c\_error\_rate\_training,  
 error\_rate\_validation = c\_error\_rate\_validation))  
}  
  
# AdaBoost  
adb\_errors = data.frame(n = numeric(), error\_rate\_training = numeric(),  
 error\_rate\_validation = numeric())  
  
for (i in tree\_sizes) {  
  
 # Create the model  
 c\_adaBoost = blackboost(formula = c\_formula,  
 data = train\_spambase,  
 family = AdaExp(),  
 control=boost\_control(mstop=i))  
   
 # Do the prediction on the validation dataset  
 c\_prediction\_training =  
 predict(object = c\_adaBoost, newdata = train\_spambase, type = "class")  
 c\_prediction\_validation =  
 predict(object = c\_adaBoost, newdata = val\_spambase, type = "class")  
   
 # Get the error rate  
 c\_error\_rate\_training = 1 - sum(c\_prediction\_training ==  
 train\_spambase$Spam)/nrow(train\_spambase)  
 c\_error\_rate\_validation = 1 - sum(c\_prediction\_validation ==  
 val\_spambase$Spam)/nrow(val\_spambase)  
   
 adb\_errors = rbind(adb\_errors,  
 list(n = i, error\_rate\_training = c\_error\_rate\_training,  
 error\_rate\_validation = c\_error\_rate\_validation))  
   
}  
  
  
kable(rf\_errors, caption = "Error rates for Random Forest")  
kable(adb\_errors, caption = "Error rates for AdaBoost")  
  
ggplot(adb\_errors) +  
 geom\_line(aes(x = n, y = error\_rate\_training,  
 colour = "AdaBoost Training"), linetype = "dashed") +  
 geom\_point(aes(x = n, y = error\_rate\_training), colour = "orange") +  
   
 geom\_line(aes(x = n, y = error\_rate\_validation,  
 colour = "AdaBoost Validation")) +  
 geom\_point(aes(x = n, y = error\_rate\_validation), colour = "red") +  
   
 geom\_line(aes(x = n, y = error\_rate\_training,  
 colour = "Random Forest Training"),  
 data = rf\_errors, linetype = "dashed") +  
 geom\_point(aes(x = n, y = error\_rate\_training),  
 colour = "blue", data = rf\_errors) +  
   
 geom\_line(aes(x = n, y = error\_rate\_validation,  
 colour = "Random Forest Validation"), data = rf\_errors) +  
 geom\_point(aes(x = n, y = error\_rate\_validation),  
 colour = "steelblue2", data = rf\_errors) +  
 labs(title = "Random Forest and AdaBoost", y = "Error Rate",  
 x = "Number of Forests", color = "Legend") +  
 scale\_color\_manual(values = c("orange", "red", "blue", "steelblue2"))  
  
  
################################################################################  
# 2. Mixture Models  
################################################################################  
  
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])  
 }  
}  
  
  
plot(true\_mu[1,], type="o", col="blue", ylim=c(0,1))  
points(true\_mu[2,], type="o", col="red")  
points(true\_mu[3,], type="o", col="green")  
  
kable(true\_pi, caption = "true\_pi")  
kable(true\_mu, caption = "true\_mu")  
  
  
################################################################################  
# K = 2  
################################################################################  
  
set.seed(1234567890)  
K = 2 # 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)  
}  
  
for(it in 1:max\_it) {  
 # E-step: Computation of the fractional component assignments  
 Bx = exp(x %\*% log(t(mu)) + (1-x) %\*% log(t(1-mu)))  
 Px = Bx \* rep(pi, nrow(Bx))  
 Z = Px / rowSums(Px)  
 #Log likelihood computation.  
 L = sum(log(rowSums(Px)))  
 llik[it] = L  
   
 # Stop if the lok likelihood has not changed significantly  
 if (it > 1 && abs(llik[it-1] - llik[it]) < min\_change) break  
   
 #M-step: ML parameter estimation from the data and fractional component assignments  
 pi = colSums(Z) / N  
 mu = (t(Z) %\*% x) / colSums(Z)  
}  
  
  
kable(pi, caption = "pi")  
kable(mu, caption = "mu")  
kable(it, caption = "Number of Iterations")  
kable(llik[it], caption = "Ln-Likelihood")  
plot(mu[1,], type="o", col="blue", ylim=c(0,1))  
points(mu[2,], type="o", col="red")  
plot(llik[1:it], type="o")  
  
  
################################################################################  
# K = 3  
################################################################################  
  
set.seed(1234567890)  
K = 3 # 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)  
}  
  
for(it in 1:max\_it) {  
 # E-step: Computation of the fractional component assignments  
 Bx = exp(x %\*% log(t(mu)) + (1-x) %\*% log(t(1-mu)))  
 Px = Bx \* rep(pi, nrow(Bx))  
 Z = Px / rowSums(Px)  
 #Log likelihood computation.  
 L = sum(log(rowSums(Px)))  
 llik[it] = L  
   
 # Stop if the lok likelihood has not changed significantly  
 if (it > 1 && abs(llik[it-1] - llik[it]) < min\_change) break  
   
 #M-step: ML parameter estimation from the data and fractional component assignments  
 pi = colSums(Z) / N  
 mu = (t(Z) %\*% x) / colSums(Z)  
}  
  
  
kable(pi, caption = "pi")  
kable(mu, caption = "mu")  
kable(it, caption = "Number of Iterations")  
kable(llik[it], caption = "Ln-Likelihood")  
plot(mu[1,], type="o", col="blue", ylim=c(0,1))  
points(mu[2,], type="o", col="red")  
points(mu[3,], type="o", col="green")  
plot(llik[1:it], type="o")  
  
  
################################################################################  
# K = 4  
################################################################################  
  
set.seed(1234567890)  
K = 4 # 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)  
}  
  
for(it in 1:max\_it) {  
 # E-step: Computation of the fractional component assignments  
 Bx = exp(x %\*% log(t(mu)) + (1-x) %\*% log(t(1-mu)))  
 Px = Bx \* rep(pi, nrow(Bx))  
 Z = Px / rowSums(Px)  
 #Log likelihood computation.  
 L = sum(log(rowSums(Px)))  
 llik[it] = L  
   
 # Stop if the lok likelihood has not changed significantly  
 if (it > 1 && abs(llik[it-1] - llik[it]) < min\_change) break  
   
 #M-step: ML parameter estimation from the data and fractional component assignments  
 pi = colSums(Z) / N  
 mu = (t(Z) %\*% x) / colSums(Z)  
}  
  
  
kable(pi, caption = "pi")  
kable(mu, caption = "mu")  
kable(it, caption = "Number of Iterations")  
kable(llik[it], caption = "Ln-Likelihood")  
plot(mu[1,], type="o", col="blue", ylim=c(0,1))  
points(mu[2,], type="o", col="red")  
points(mu[3,], type="o", col="green")  
points(mu[4,], type="o", col="orange")  
plot(llik[1:it], type="o")

# Bibliography

Bishop, Christopher M. 2006. “Pattern Recognition and Machine Learning.” Springer Science; Business Media, LLC.