lab1block2_bjorn_hansen

Bjorn_Hansen

12/3/2019

1. ENSEMBLE METHODS

To start, the data is loaded into test and train sets with 2/3 of the data being used for training and 1/3 used for testing.

Adaboost

Adaboost classification trees are used to train the model.Inside the boost control argument a value of 0.6 for "nu" was used as this value gave the lowest error. Adaboost predicts assignment through majority voting, the comand type = "class" is used to do this. As can be seen from the confusion matrix below (using 100 trees) the diagonal values are quite high meaning that the ada model is predicting with high accuaracy.

```
## Loading required package: parallel
```

```
## Loading required package: stabs
```

```
## This is mboost 2.9-1. See 'package?mboost' and 'news(package  = "mboost")'
## for a complete list of changes.
```

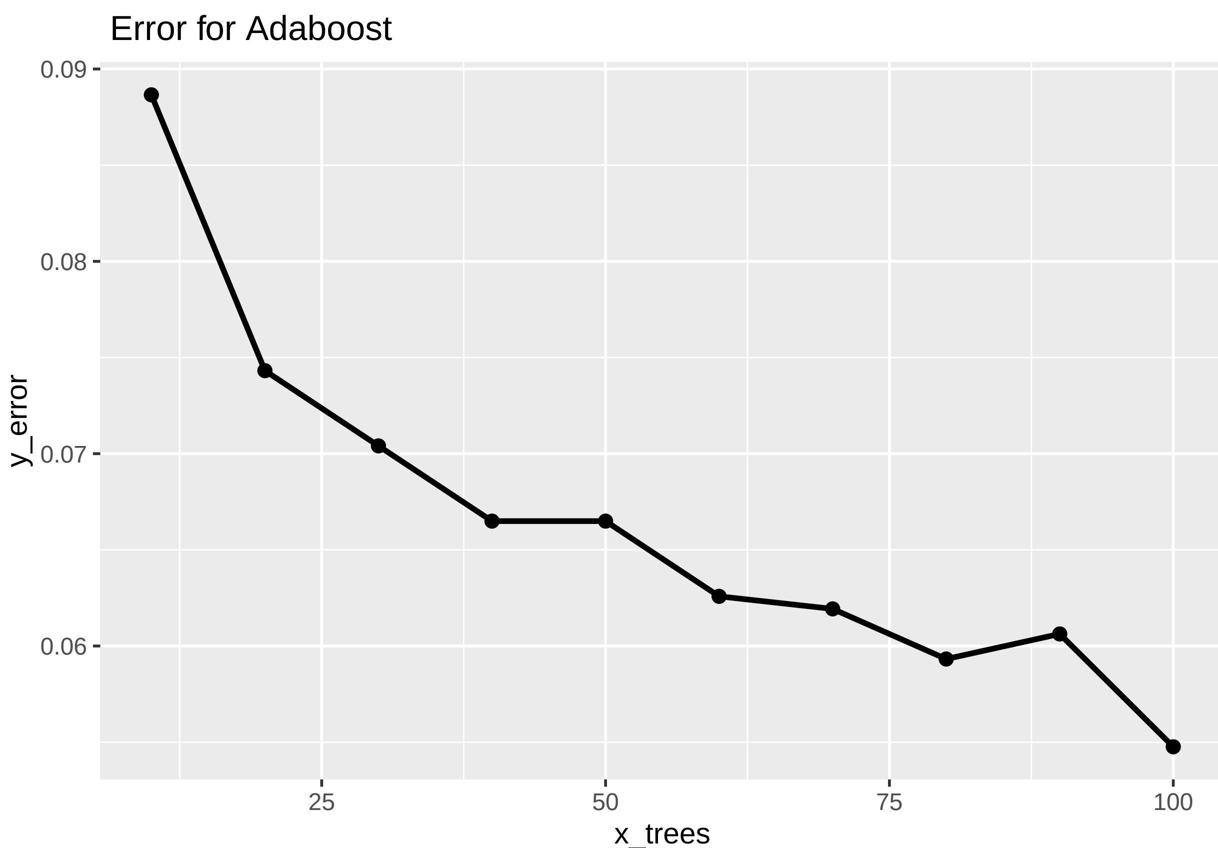
```
##    Y hat
## Y     0   1
##   0 903  38
##   1  46 547
```

```
## [1] "Error for ada model: 0.0547588005215124"
```

Below is a plot of the recorded error for 10 up to 100 trees.

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:mboost':
##
##     %+%
```



Random Forest

A similar procedure was used to classify the spam data, this time using random forests. Again 10 up to 100 trees were used to train the model. The random forest model performed very well and had slightly better results than the adaboost model.

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

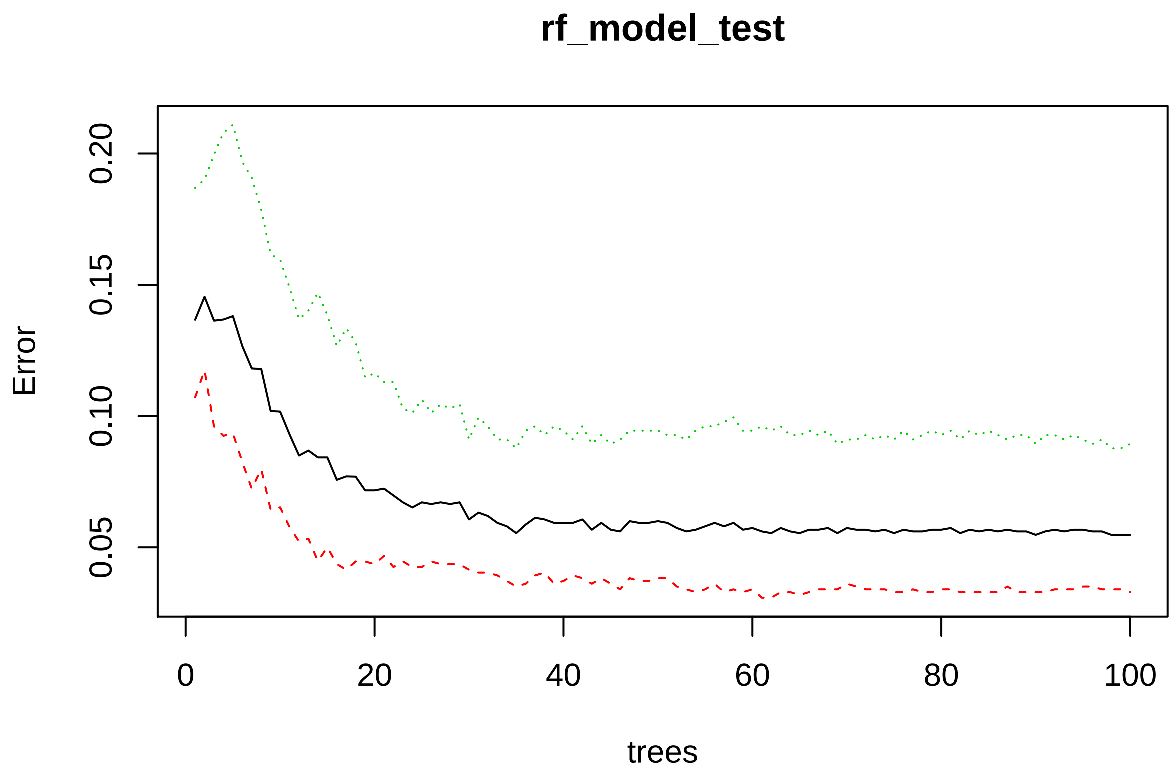
```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
##    Y hat
## Y     0   1
##   0 914  27
##   1  40 553
```

```
## [1] "Error random forest model: 0.0436766623207301"
```

In the plot below how the random forest error decreases linearly until about 30 trees but then stays at around 0.05 as the number of trees grows.



2. MIXTURE MODELS

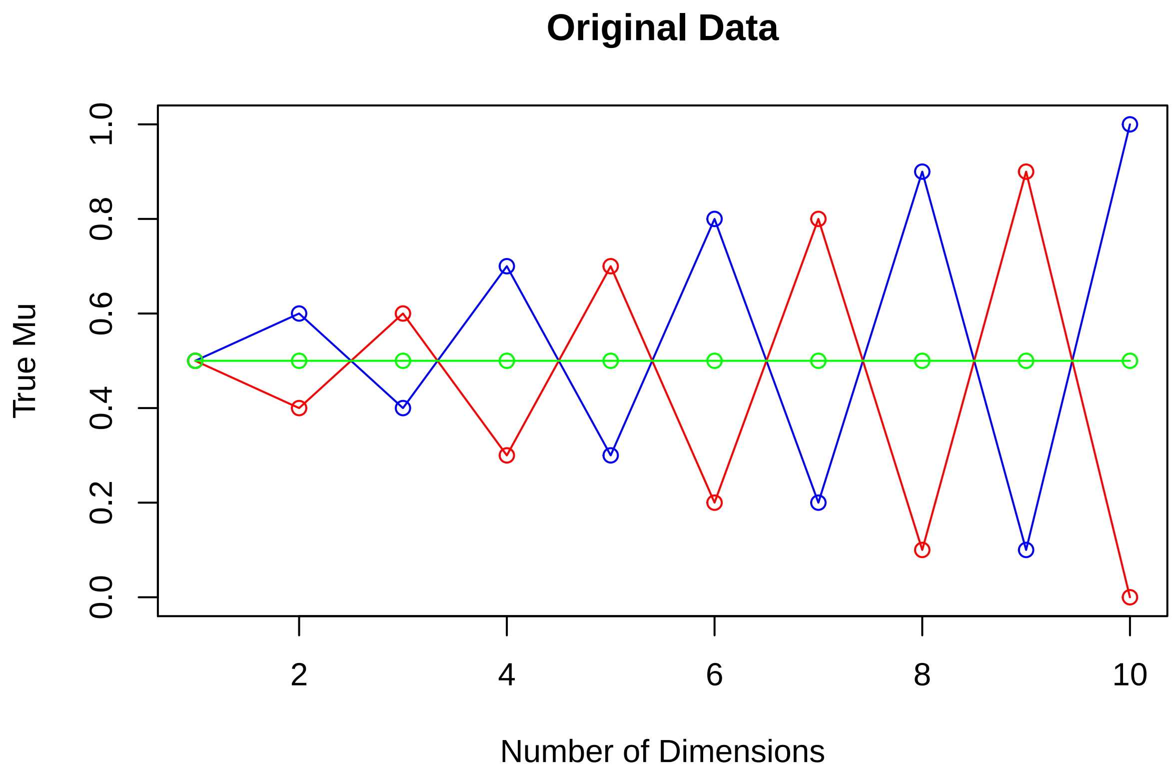
Below are plots for the em algorithm. We start by creating some random data. The obejctive of the em algorithm is to estimate given data points with the help of a bayesian approach. The em function takes data to try and estimate mu and pi.

In the so called E- step we are computing the posterior values for each observation via Bayes Theorem. We assume a certain ammount of clusters K, for example if K = 2 we are assuming the points are divided into two clusters.

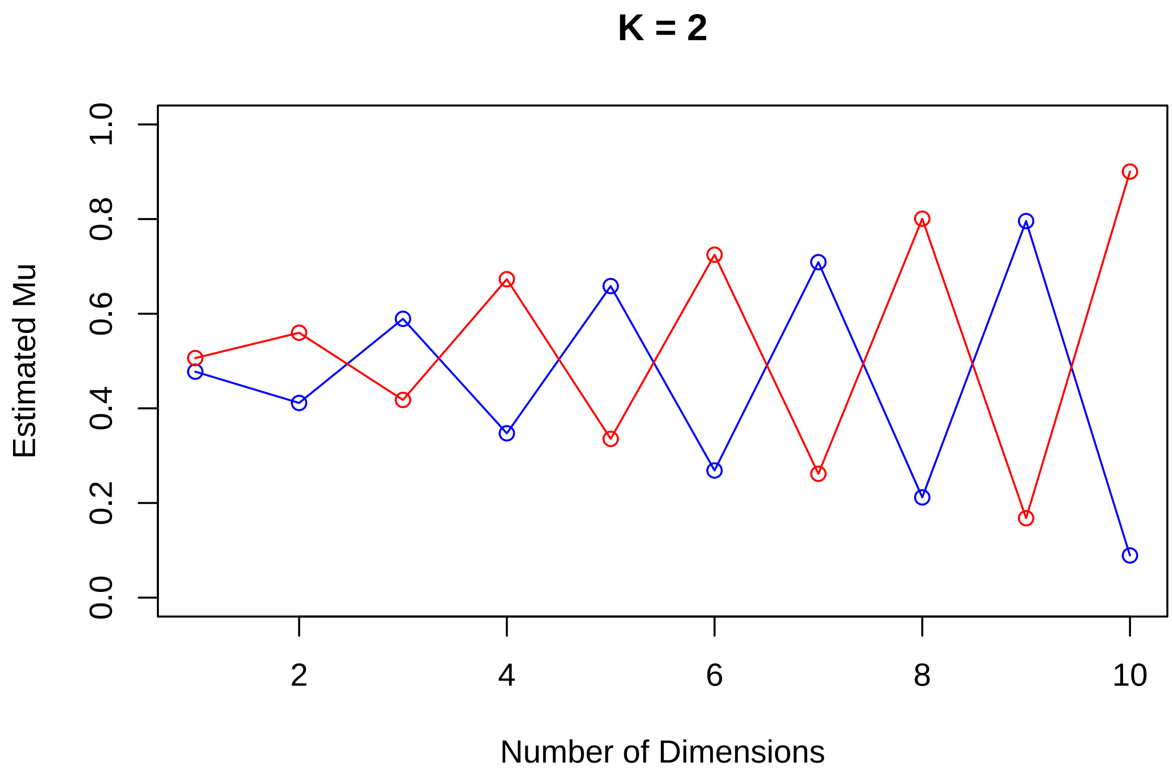
In the so called M- step our previously estimated mu and pi are updated based on the new values. The loglikelihood of these values are calculated. This process continues itereating (100 times in our case) until are certain threshold of change between iterations is met.

The EM algorithm is a great solution when for example values for data points are missing from data sets. A risk that this algorithm has is that the true mu for the data distribution can be unknown and the algorithm can iterate and converge on a local maximi point instead of the global maximi point which is the true mu. This can lead to estimates that are not completely correct.

Below are the results for when K is equal to two, three, and four. For all K values there is very like change between iterations after iteration eight. It is seen from the iterations that when K = 2 the convergence happens the quickest with only 12 iterations needed for the threshold to be met. K = 3 had the most interations with 46 total. This is probably due to the for two mus overlappping the third which made it not clear to distinguish it from the others.



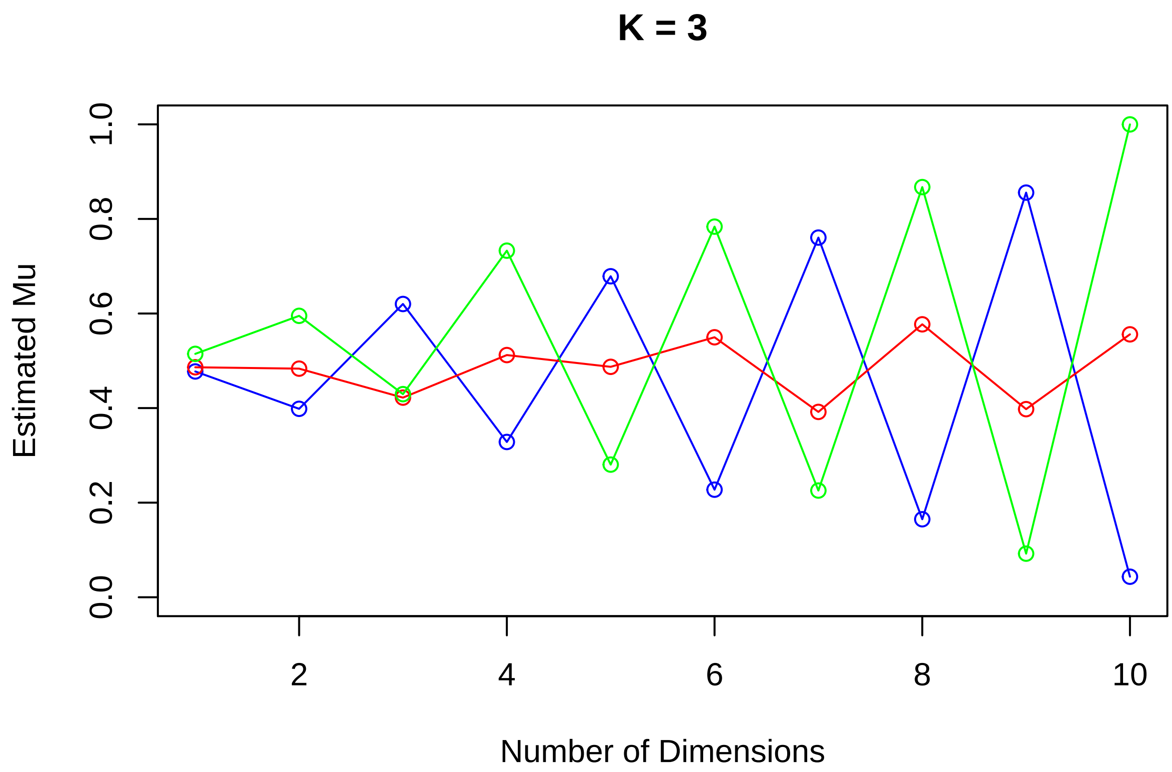

```
## iteration:  1 log likelihood:  -6930.975
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.055
## iteration:  6 log likelihood:  -6728.948
## iteration:  7 log likelihood:  -6443.28
## iteration:  8 log likelihood:  -6368.318
## iteration:  9 log likelihood:  -6363.734
## iteration:  10 log likelihood:  -6363.109
## iteration:  11 log likelihood:  -6362.947
## iteration:  12 log likelihood:  -6362.897
```




```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775488 0.4113939 0.5892308 0.3472420 0.6583712 0.2686589 0.7089490
## [2,] 0.5062860 0.5597531 0.4177551 0.6728856 0.3354854 0.7247188 0.2616231
##            [,8]      [,9]      [,10]
## [1,] 0.2118629 0.7957549 0.08905747
## [2,] 0.8007511 0.1678555 0.90027808
```

```
## iteration:  1 log likelihood:  -6931.064
## iteration:  2 log likelihood:  -6928.051
## iteration:  3 log likelihood:  -6920.026
## iteration:  4 log likelihood:  -6864.176
## iteration:  5 log likelihood:  -6634.916
## iteration:  6 log likelihood:  -6409.234
## iteration:  7 log likelihood:  -6373.593
## iteration:  8 log likelihood:  -6367.833
## iteration:  9 log likelihood:  -6364.983
## iteration:  10 log likelihood:  -6363.074
## iteration:  11 log likelihood:  -6361.594
## iteration:  12 log likelihood:  -6360.309
## iteration:  13 log likelihood:  -6359.103
## iteration:  14 log likelihood:  -6357.93
## iteration:  15 log likelihood:  -6356.786
## iteration:  16 log likelihood:  -6355.689
## iteration:  17 log likelihood:  -6354.668
## iteration:  18 log likelihood:  -6353.742
```

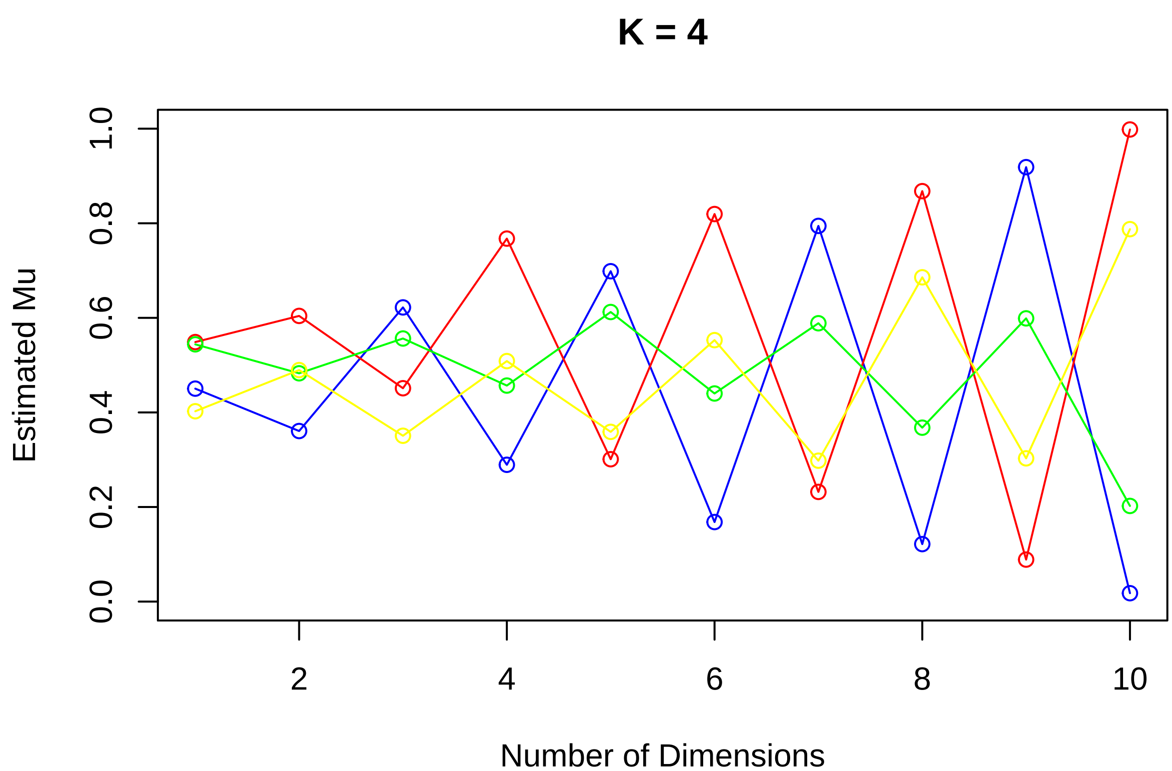
```
## iteration:  19 log likelihood:  -6352.92
## iteration:  20 log likelihood:  -6352.199
## iteration:  21 log likelihood:  -6351.567
## iteration:  22 log likelihood:  -6351.011
## iteration:  23 log likelihood:  -6350.515
## iteration:  24 log likelihood:  -6350.069
## iteration:  25 log likelihood:  -6349.661
## iteration:  26 log likelihood:  -6349.286
## iteration:  27 log likelihood:  -6348.938
## iteration:  28 log likelihood:  -6348.616
## iteration:  29 log likelihood:  -6348.315
## iteration:  30 log likelihood:  -6348.036
## iteration:  31 log likelihood:  -6347.776
## iteration:  32 log likelihood:  -6347.534
## iteration:  33 log likelihood:  -6347.308
## iteration:  34 log likelihood:  -6347.099
## iteration:  35 log likelihood:  -6346.904
## iteration:  36 log likelihood:  -6346.722
## iteration:  37 log likelihood:  -6346.553
## iteration:  38 log likelihood:  -6346.394
## iteration:  39 log likelihood:  -6346.246
## iteration:  40 log likelihood:  -6346.107
## iteration:  41 log likelihood:  -6345.977
## iteration:  42 log likelihood:  -6345.854
## iteration:  43 log likelihood:  -6345.739
## iteration:  44 log likelihood:  -6345.63
## iteration:  45 log likelihood:  -6345.528
## iteration:  46 log likelihood:  -6345.431
```




```
##             [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4774399 0.3984407 0.6200854 0.3283412 0.6787726 0.2274644 0.7605397
## [2,] 0.4863053 0.4834283 0.4221438 0.5121400 0.4872729 0.5497432 0.3918968
## [3,] 0.5146518 0.5950474 0.4294967 0.7329049 0.2804651 0.7837214 0.2255769
##             [,8]       [,9]      [,10]
## [1,] 0.1649014 0.85568380 0.04340808
## [2,] 0.5771323 0.39774130 0.55592505
## [3,] 0.8673460 0.09215422 0.99992821
```

```
## iteration:  1 log likelihood:  -6930.838
## iteration:  2 log likelihood:  -6928.641
## iteration:  3 log likelihood:  -6924.748
## iteration:  4 log likelihood:  -6896.25
## iteration:  5 log likelihood:  -6741.896
## iteration:  6 log likelihood:  -6452.658
## iteration:  7 log likelihood:  -6366.493
## iteration:  8 log likelihood:  -6359.764
## iteration:  9 log likelihood:  -6357.876
## iteration:  10 log likelihood:  -6356.372
## iteration:  11 log likelihood:  -6354.86
## iteration:  12 log likelihood:  -6353.31
## iteration:  13 log likelihood:  -6351.776
## iteration:  14 log likelihood:  -6350.33
## iteration:  15 log likelihood:  -6349.03
## iteration:  16 log likelihood:  -6347.908
```

```
## iteration:  17 log likelihood:  -6346.968
## iteration:  18 log likelihood:  -6346.196
## iteration:  19 log likelihood:  -6345.566
## iteration:  20 log likelihood:  -6345.055
## iteration:  21 log likelihood:  -6344.637
## iteration:  22 log likelihood:  -6344.293
## iteration:  23 log likelihood:  -6344.008
## iteration:  24 log likelihood:  -6343.768
## iteration:  25 log likelihood:  -6343.563
## iteration:  26 log likelihood:  -6343.387
## iteration:  27 log likelihood:  -6343.233
## iteration:  28 log likelihood:  -6343.097
## iteration:  29 log likelihood:  -6342.975
## iteration:  30 log likelihood:  -6342.864
## iteration:  31 log likelihood:  -6342.762
## iteration:  32 log likelihood:  -6342.668
```




```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4502917 0.3606587 0.6220817 0.2892407 0.6986320 0.1681768 0.7943990
## [2,] 0.5487864 0.6040921 0.4511711 0.7675478 0.3010522 0.8195305 0.2318913
## [3,] 0.5439579 0.4827437 0.5563603 0.4568300 0.6123061 0.4400351 0.5885625
## [4,] 0.4025047 0.4895637 0.3506597 0.5085745 0.3588983 0.5528693 0.2979403
##           [,8]       [,9]      [,10]
## [1,] 0.1215732 0.91870837 0.01774129
## [2,] 0.8679302 0.08877946 0.99833984
```

```
## [3,] 0.3677877 0.59890299 0.20227253
## [4,] 0.6857315 0.30292227 0.78760041
```

Appendix

```
library(mboost)
library(readxl)
library(randomForest)
library(ggplot2)

setwd("C:/Users/Bjorn/Documents/LIU/machine_learning")
#data=read_excel("spambase.xlsx")

sp = read.csv2("spambase_lab1_block2.csv")
sp$Spam = as.factor(sp$Spam)

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

## Adaboost ##

ada_model = blackboost(Spam ~., data = train, control = boost_control(mstop = 100, nu=0.6),
                       family = AdaExp())

acc=0
n = seq(10, 100, by = 10)
for(i in n){
  ada_model = blackboost(Spam ~., data = train, control = boost_control(mstop = i, nu = 0.6),
                      family = AdaExp())
  fitted.results_test= predict(ada_model,newdata=test, type = "class")
  #fitted.results_test = ifelse(fitted.results_test > 0.1,1,0)
  misClasificError_test = mean(fitted.results_test != test$Spam)
  ada_confmat_test = table("Y"=test$Spam,"Y hat"=fitted.results_test)
  acc[i] = sum(diag(ada_confmat_test)) / sum(ada_confmat_test)
}
acc

y_error <- na.omit(1-acc)
y_error = y_error[-1]
x_trees <- n
ada_m <- na.omit(data.frame(x_trees, y_error))

ggplot()+
  ggtitle(" Error for Adaboost ")+
  geom_point(data=ada_m, aes(x=x_trees, y=y_error), size=2)+
  geom_line(data=ada_m, aes(x=x_trees, y=y_error), size=1)


#train data
```

```r
fitted.results_train = predict(ada_model,newdata=train)
fitted.results_train = ifelse(fitted.results_train > 0.1,1,0)
misClasificError_train = mean(fitted.results_train != train$Spam)
ada_confmat_train = table("Y"=train$Spam,"Y hat"=fitted.results_train)
ada_confmat_train
print(paste('Accuracy:',sum(diag(ada_confmat_train)) / sum(ada_confmat_train)))

#test data
fitted.results_test= predict(ada_model,newdata=test)
fitted.results_test = ifelse(fitted.results_test > 0.1,1,0)
misClasificError_test = mean(fitted.results_test != test$Spam)
ada_confmat_test = table("Y"=test$Spam,"Y hat"=fitted.results_test)
print(paste('Accuracy:',sum(diag(ada_confmat_test)) / sum(ada_confmat_test)))
#print(paste('Accuracy:',1-misClasificError_test))

## Random Forest ##

#train
rf_model_train = randomForest(Spam ~ ., data = train, ntree = 100)

rf_model_test = randomForest(Spam ~ ., data = test, ntree = 100)

plot(rf_model_train)
plot(rf_model_test)
## The plot above seems to show that the error decreases linealy until about
## 30 trees then stays relatively stable at around 0.05 as #trees grows.

#test data
fitted.results_test_rf = predict(rf_model, test)

rf_confmat_test = table("Y"=test$Spam,"Y hat"=fitted.results_test_rf)
print(paste('Error:',1-sum(diag(rf_confmat_test)) / sum(rf_confmat_test)))
error = 0
ntree= seq(10,100,10)
for(i in ntree){

  rf_model = randomForest(Spam ~ ., data = train, ntree = i)
  fitted.results_test_rf = predict(rf_model, test)

  rf_confmat_test = table("Y"=test$Spam,"Y hat"=fitted.results_test_rf)
 error[i] = (sum(diag(rf_confmat_test)) / sum(rf_confmat_test))

}
error_rf = na.omit(error)


y_error = error_rf
x_trees <- ntree
rf_m <- na.omit(data.frame(x_trees, y_error))

ggplot()+
  ggtitle(" Error for Random Forest ")+
  geom_point(data=rf_m, aes(x=x_trees, y=y_error), size=2)+
```

```
  geom_line(data=rf_m, aes(x=x_trees, y=y_error), size=1)


## EM Algorithm ##

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)
plot(true_mu[1,],
     type="o",
     col="blue",
     ylim=c(0,1),
     main = "Original Data",
     xlab = "Number of Dimensions",
     ylab = "True Mu")
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
# 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])
  }
}
em_algorithm <- function(c){
  K=c # 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)
  }
  pi
  mu
  for(it in 1:max_it) {
    #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="yellow")
```

```r
    #Sys.sleep(0.5)
    # E-step: Computation of the fractional component assignments (responsiblities)
    # Your code here
    for (n in 1:N){
      phi = c()
      for (j in 1:K){
        y1 = mu[j,]^x[n,]
        y2 = (1- mu[j,])^(1-x[n,])
        phi = c(phi, prod(y1,y2))
      }

      z[n,] = (pi*phi) / sum(pi*phi)
    }
    #Log likelihood computation.
    # Your code here
    likelihood = matrix(0,1000,K)
    llik[it] = 0
    for(n in 1:N){
      for (k in 1:K){
        likelihood[n,k] = pi[k]*prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
      }
      llik[it] = sum(log(rowSums(likelihood)))
    }
    cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    flush.console()
    # Stop if the lok likelihood has not changed significantly
    # Your code here
    if (it > 1){
      if (llik[it]-llik[it-1] < min_change){
        if(K == 2){
          plot(mu[1,],
               type="o",
               col="blue",
               ylim=c(0,1),
               main = "K = 2",
               xlab = "Number of Dimensions",
               ylab = "Estimated Mu")
          points(mu[2,], type="o", col="red")
        }
        else if(K == 3){
          plot(mu[1,],
               type="o",
               col="blue",
               ylim=c(0,1),
               main = "K = 3",
               xlab = "Number of Dimensions",
               ylab = "Estimated Mu")
          points(mu[2,], type="o", col="red")
          points(mu[3,], type="o", col="green")
        }
        else if(K == 4){
          plot(mu[1,],
               type="o",
```

```r
               col="blue",
               ylim=c(0,1),
               main = "K = 4",
               xlab = "Number of Dimensions",
               ylab = "Estimated Mu")
          points(mu[2,], type="o", col="red")
          points(mu[3,], type="o", col="green")
          points(mu[4,], type="o", col="yellow")
        }
        break()
      }
    }
    #M-step: ML parameter estimation from the data and fractional component assignments
    # Your code here
    mu = (t(z) %*% x) /colSums(z)
    # N - Total no. of observations
    pi = colSums(z)/N
  }
  pi
  mu
  # plot(llik[1:it],
  #      type="o",
  #      main = "Log Likelihood",
  #      xlab = "Number of Iterations",
  #      ylab = "Log Likelihood")
}

em_algorithm(2)
em_algorithm(3)
em_algorithm(4)
```