Ensemble Methods and Mixture Models

Maximilian Pfundstein (maxpf364)

27 November 2018

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.

Table 1: 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 following 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.

```
# 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))

}
```

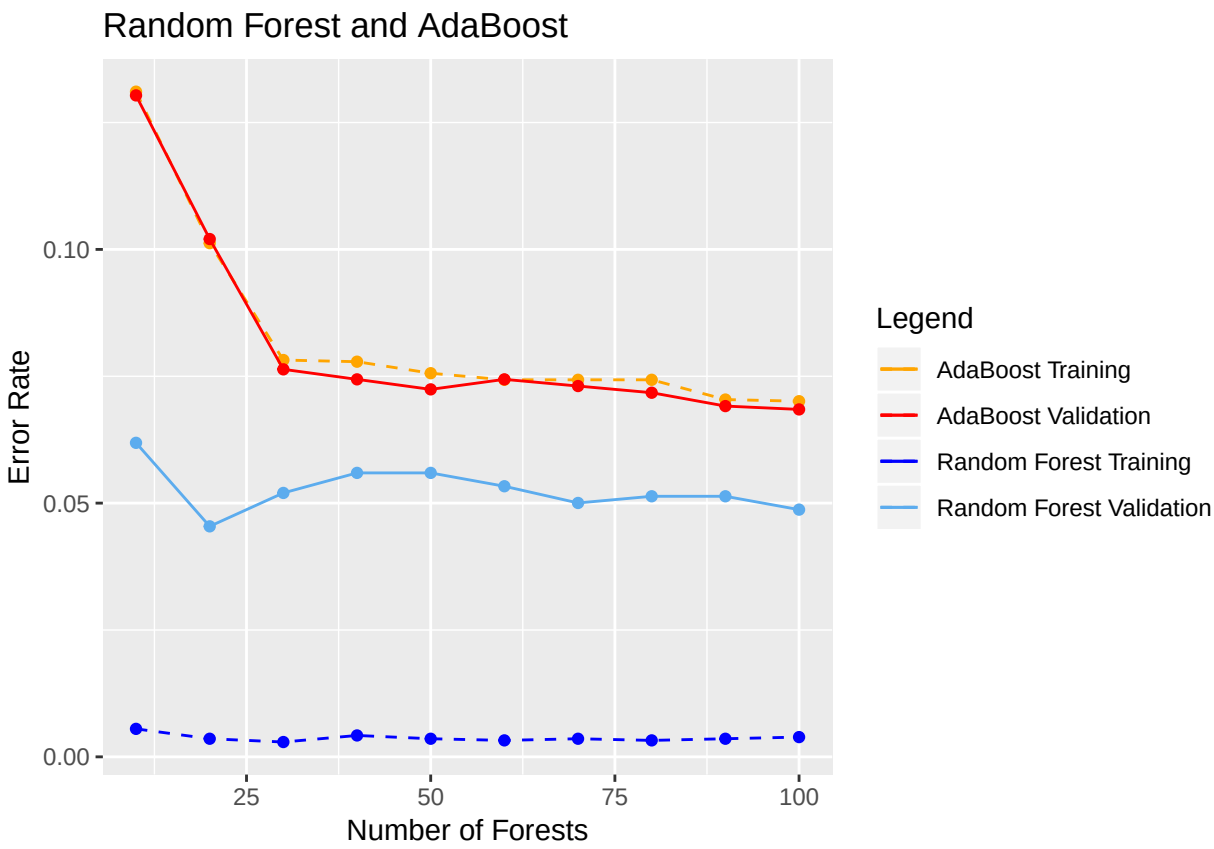
The following tables show the error rates for Random Forst and AdaBoost. The plot visualizes this data, the dashed lines represent the performance on the training data set.

Table 2: 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

Table 3: 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 Z matrix, the likelihood L and the new μ and π 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 Z, L, μ, π and X mean.

Values axplaination

We have X which holds $N = 1000$ coin tosses where the coin was tossed D times. 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 π holds our estimate how many datapoints from X belong to K_i (which is 1/3 for each, but let's suppose we don't know.)

μ represents the probability for each D to be head or tail for each K_i.

The likelihood L 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 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. 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 min_change is set to 0.1 (note that we speak about the ln-likelihood).

Z holds the probability for each set of coin tosses (row in X) to belong to each K_i. 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 Z. We will divide that in first calculating Bx which holds $Bernoulli(x|\mu_k)$ and afterwards calculating $p(x)$ which we can use to calculate Z. The formales will reduce the left side to single letters which you will find in the source code.

$$Bernoulli(x|\mu_k) = B_x = \prod_i \mu_{k_i}^{x_i} * (1 - \mu_{k_i})^{1-x_i}$$

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

$$\ln(B) = \sum \ln(\mu_{k_i}^{x_i}) * x_i + \sum \ln(1 - \mu_{x_i}) * (1 - x_i)$$

Now let's get rid of the ln on the left side:

$$B_x = e^{\sum \ln(\mu_{k_i}^{x_i}) * x_i + \sum \ln(1-\mu_{x_i}) * (1-x_i)}$$

This craves to be put into a neat matrix multiplication. Okay, let's look for $p(x)$.

$$p(x) = P_x = \sum_k \pi_k * Bernoulli(x|\mu_k) = \sum_k \pi_k * B$$

We will use $P(x)$ later to calculate the likelihood L but for now let's calculate Z:

$$P(z_{nk}|x_n, \mu, \pi) = Z = \frac{\pi_k * p(x_n|\mu_k)}{\sum_k * p(x_n|\mu_k)}$$

The likelihood L is given by the following equation which can be found in (Bishop 2006) on page 443 equation 9.14.

$$\ln p(X|\pi, \mu, \Sigma) = L = \sum_{n=1}^{N} ln\{\sum_{k=1}^{K} \pi_k * N(x_n|\mu_k, \Sigma_k)\}$$

As we already have the value inside the curly braces ($P(x)$) it's basically just the sum of the logarithms over n.

For calculating π we use the following.

$$\pi_k^{ML} = pi = \frac{\sum_k p(z_{nk|x_n|\mu|\pi})}{N}$$

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

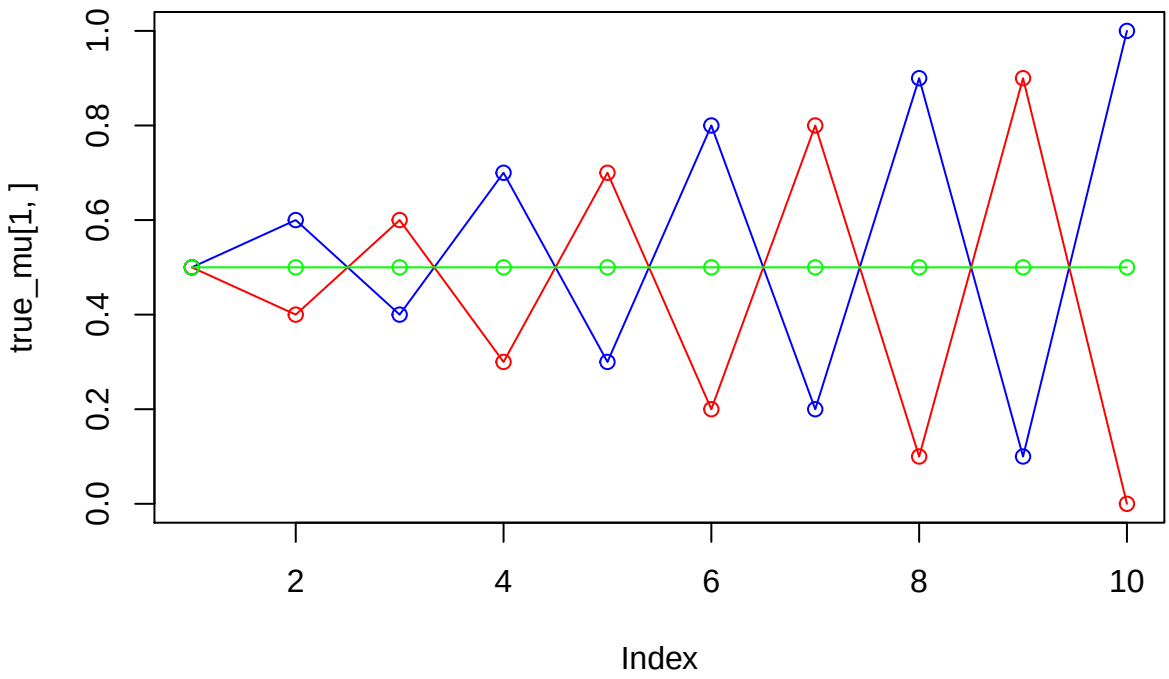
$$\mu_k^{ML} = mu = \frac{\pi_k p(z_{nk|x_n|\mu|\pi})}{\sum_k \pi_k p(z_{nk|x_n|\mu|\pi})}$$

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 μ and π

These are the true values for μ and π:




```
## [1] 0.3333333 0.3333333 0.3333333
```

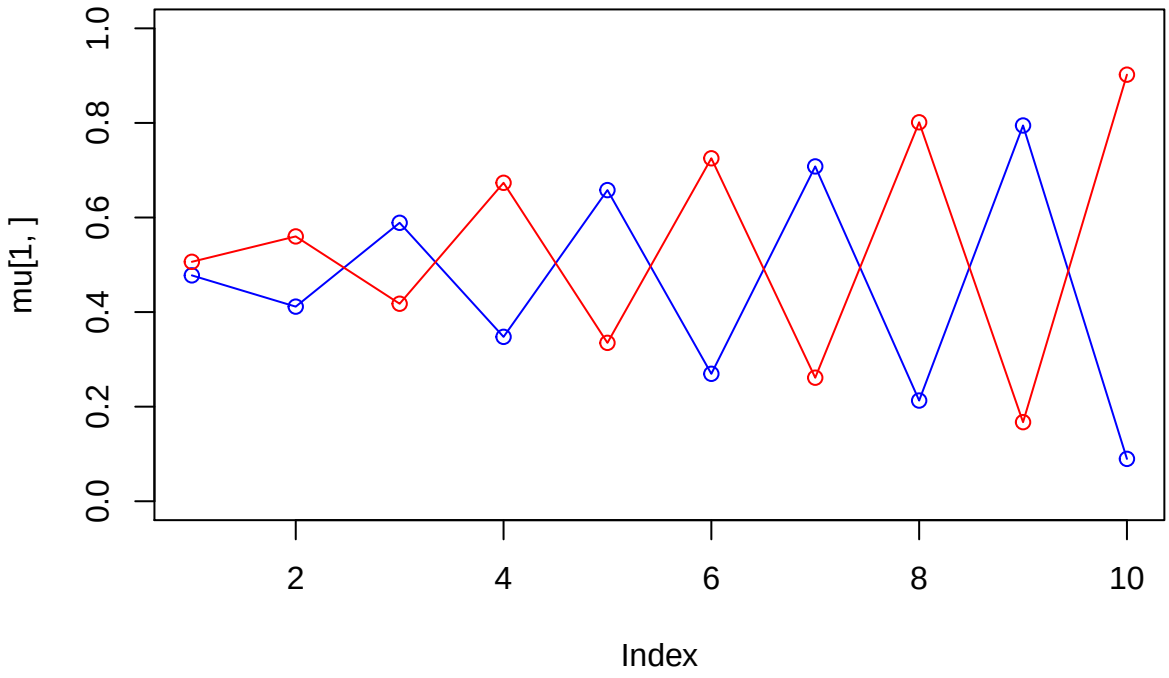
```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  0.5  0.6  0.4  0.7  0.3  0.8  0.2  0.9  0.1   1.0
## [2,]  0.5  0.4  0.6  0.3  0.7  0.2  0.8  0.1  0.9   0.0
## [3,]  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5   0.5
```

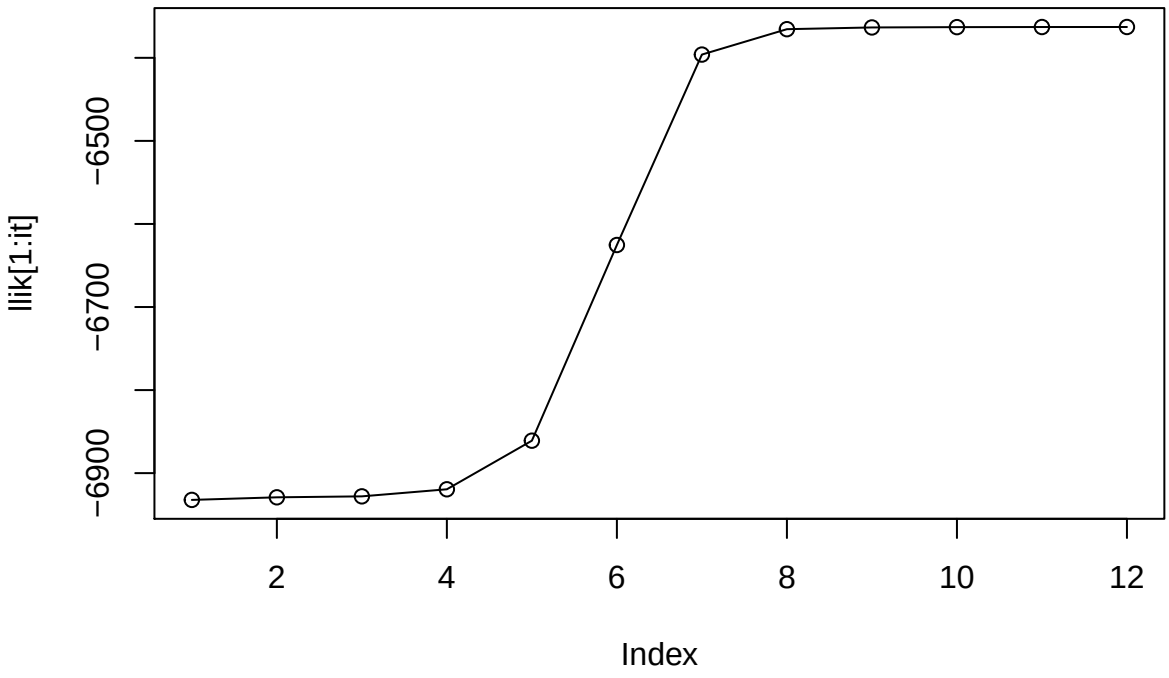
K = 2

Table 4: Pi

x
0.4985558
0.5014442

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

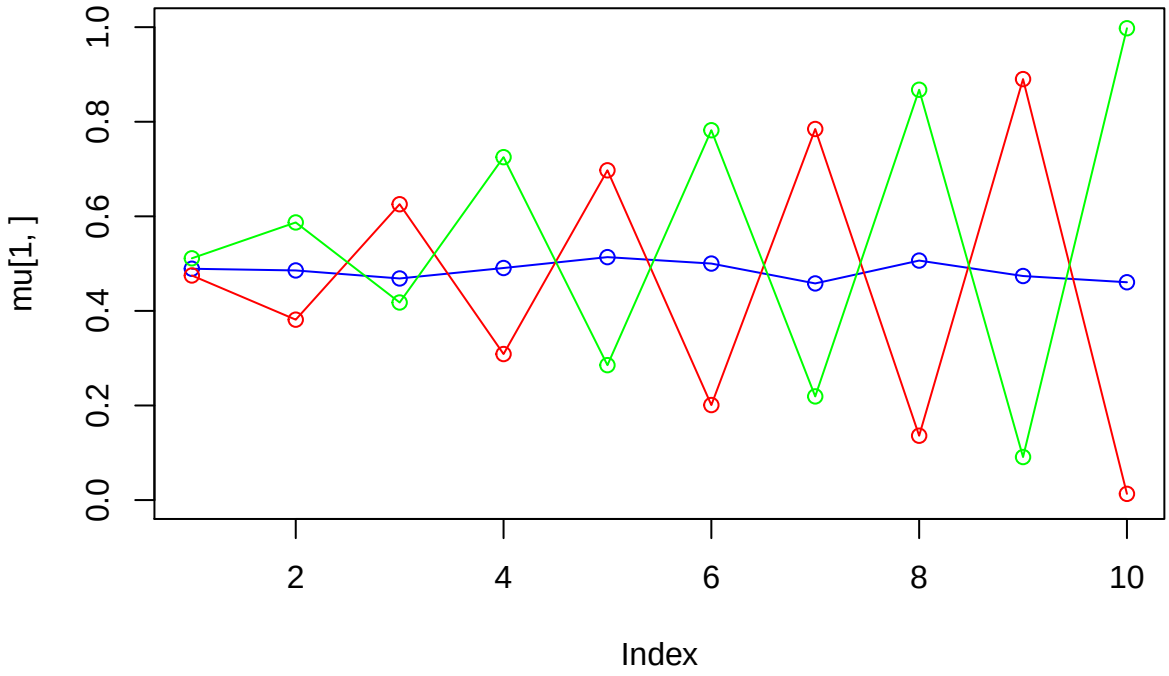

mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

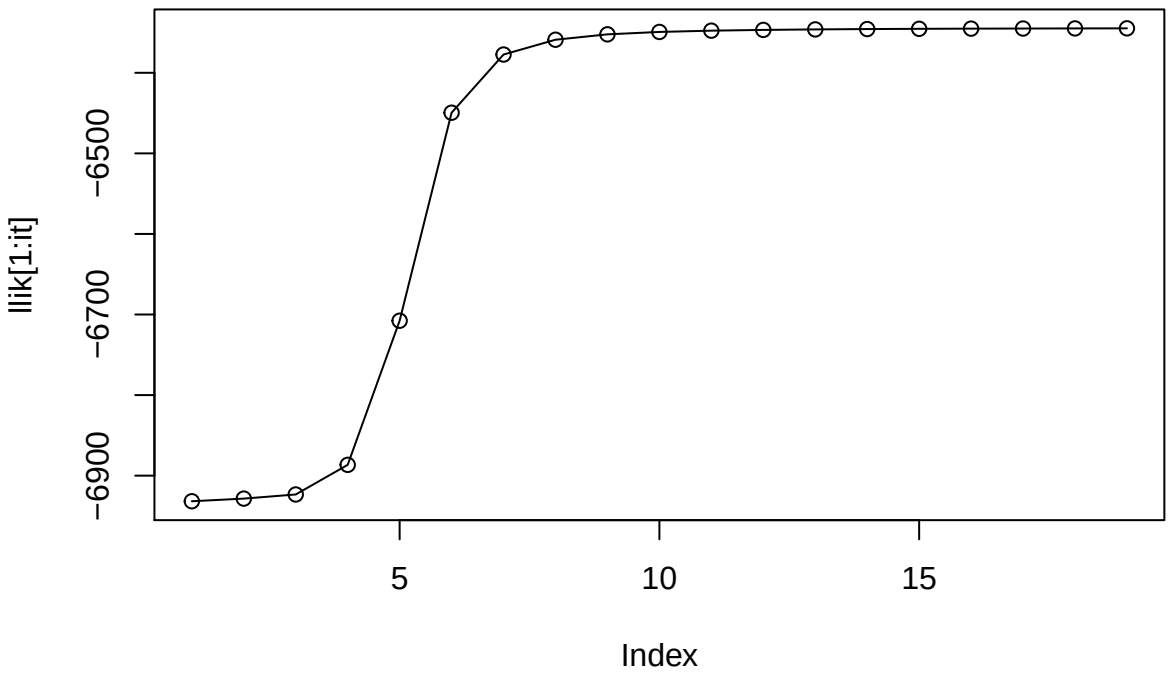

llik[1:it]
−6900
−6700
−6500
2
4
6
8
10
12
Index

K = 3

```
## [1] 0.3272335 0.3300263 0.3427403
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4891351 0.4855618 0.4686094 0.4907012 0.5136872 0.5001229 0.4581237
## [2,] 0.4748950 0.3814730 0.6255717 0.3087514 0.6973317 0.2008993 0.7846864
## [3,] 0.5112058 0.5870679 0.4178097 0.7251266 0.2852498 0.7820528 0.2191731
##           [,8]       [,9]      [,10]
## [1,] 0.5064250 0.47377097 0.46056120
## [2,] 0.1360473 0.89018026 0.01313653
## [3,] 0.8676588 0.09098265 0.99770466
```

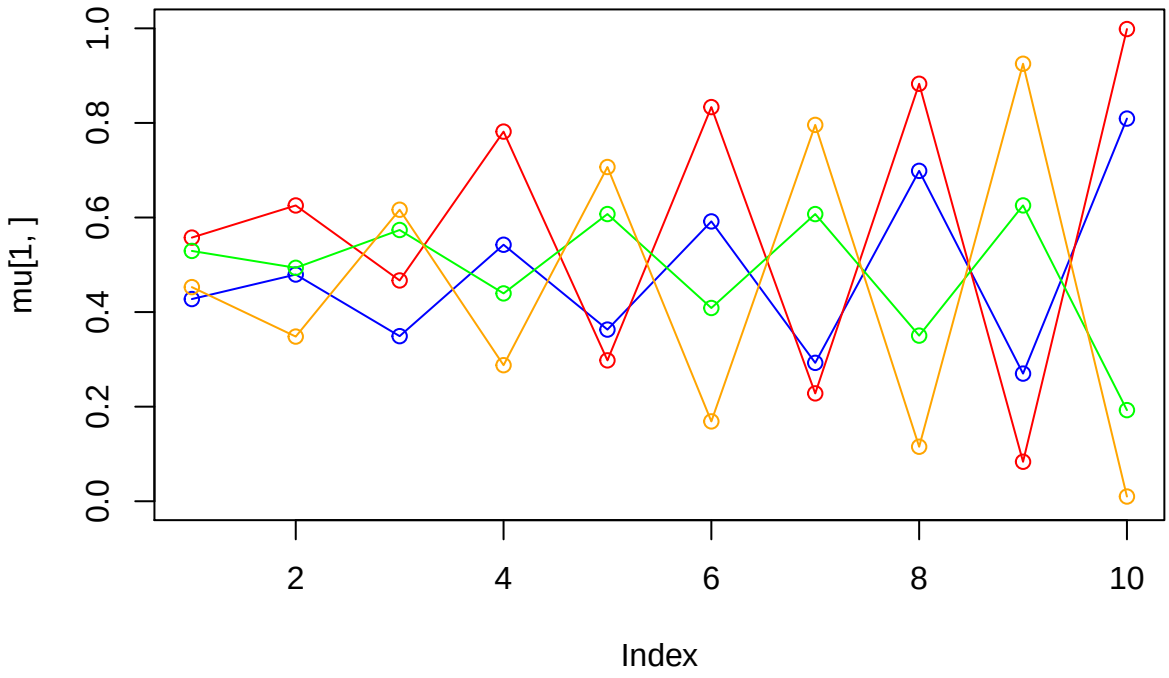


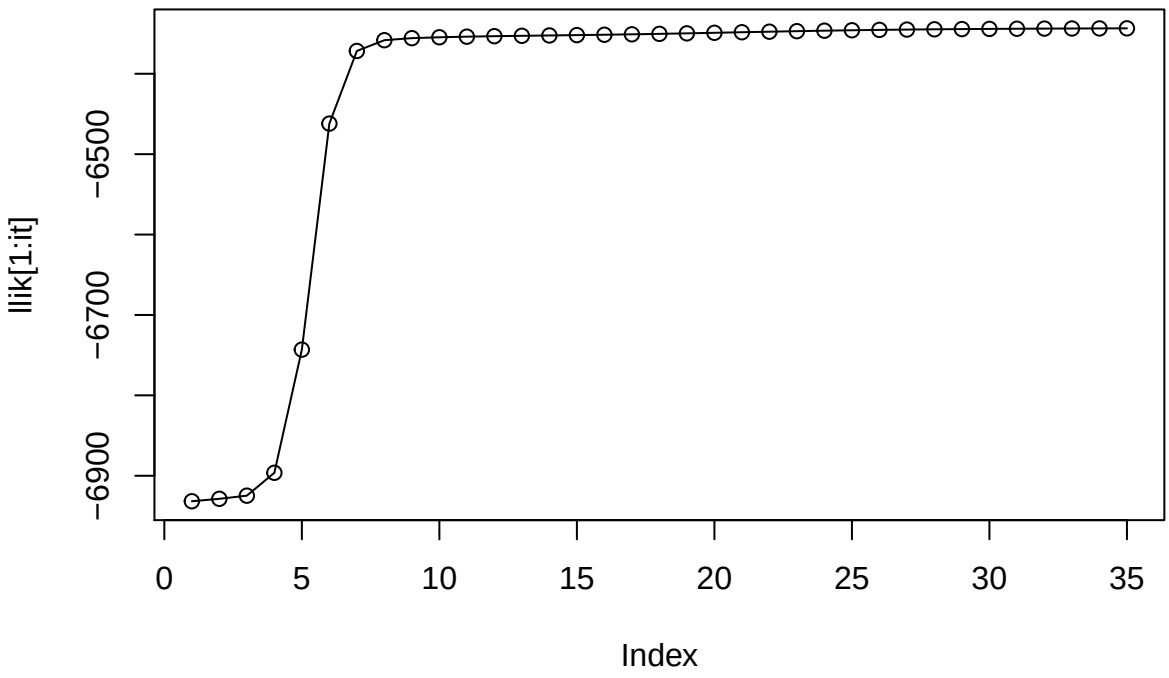


K = 4

```
## [1] 0.2451226 0.2474470 0.2545552 0.2528752
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4275214 0.4794378 0.3490801 0.5424050 0.3629693 0.5917624 0.2926573
## [2,] 0.5575234 0.6254826 0.4669151 0.7816688 0.2979063 0.8334707 0.2280754
## [3,] 0.5293161 0.4936496 0.5735814 0.4394978 0.6072886 0.4086347 0.6072466
## [4,] 0.4528209 0.3481721 0.6164610 0.2876763 0.7067658 0.1688016 0.7958422
##           [,8]       [,9]       [,10]
## [1,] 0.6986396 0.26989755 0.809218119
## [2,] 0.8828056 0.08348568 0.998501365
## [3,] 0.3502206 0.62559006 0.192665069
## [4,] 0.1152724 0.92510672 0.009974357
```


mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index



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)
```

```r
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")

true_pi
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)
}
```

```r
kable(pi, caption = "Pi")
kable(mu)
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)
}


pi
mu
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)
}


pi
mu
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.