Applied Statistical Programming - The EM Algorithm

Amaan, Alex, Patrick, Peter

4/13/2022

Write R and Rcpp code to answer the following questions. Write the code, and then show what the computer returns when that code is run. Thoroughly comment your solutions.

Complete this assignment before 10:00am on Wednesday, April 20. Submit the R implementation as an Rmarkdown and the knitted PDF to Canvas. Have one group member submit the activity with all group members listed at the top. The Rcpp portion will be given to you as your final assignment.

In-class Background: The Expectation-Maximization Algorithm

The goal of this in-class exercise is to implement an ensemble of models. You will combine forecasts of US presidential elections using ensemble Bayesian model averaging (EBMA). To do this, you must decide how to weight each component of the forecast in the prediction. The collection of these weighted forecasts form the ensemble, and you will use something called the EM (expectation-maximization) algorithm.

The task is to choose values w_k that maximize the following equation:

$$p(y|f_1^{s|t^\star}, ..., f_K^{s|t^\star}) = \sum_{k=1}^{N} w_k N(f_k^{t^\star}, \sigma^2) \tag{1}$$

For the remainder of this assignment, assume that the parameter σ^2 is known and that $\sigma^2 = 1$.

The first step of the EM algorithm is to estimate the latent quantity $\hat{z}_k^t$ that represents the probability that observation t was best predicted by model k.

$$\hat{z}_k^{(j+1)t} = \frac{\hat{w}_k^{(j)} N(y^t|f_k^t, 1)}{\sum_{k=1}^{N} \hat{w}_k^{(j)} N(y^t|f_k^t, 1)} \tag{2}$$

In this equation, j is the particular iteration of the EM algorithm, and $N(y^t|f_k^t, 1)$ is the normal cumulative distribution function evaluated at the observed election outcome (`dnorm(y, ftk, 1)`).

The second step of the EM algorithm is to estimate the expected value of the weights assuming that all $\hat{z}_k^t$ are correct.

$$\hat{w}_k^{(j+1)} = \frac{1}{n} \sum_t \hat{z}_k^{(j+1)t} \tag{3}$$

The estimation procedure is as follows:

1. Start with the assumption that all models are weighted equally.
2. Calculate $\hat{z}_k^{(j+1)t}$ for each model for each election.
3. Calculate $\hat{w}_k^{(j+1)}$ for each model.
4. Repeat steps 2-3 twenty times.

Complete the preceding tasks in `R` alone.

ANSWERS: R Section.

FINISHED EM ALGORITHM AT BOTTOM OF R CODE SECTION

- we use the Expectation-Maximization (EM) algorithm on datasets that combine multiple linear models. In this case, we combine different forecasts of US presidential elections using EBMA.
- To learn more about EBMA forecasts and (hopefully) find some toy data to use with our functions, let's install and investigate the `EBMAforecast` package.

```
# FINISHED EM ALGORITHM AT BOTTOM OF R CODE SECTION.

## Install package:

# install.packages('EBMAforecast')
library(EBMAforecast)

## Read documentation:

# ?EBMAforecast

## Find data used in demo(PresForecast):

# demo(presForecast)
```

- Let's load the `presidentialForecast` data from the `EBMAforecast` package:

```
# FINISHED EM ALGORITHM AT BOTTOM OF SECTION.

## Load `presidentialForecast` data from `EBMAforecast` package:
data("presidentialForecast")
# ?presidentialForecast

df <- EBMAforecast::presidentialForecast

## OUTCOME VARIABLE: incumbent-party vote share in each presidential election.
## In this dataset, the actual outcome is the column `Actual`.
y <- df$Actual

# Now that the outcome variable `y` is another object, remove column `Actual`
# from `df`, which now represents the matrix of models (analogous to matrix of
# X's)
```

SECOND: implement steps 1-3 above.

For the remaining R code, let's first develop code for each of the steps referenced above before properly structuring them in a for-loop.

1. Start with the assumption that all models are weighted equally.

NOTE: recall from Montgomery, Hollenbach, and Ward (2012) that the $w_k \in [0, 1]$'s are model probabilities associated with each component model's predictive performance. In other words, the $w_k \in [0, 1]$'s are weights associated each each model such that $\sum_{k=1}^{K} w_k = 1$.

ASSUMED DATA STRUCTURE:

- dataframe `df` with the following dimensions:
  - `K+1` columns, which include `K` columns of models and `1` column of outcomes (i.e., the actual results of one presidential elections).
  - `T` rows of predictions (substantively, these are the predicted incumbent-party vote share for each presidential election).
- Vector `y` with `T` elements.
  - Each element $t \in T$ of `y` represents the actual results of one presidential election.

```
# SKIP, FINISHED EM ALGORITHM AT BOTTOM OF SECTION.

## Find number of models:
K <- dim(df)[2] - 1

## Find the number of observations (in this case, presidential elections):
T <- dim(df)[1]

## Define w_hat with all models weighted equally.
w_hat <- replicate(K, 1/K)

## COMPLETE!
```

2. Calculate $\hat{z}_k^{(j+1)t}$ for each model for each election.

NOTE: the function $\hat{z}_k^{(j+1)t} = \frac{\hat{w}_k^{(j)} N(y^t | f_k^t, 1)}{\sum_{k=1}^{N} \hat{w}_k^{(j)} N(y^t | f_k^t, 1)}$ is complex, containing several unexplained terms. Let's consider each of these terms:

- Recall from above that `j` represents the prior iteration of the EM algorithm, so the EM algorithm is recursive.
- Also recall from above that $\hat{z}_k^t$ is the probability that observation `t` (i.e., a particular presidential election; one row in the test dataset) was best predicted by model `k` (i.e., one column in the test dataset).
- Also recall from above that $\hat{w}_k^{(j)}$ is the probability weights for each model `k`. We assume that all $\hat{w}_k^{(j)}$'s in the first iteration.
- y^t is the *actual* election result of presidential election `t` (i.e., row `t`'s entry of column `Actual` in the test dataset).
- f_k^t is model `k`'s *predicted* election result for presidential election `t` (i.e., row `t`'s entry of column associated with model `k`).

- $N(y^t|f_k^t, 1)$, then, is the probability that model `k` found predicted outcome f_k^t for presidential election `t` given that the sample distribution is normally distributed around the actual outcome y^t with st. dev. $\sigma^2 = 1$ Generate $N(y^t|f_k^t, 1)$ using `norm(y, ftk, 1)` where `y` $= y^t$ and `ftk` $= f_k^t$.
- Finally, note that the outcome of this step will be a vector because the numerator $\hat{w}_k^{(j)} N(y^t|f_k^t, 1)$ is a vector of `K` elements for EACH model's `w_hat`, while the denominator $\sum_{k=1}^{N} \hat{w}_k^{(j)} N(y^t|f_k^t, 1)$ is a scalar that's the sum of the numerator for each model `k`.

With this knowledge in mind, let's create code that estimates $\hat{z}_k^{(j+1)t}$:

```
# SKIP, FINISHED EM ALGORITHM AT BOTTOM OF SECTION.

## NOTE 1: `y = df$Actual`.

## NOTE 2: `k` denotes the current model (column `k` of `K`).

## NOTE 3: `t` denotes the current prior presidential election (row `t` of `T`).

## NOTE 4: input `w_hat` and output `z_hat` are vectors of `K` elements.

z_hat <- (w_hat * dnorm(y[t], df[t,k], 1))/(sum(w_hat * dnorm(y[t], df[t,k], 1)))

# COMPLETE!
```

3. Calculate $\hat{w}_k^{(j+1)}$ for each model.

Whereas step (2) estimated $\hat{z}_k^{(j+1)t}$ (the probability that model `k` best predicted the outcome of ONE presidential election `t`), step (3) determines the next iteration's $\hat{w}_k^{(j+1)}$ (i.e., the weight associated with model `k`) by finding the average $\hat{z}_k^{(j+1)t}$ over ALL presidential elections (i.e., $\forall t \in T$).

```
# SKIP, FINISHED EM ALGORITHM AT BOTTOM OF SECTION.

## NOTE 1: input `z_hat` and output `w_hat` are vectors of `K` elements.

## NOTE 2: `T = dim(df)[1]`, or the number of all presidential elections in sample.
w_hat <- sum(z_hat)/T

# COMPLETE!
```

THIRD: perform step (4).

perform step (4) above by creating a for-loop that repeats steps 2-3 twenty times.

This step pulls everything together. In brief, there will be one for-loop containing the two steps from above:

- **step (2)**: find `T`-by-`K` matrix of `T` vectors containing `K` elements $\hat{z}_k^{(j+1)t}$ for each election (i.e., for every row in `df`). In other words, each row in the matrix is a $\hat{z}_k^{(j+1)t}$.
- **step (3)**: find `K`-length vector $\hat{w}_k^{(j+1)}$ for each model (i.e., for every column in `df`). In brief, we average each column of the matrix from step (2). The result is the new $\hat{w}_k^{(j+1)}$ weights for each model.

```
# SKIP, FINISHED EM ALGORITHM AT BOTTOM OF SECTION.

# ANOTHER DEBUGGING METHOD: manually run each iteration and see what happens.
# SUCCESS! Correction added to final model below.
```

```r
df <- EBMAforecast::presidentialForecast # Reload data.

# PRELIMINARY 1: Number of models:
K <- dim(df)[2] - 1

# PRELIMINARY 2: Number of observations:
Ti <- dim(df)[1]

# PRELIMINARY 3: Create function that combines steps (2) & (3):
steps23 <- function(df = df, w_hat = wHat_it[i-1,]){
  y <- df[[7]] # Create vector of outcome variables.
  M <- df[,1:K] # create matrix of models.
  K <- dim(df)[2] - 1 # Number of models.
  Ti <- dim(df)[1] #Number of observations
  # Create dnorm object for step 2:
  dnormal <- t(
    sapply(
      X = 1:Ti,
      FUN = function(t){
        dnorm(
          y[t],
          sapply(
            X = 1:K,
            FUN = function(k){df[t,k]}
            ),
          sd = 1
          )
        }
      )
    )
  # Do step 2 function:
  z_hats <- t( # CORRECTION! Turned z_hats back into Ti X K matrix.
    sapply(
      X = 1:Ti,
      FUN = function(t){
        (w_hat * dnormal[t,])/(sum(w_hat * dnormal[t,]))
        }
      )
    )
  # Do step 3 function:
  w_hat2 <- sapply(
    X = 1:K,
    FUN = function(k){
      sum(z_hats[,k])/Ti
    }
  )
  return(w_hat2)
}

# PRELIMINARY 4: create matrix to collect iterations data. It needs to have 21
# rows for the initial w_hat and the 20 iterations of w_hat.
itnum <- 20 # Number of iterations
wHat_it <- matrix(
```

```
  data = NA,
  nrow = itnum + 1,
  ncol = K
  )

# PRELIMINARY 4.5: create vector to check if each iteration sums to 1.
sum_wHat <- rep(NA, 21)
length(sum_wHat) == dim(wHat_it)[1]

# STEP 1: Define w_hat with all models weighted equally.
wHat_it[1,] <- replicate(K,1/K)

sum_wHat[1] <- sum(wHat_it[1,])
sum_wHat

# ITERATION 1:
wHat_it[2,] <- steps23(df = df, w_hat = wHat_it[1,])
wHat_it

sum_wHat[2] <- sum(wHat_it[2,])
sum_wHat # STILL SUMS TO 1!!

# ITERATION 2:
wHat_it[3,] <- steps23(df = df, w_hat = wHat_it[2,])
wHat_it

sum_wHat[3] <- sum(wHat_it[3,])
sum_wHat # STILL SUMS TO 1!!


# IT'S WORKING!!!!
```

```
# FINISHED! WORKING EM ALGORITHM!!  BRINGING IT ALL TOGETHER!

# Reload data:
df <- EBMAforecast::presidentialForecast

# PRELIMINARY 1: Number of models:
K <- dim(df)[2] - 1

# PRELIMINARY 2: Number of observations:
Ti <- dim(df)[1]

# PRELIMINARY 3: Create function that combines steps (2) & (3):
steps23 <- function(df = df, w_hat = wHat_it[i - 1, ]) {
    y <- df[[7]]  # Create vector of outcome variables.
    M <- df[, 1:K]  # create matrix of models.
    K <- dim(df)[2] - 1  # Number of models.
    Ti <- dim(df)[1]  #Number of observations
    # Create dnorm object for step 2:
    dnormal <- t(sapply(X = 1:Ti, FUN = function(t) {
        dnorm(y[t], sapply(X = 1:K, FUN = function(k) {
            df[t, k]
        }), sd = 1)
```

```
    }))
    # Do step 2 function:
    z_hats <- t(sapply(X = 1:Ti, FUN = function(t) {
        (w_hat * dnormal[t, ])/(sum(w_hat * dnormal[t, ]))
    }))
    # Do step 3 function:
    w_hat2 <- sapply(X = 1:K, FUN = function(k) {
        sum(z_hats[, k])/Ti
    })
    return(w_hat2)
}

# PRELIMINARY 4: create matrix to collect iterations data. It needs to have 21
# rows for the initial w_hat and the 20 iterations of w_hat.
itnum <- 20  # Number of iterations
wHat_it <- matrix(data = NA, nrow = itnum + 1, ncol = K)
colnames(wHat_it) <- c("Campbell", "Lewis-Beck", "EWT2C2", "Fair", "Hibbs", "Abramowitz")
rownames(wHat_it) <- 0:itnum

# STEP 1: Define w_hat with all models weighted equally.
wHat_it[1, ] <- replicate(K, 1/K)

# Steps 2-4 (ITERATION):
i <- 2
while (i <= itnum + 1) {
    wHat_it[i, ] <- steps23(df = df, w_hat = wHat_it[i - 1, ])
    print(i)
    i = i + 1
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
```

```
# FINAL ITERATION:
wHat_it[tail(itnum, 1), ]
```

```
##    Campbell  Lewis-Beck      EWT2C2        Fair       Hibbs  Abramowitz
```

```
## 0.317363878 0.116196147 0.234058800 0.009787069 0.287805926 0.034788180
```

Assignment: Rcpp Practice

1. Write an Rcpp function that will calculate the answer to Equation (2). The output will be a matrix.
2. Write an Rcpp function that will calculate the answer to Equation (3). The output will be a vector.
3. Write an Rcpp function that will complete the entire algorithm.

```
library(Rcpp)

# EQUATION 2 ------------------------------------------------------------------
cppFunction('NumericMatrix rCppStep2(NumericMatrix x, NumericVector y, NumericVector weights, double sd
  int rows = x.nrow();
  NumericMatrix dNormal(x);
  NumericMatrix out(x);

  for (int i = 0; i < x.nrow(); ++i) {
    for (int j = 0; j < x.ncol(); ++j) {
      double datNumber = R::dnorm(y[i], x(i,j), sd, FALSE);
      dNormal(i,j) =  datNumber; //
    }
  }

  NumericVector sums(rows);
  for (int i = 0; i < rows; ++i) {

    double rowSum = 0;
    for (int j = 0; j < dNormal.ncol(); ++j) {
      rowSum += weights[j] * dNormal(i,j);
    }
  sums[i] = rowSum;
  }

  for (int i = 0; i < x.nrow(); ++i) {
    for (int j = 0; j < x.ncol(); ++j) {
      out(i,j) =  weights[j] * dNormal(i,j) / sums[i]; //
    }
  }

  return out;
}')

# EQUATION 3 ------------------------------------------------------------------
cppFunction('NumericVector rCppStep3(NumericMatrix zHat) {
  int rows = zHat.nrow();
  int cols = zHat.ncol();
  NumericVector out(cols);

  for (int j = 0; j < zHat.ncol(); ++j) {

    double newWeight = 0;
    for (int i = 0; i < zHat.nrow(); ++i) {
      newWeight +=  zHat(i,j);
```

```
    }
  out[j] = newWeight / rows;
  }
  return out;
}')

# WHOLE ALGORITHM ---------------------------------------------------------------
src <- "#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]

NumericMatrix rCppStep2(NumericMatrix x, NumericVector y, NumericVector weights, double sd){
  int rows = x.nrow();
  NumericMatrix dNormal(x);
  NumericMatrix out(x);

  for (int i = 0; i < x.nrow(); ++i) {
    for (int j = 0; j < x.ncol(); ++j) {
      double datNumber = R::dnorm(y[i], x(i,j), sd, FALSE);
      dNormal(i,j) =  datNumber; //
    }
  }

  NumericVector sums(rows);
  for (int i = 0; i < rows; ++i) {

    double rowSum = 0;
    for (int j = 0; j < dNormal.ncol(); ++j) {
      rowSum += weights[j] * dNormal(i,j);
    }
  sums[i] = rowSum;
  }

  for (int i = 0; i < x.nrow(); ++i) {
    for (int j = 0; j < x.ncol(); ++j) {
      out(i,j) =  weights[j] * dNormal(i,j) / sums[i]; //
    }
  }

  return out;
}

// [[Rcpp::export]]
NumericVector rCppStep3(NumericMatrix zHat) {
  int rows = zHat.nrow();
  int cols = zHat.ncol();
  NumericVector out(cols);

  for (int j = 0; j < zHat.ncol(); ++j) {

    double newWeight = 0;
    for (int i = 0; i < zHat.nrow(); ++i) {
      newWeight +=  zHat(i,j);
```

```
    }
  out[j] = newWeight / rows;
  }
  return out;
}

// [[Rcpp::export]]
NumericVector rCppEBMA(NumericMatrix x, NumericVector y, NumericVector weights, double sd, int iteratio
NumericMatrix w((iterations + 1), x.ncol());
w( 0 , _ ) = weights;

for (int i = 0; i < iterations; ++i){
  w((i + 1), _ ) = rCppStep3(rCppStep2(x, y, w(i, _ ), sd));
}

return w( iterations , _ );
}"

sourceCpp(code = src)

df <- EBMAforecast::presidentialForecast
x <- df[,1:6]
y <- df[,7]
wHat <- replicate(dim(x)[2],1/dim(x)[2])

zHat <- rCppStep2(as.matrix(x), y, wHat, 1)
weights <- rCppStep3(zHat)

# This doesn't work past 1 iteration and I have no clue why and at this point
# I'm too frustrated to find out why. Not sure how we were expected to implement
# this all in rcpp considering we didn't go over matrices in rcpp or how to
# overwrite objects in rcpp (which I'm assuming we can't) I'm just throwing this
# in a loop and calling it good.
#
# I'm curious on how this could have been implemented. Please let me know
# -- Peter
rCppEBMA(as.matrix(x), y, wHat, 1, 2)
```

```
## [1] NaN NaN NaN NaN NaN NaN
```

```
for (i in 1:20) {
  zHat <- rCppStep2(as.matrix(x), y, wHat, 1)
  wHat <- rCppStep3(zHat)
}
wHat
```

```
## [1] 0.317273718 0.115254756 0.235069745 0.008694366 0.291233448 0.032473966
```