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.

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

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.

FIRST: find test data to use on our model.

- 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.

```
## Install package:

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

```
## Warning: package 'EBMAforecast' was built under R version 4.1.3
```

```
## Read documentation:

# ?EBMAforecast

## Find data used in demo(PresForecast):

# demo(presForecast)
```

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

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

df <- 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) df <- df[,-7] DISREGARD! NOT DOING FOR NOW!
```

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.

```r
## 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 $\hat{z}_k^{(j+1)t} = \frac{\hat{w}_k^{(j)} N(y^t|f_k^t,1)}{\sum\limits_{k=1}^{N} \hat{w}_k^{(j)} N(y^t|f_k^t,1)}$ 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 $\hat{z}_k^{(j+1)t}$:

```
## 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., $\forall t \in T$).

```
## 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 `K`-length vector $\hat{z}_k^{(j+1)t}$ for each election (i.e., for every row in `df`).
- **step (3)**: find `K`-length vector $\hat{w}_k^{(j+1)}$ for each model (i.e., for every column in `df`)

```r
# WORK IN PROGRESS!

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

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

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

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

# Steps 2-4:

sapply(
```

```r
  X = 1:20,
  FUN =
)
```

```r
# Eventually wrap everything in a loop, or apply function?

# Calculate zHat

# I'm not sure if the indices are in the right places. This is mostly because
# I'm not entirely sure what's going on
zHat[i + 1] <- (wHat[i] * dnorm(y[i], ftk[i], 1)) / sum(wHat * dnorm(y, ftk, 1))

# Calculate wHat

# Not sure for the same reasons as above.
wHat[i + 1] <- sum(zHat) / length(zHat)
```

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.