likelihoods

GK

January 11, 2019

Now we will estimate weights by using eBayes approach. To do so, we will optimize the log-likelihood function using optim() function. More precisely, we will find values of weights which maximize the log-likelihood function.

As a **first step**, we have to define the log-likelihood function:

$$L(w) = f(Y|w) = \prod_{i=1}^{n} f(y_i|w)$$

since we assume that y_i are independent for $i = 1, ..., n$ Hence,

$$L(w) = (\frac{1}{(2\pi)^n det(\Sigma)})^{1/2} exp(-\frac{1}{2}(y'\Omega^{-1}y)$$

where Ω is function of weight: w_i.

Given the likelihood funtion above, we know that maximizing the log of likelihood is the same as maximizing the likelihood. So, we will stick with log-likelihood function from there.

As a **next step**, we will add penalty term to the log-likelihood function to smooth it and make estimation more efficient.

We add a penalty on a function as structure of weight parameter defined in CC-Process 1c. Hence from our goal is to maximize log of the following joint likelihood function:

$$p(w, y) = p(y|w)p(w)$$

where $p(y|w)$ is a data model, i.e $L(w)$ and $p(w)$ is already defined in CC-Process 1c s.t

$$p(y|w) = N(y; 0, \Omega(X, w))$$
$$p(w) = N(w; 0, \Pi)$$

Note that this idea comes from LASSO or Ridge regression methods of parameter estimation.

We can write the function in R as follows:

```
loglkl_mvn_penalty <- function(w,d) { #

    time <- ncol(d)-2
    # calculate the covariance matrix for yw
    omega <- gcalc_corr(d,w)

    # calculate the convariance matrix for w
    sigma <- 1/(0.36)*calc_Sigma(time+1, 0.99)

    # data model: log_likelihood
    p_yw <- dmvnorm(d[,ncol(d)], mean = rep(0, nrow(d)),
                    sigma = omega, log = TRUE)
```

```
    # prior on w: log_likelihood
    p_w <- dmvnorm(w, mean = rep(0, time+1),
                   sigma = sigma , log = TRUE)

    # the posterior which will be maximized
    result <- p_yw + p_w

    return(result)
}
```

In pair with the following function:

```
estimate_w <- function(opt_f, grad, pars, d, maxit){
# Estimates w using eBayes approach, i.e finds MLE
# estimates of w.
#
# Args:
#   opt_f: likelihood function to be optimized
#   grad:  gradient of opt_f
#   pars:  initial values for w
#   d:     observed data
#   maxit: maximum number of iterations
#
# Output:
#   opt:   results of optim()

    opt <- optim(par = pars, opt_f, d = d,
                 control = list(fnscale = -1,maxit=maxit),
                 gr = grad)

    # print true values of w
    print("True values of w")
    print(w)

    # estimated w
    return(opt)
}
```

Note that we have added gradient of log-likelihood function f for better optimization. Since the differentitaion of the original likelihood function is way difficult, we tried first numerical (approximate) version of gradient:

$$\bigtriangledown f = (f'_{w_0}, ..., f'_{w_T})$$

where $w = (w_0, ..., w_T)$. By the fundamentals of calculus we have that:

$$f'_{w_i}(w_0, ..., w_T) = \lim_{h \to \infty} \frac{f(w_0, ..., w_i + h, ..., w_T) - f(w_0, ..., w_T)}{h}$$

We have implemented this in practice as follows:

```
calc_gradient_num <- function(f,w,d,epsilon=10^-8){
# Calculates the gradient of a function numerically
#
# Args:
#   f: function (log-likelihood function)
#   d: data frame, last column is response Y,
#      others are input X's
#   w: weights, in a vector form
#
# Output:
#   gr: gradients, in a vector form

    n <- length(w)
    gr <- numeric(n)
    for(i in 1:n) {
        h <- rep(0,n); h[i] <- epsilon
        gr[i] <- (f(w+h,d)-
                      f(w,d))/epsilon
    }
    return(gr)
}
```

Here h is epsilon, given as an infinitesimally small number.

Taking into account assumption about density of weights, log-likelihood function and its gradient, we obtained estimates of w which are reasonably close to the true weight parameters. However, there is a small concern about the estimates with different initial values used in optim(), so optim() might be converging to the local maxima. To see that, we can plot 3D graphs of log-likelihood using the functions given in $3D\_plot\_loglkl.R$ (!!! More work: attach images, additionally code has to be refined)

To better optimize the likelihood function, we will modify some assumptions about the model (specifically, weight parameter). We will transform w_t into the log-scale s.t

$$log(w) \sim N(-1or-2, \Sigma)$$

Following the simple rules of variable transformation, we have the following first result:

1. For $w = (w_0, ..., w_T)$ define $\theta = (\theta_0, ..., \theta_T)$ s.t $\theta_i = log(w_i)$. Hence, $\theta = log(w) = h(w)$ s.t $h^{-1}(\theta) = e^{\theta} = w$. Then

$$p_\theta(\theta) = p_w(h^{-1}(\theta))\left|detJ_{h^{-1}(\theta)}\right|$$

$$J_{h^{-1}(\theta)} = diag(e^{\theta_0}, ..., e^{\theta_T})$$

Hence,

$$\left|detJ_{h^{-1}(\theta)}\right| = \prod_{i=0}^{T} e^{\theta_i} = e^{\sum_{i=0}^{T} y_i} = e^{(T+1)\bar{\theta}}$$

$$p_\theta(\theta) = (2\pi)^{-(T+1)/2}(det\Pi)^{-1/2}e^{(T+1)\bar{\theta}}e^{-\frac{1}{2}(e^{\theta_0},...,e^{\theta_T})\Pi^{-1}(e^{\theta_0},...,e^{\theta_T})'}$$

We will sample θ from the distribution given below and convert sampled θ's to w: the variable of interest:

2. For $w = (w_0, ..., w_T)$ define $\theta = (\theta_0, ..., \theta_T)$ s.t $\theta_i = log(w_i)$. We will sample θ from the distribution given below and convert sampled θ's to w: the variable of interest.

Appropriate code is given below:

To sample weights wisely we could find a distribution which takes into account facts about the structure of w:

1. Highly correlated weights, with possibly AR(1) structure
2. Concentrated around zero
3. Skewed to the right
4. Positive s.t $0 \leq w \leq 1$

There are some processes that likely to follow these characteristics. One is **Beta AR(1)** process model, which has the following structure: (cited from internet/article: An Autoregressive Process for Beta Random Variables, by E.Mackenzie and Bayesian Model Selection for Beta Autoregressive Processes by R.Casarin)

Also we can improve the optimization process by deriving the real gradient of log-likelihood function with penalty term. For that purpose we will utulize the method given in Rassmussen's book GP for ML. We will take the following steps:

As an ultimate measure of accuracy we can calculate the MSE for each method of estimation and different initial values. Appropriate lines of codes are given in *assess_accuracy.R* file and their implementation is given in *test_on_simulation.R* file

Last important part of optimization is a time required for the whole run of optim() function. Lines of codes for time assessment (in benchmark style) is given in the *assess_runtime.R* file