MTH 511a - 2020: Lecture 16

Instructor: Dootika Vats

1 Numerical optimization methods

1.1 Gradient Ascent (Descent)

For concave objective functions, Newton-Raphson is essentially the best algorithm. However, one significant flaw in the algorithm is that information about the Hessian is required to implement it. The Hessian (or the double derivative) may be unavailable or difficult to calculate in some situations.

In such a case, gradient ascent (or gradient descent if the problem is a minimizing problem) is a useful alternative as it does not require the gradient. Consider the objective function $f(\theta)$ that we want to maximize and suppose θ_* is the true maximum. Then, by the Taylor series approximation at a fixed θ_0

$$f(\theta) \approx f(\theta_0) + f'(\theta_0)(\theta - \theta_0) + \frac{f''(\theta_0)}{2}(\theta - \theta_0)^2$$

If $f''(\theta)$ is unavailable, consider assuming that the double derivative is a negative constant. That is, that f is quadratic and concave. Then for a $t > 0$,

$$f(\theta) \approx f(\theta_0) + f'(\theta_0)(\theta - \theta_0) - \frac{1}{2t}(\theta - \theta_0)^2$$

Maximizing $f(\theta)$ and using this approximation would imply maximizing the right hand side. Taking the derivative with respect to θ setting it to zero:

$$f'(\theta_0) - \frac{\theta - \theta_0}{t} \stackrel{set}{=} 0 \Rightarrow \theta = \theta_0 + t f'(\theta_0)\,.$$

The algorithm essentially does a locally-quadratic approximation at the current point $\theta_{(k)}$ and then maximizes that quadratic equation. The value of t indicates how far do we want to jump and is a tuning parameter. If t is large, we take big jumps, as opposed to t small, where the jumps are smaller.

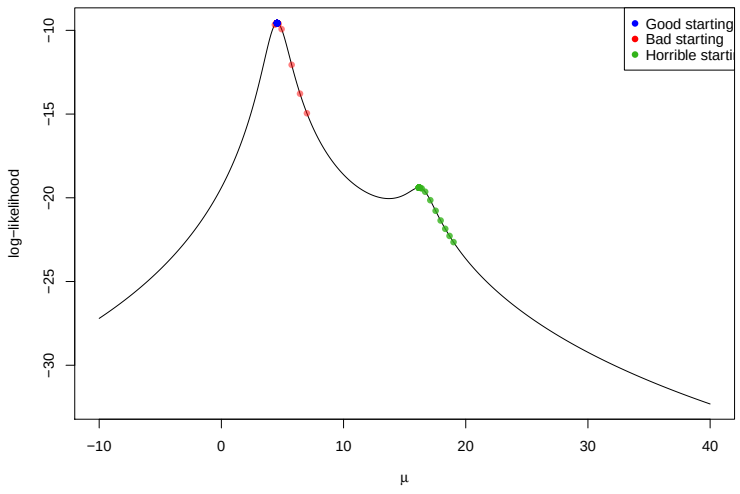
Given a $\theta_{(k)}$, the gradient ascent algorithm does the update

$$\theta = \theta_{(k)} + tf'(\theta_{(k)}),$$

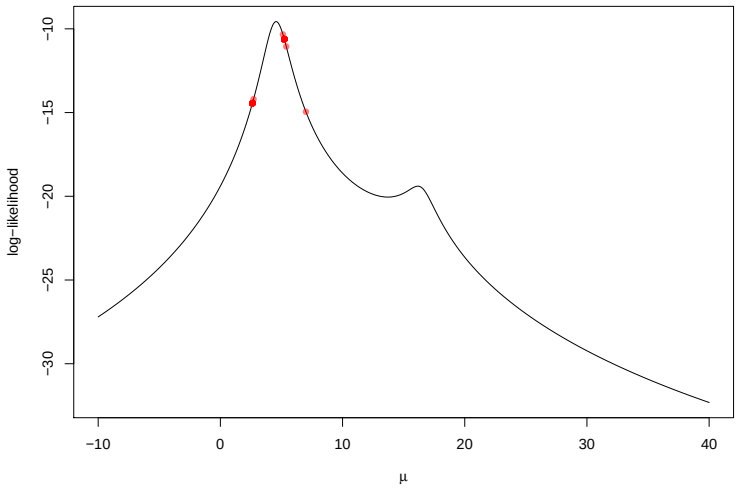
for $t > 0$. The iteration can be stopped when $|\theta_{(k+1)} - \theta_{(k)}| < \epsilon$ for $\epsilon > 0$.

> *For concave functions, gradient ascent converges to the global maxima. In general, gradient ascent always converges to a local maxima.*

Example 1 (Location Cauchy). Recall the location Cauchy example in the previous lecture where the likelihood is not concave. We can implement gradient ascent here and since gradient ascent always converges to a local maxima, it does something more reasonable here (we set $t = .3$) (the codes have been shared)



If we rerun this with the red starting values and $t = 1$, this is too much of a jump size, and the optimization starts oscillating between two points.



So it is really important to try different step sizes t when implementing gradient ascent. There are other solutions to this like adaptive step sizes which we will not discuss.

Gradient Ascent in Higher Dimensions

By a multivariate Taylor series expansion, we can obtain a similar motivation and the iteration in the algorithm is:

$$\theta_{(k+1)} = \theta_{(k)} + t\nabla f(\theta_{(k)})\,.$$

Example 2 (Logistic regression). We have studied linear regression for modeling continuous responses. But when Y is a response of 1s and 0s (Bernoulli) then assuming the Ys are normally distributed is not appropriate. Instead, when the ith covariate is $(x_{i1}, \ldots, x_{ip})^T$, then for $\beta \in \mathbb{R}^p$ logistic regression assumes the model

$$Y_i \sim \text{Bern}\left(\frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}}\right)\,.$$

In other words, the probability that any response takes the value 1 is

$$\Pr(Y_i = 1) = \frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}} := p_i\,.$$

Our goal is to obtain the MLE of β. As usual, first we write down the log-likelihood.

$$\begin{aligned} L(\beta|Y) &= \prod_{i=1}^{n} (p_i)^{y_i}\,(1-p_i)^{1-y_i} \\ \Rightarrow l(\beta) &= \sum_{i=1}^{n} y_i \log p_i + \sum_{i=1}^{n}(1-y_i)\log(1-p_i) \end{aligned}$$

$$= \sum_{i=1}^{n} \log(1-p_i) + \sum_{i=1}^{n} y_i \left(\log p_i - \log(1-p_i)\right)$$
$$= -\sum_{i=1}^{n} \log\left(1+\exp(x_i^T\beta)\right) + \sum_{i=1}^{n} y_i x_i^T \beta$$

Taking derivative:

$$\nabla l(\beta) = \sum_{i=1}^{n} x_i \left[y_i - \frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}} \right] \stackrel{set}{=} 0$$

An analytical solution here is not possible, thus a numerical optimization tool is required. **Note:** You can verify by studying the Hessian that this likelihood function is indeed concave and thus we can use either Newton-Raphson or Gradient Ascent successfully. We will use gradient ascent, and you should on your own, implement N-R.

```
################################################
## MLE for logistic regression
## Using gradient ascent
################################################
library(mcmc) #to load a dataset
data(logit)
head(logit) # y is response and 4 covariates
#   y     x1     x2     x3     x4
# 1 0 -0.162 -0.348 -0.524 -0.312
# 2 1  0.187 -0.410  0.362 -0.366
# 3 1  0.160  1.649 -0.664  0.051
# 4 1  1.536  0.084  0.403  1.732
# 5 1 -0.162 -0.061  0.192  0.256
# 6 0 -2.855 -3.341 -2.549 -1.461


y <- logit$y
X <- as.matrix(logit[, 2:5])
p <- dim(X)[2]

f.gradient <- function(y, X, beta)
{
  beta <- matrix(beta, ncol = 1)
  pi <- exp(X %*% beta) / (1 + exp(X%*%beta))
  rtn <- colSums(X* as.numeric(y - pi))
  return(rtn)
}


store.beta <- matrix(0, nrow = 1, ncol = p)
```

```r
beta_k <- rep(0, p) # start at all 0s
t <- .1
tol <- 1e-5
iter <- 0
diff <- 100
while((diff > tol) && iter < 100) #not too many iterations
{
  iter <- iter+1
  old <- beta_k
  beta_k = old + t* f.gradient(y = y, X= X, beta = old)
  store.beta <- rbind(store.beta, beta_k)
  diff <- sum( (beta_k - old)^2)
}
iter # number of iterations
#[1] 10

beta_k # last step in the optimization
#       x1       x2       x3       x4
#0.7169059 0.8792911 0.4231445 0.5884162
```

1.2 Questions to think about

1. Can you implement Newton-Raphson for the logistic regression problem? Which of the two algorithms is better?
2. What might be a way to adapt the step size t in a problem?