# Newton's Method Notes

2024-10-26

## Introduction:

Newton's method is a powerful optimization technique that extends well into logistic regression, particularly useful for maximum likelihood estimation (MLE). Let us begin with the motivation:

## Motivation:

In logistic regression, our goal is to estimate the probability of a binary outcome $y$(0 or 1) given a set of predictors $x$, using the logistic function:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+...+\beta_n x_n)}}$$

To fit this model, we use MLE, seeking to find the parameters $\beta$ that maximize the likelihood of observing our data. Newton's method with its faster convergence rate, helps solve this by finding a local maximum of the log-likelihood function.

## Newton's Method for Optimization:

Newton's Method is an iterative optimization technique that uses the gradient and curvature (second derivative) of a function to quickly converge to a solution. For logistic regression, this involves:

- Gradient: The first derivative of the log-likelihood function concerning $\beta$
- Hessian: The second derivative (or matrix of second partial derivatives) of the log-likelihood concerning $\beta$.

In general, newton's update step for a parameter $/beta$ is given by:

$$\beta^{new} = \beta^{old} - [H(\beta^{old})]^{-1} \nabla L(\beta^{old})$$

where: * $\nabla L(\beta)$ is the gradient of the log-likelihood * $H(\beta)$ is the Hessian matrix of the log-likelihood

## Logistic Regression: Likelihood Function, Gradient and Hessian

For logistic regression, the log-likelihood function $L(\beta)$ for $n$ observation is:

$$L(\beta) = \sum_{i=1}^{n} (y_i log(p_i) + (1 - y_i) log(1 - p_i))$$

where $p_i = \frac{1}{1+e^{-\beta^T x_i}}$

The gradient of $L(\beta)$ with respect to $\beta$ is:

$$\nabla L(\beta) = X^T(y - p)$$

where:

- $X$ is the design matrix of predictors
- $y$ is the vector of observed outcomes
- $p$ is the vector of predicted probabilities, calculated as $p = \frac{1}{1+e^{-\beta^T x_i}}$

The Hessian matrix $H(\beta)$ is:

$$H(\beta) = -X^T W X$$

where $W$ is a diagonal matrix with entries $W_{ii} = p_i(1 - p_i)$

## Implimentaiton in R:

```
# Sample Data:
set.seed(123)
n <- 100 # Number of Observations
x <- matrix(rnorm(n*2),n,2) # Two Predictors
beta_true <- c(0.5,-0.3)
y <- rbinom(n,1,plogis(x %*% beta_true))

# Adding Intercept Term:
X <- cbind(1,x) # Design Matrix with Intercept
beta <- rep(0, ncol(X)) # Initial Values for Beta Coefficient

# Newton's Method Parameters:
max.iter <- 100
tol <- 1e-6

# NM Loop:
for(i in 1:max.iter){
  p <- 1/(1 + exp(-X %*% beta)) # Predicted Probabilities
  gradient <- t(X) %*% (y-p) # Gradient of the Log-Likelihood
  W <- diag(as.vector(p * (1-p))) # Diagonal Matrix
  hessian <- -t(X) %*% W %*% X # Hessian Matrix

  # Newton's Update Step:
  beta_new <- beta - solve(hessian) %*% gradient

  # Check Convergence:
  if(sum(abs(beta_new - beta)) < tol){
    beta <- beta_new
    break
  }
```

```
  beta <- beta_new

}

cat("Estimated Coefficients:\n",beta)
```

```
## Estimated Coefficients:
##  0.02968842 0.2870145 -0.245101
```

# Explanation of code:

1. **Data Generation**: Simulated binary outcomes and predictors
2. **Initialization:** $X$ is the design matrix (including the intercept), nd $\beta$ is initialized as a zero vector.
3. **Iterative Optimization**:

- Calculate predicted probabilities $p$
- Compute the gradient and Hessian of the log-likelihood
- Perform the Newton's Update

4. **Convergence Check**: The loop stops when the change in $\beta$ falls below a tolerance level, indicating convergence

**This code gives the estimated coefficients, approximating those that maximize the likelihood**

# Checking Model Perfomance:

Now, let us check for model accuracy. We will use a created dataset where the x-values are from a sequence from 1 to 500 with added noise. The response variable is whether or not the x-value is greater than the rolling average with a window of 5:

```
# Create Data:
window <- 3 # Moving Average Window
n <- 500 # Datapoints
x_val <- seq(1,n,by=1) + rnorm(n,0,n/2)
moving_average <- roll_mean(x_val,window)
y <- ifelse(x_val > moving_average,1,0)

# Data Frame of X and Y Values:
xValues <- x_val[window:n]
yValues <- y[window:n]
data <- data.frame(xValues,yValues)
head(data)
```

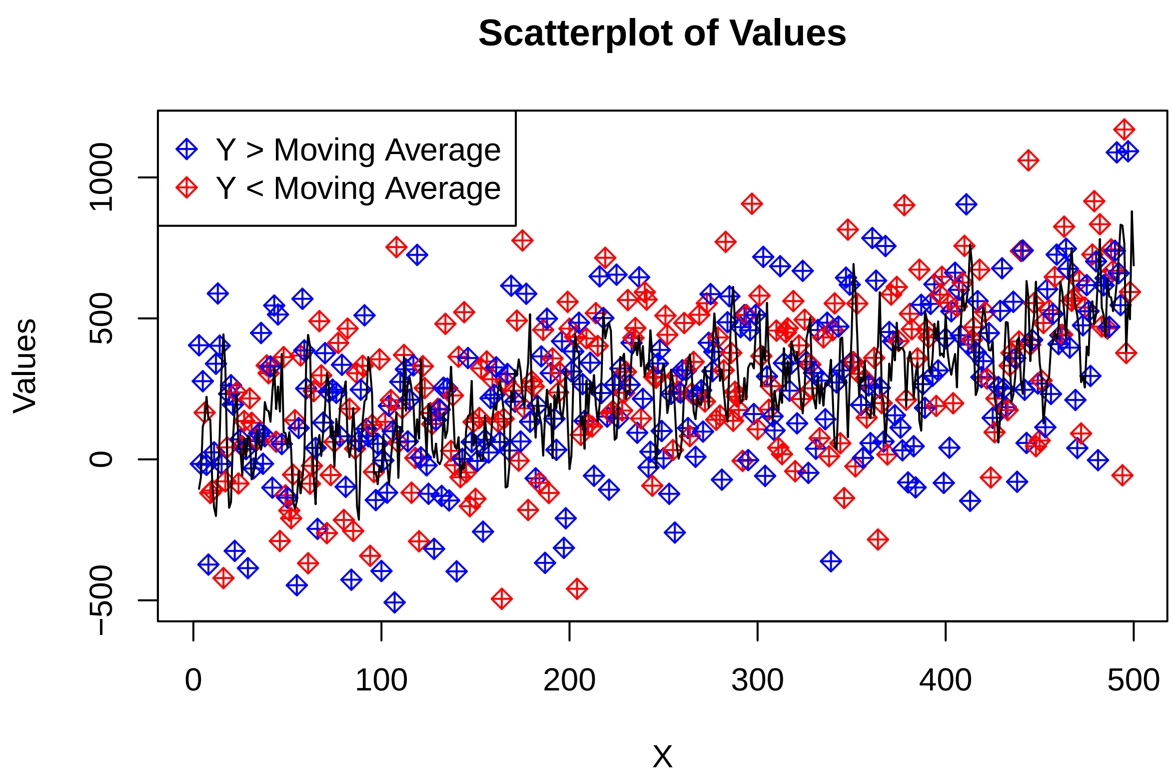
```
##     xValues yValues
## 1 -82.97931       1
## 2  26.62416       1
## 3 404.62719       1
## 4 -16.14128       0
## 5 277.19987       1
## 6 165.68853       1
```

```r
# Full Data Frame:
#newton <- data.frame(xValues,moving_average[window:n],yValues)
```

Graph the scatterplot:

```r
pch <- 9 # pch Value

# Plot
plot(data$xValues,
     col = ifelse(y == 1, "blue","red"),
     xlab = "X",
     ylab = "Values",
     main = "Scatterplot of Values",
     pch = pch
     )
lines(moving_average)
legend("topleft",
       legend = c("Y > Moving Average","Y < Moving Average"),
       col = c("blue","red"),
       pch = pch
       )
```




Define the initial parameters and compute initial log-likelihood

```r
# Create Matrices for X and Y:
X <- as.matrix(cbind(1,data$xValues))
Y <- as.matrix(data$yValues)

# Initialize beta at zeros:
beta_init <- rep(0,ncol(X))

# Logistic Function:
logistic <- function(z){
  1 / ( 1 +exp(-z))
}

# Coefficient Vector
beta_vec <- c()

# Log-Likelihood Function:
log_likelihood <- function(Y, X, beta){
  p <- logistic(X %*% beta)
  sum(Y * log(p) + (1 - Y) * log(1 - p))
}

# Compute initial Log-Likelihood:
initial_ll <- log_likelihood(Y, X, beta_init)
cat("Initial Log Likelihood: ", initial_ll)
```

```
## Initial Log Likelihood:  -345.1873
```

Apply Newton's Method to Optimize Parameters

```r
# Newton's Method parameters
max_iter <- 100
tol <- 1e-6
beta <- beta_init

# Beta Vector
beta_vec <- c()

# Newton's Method implementation
for (i in 1:max_iter) {
  p <- logistic(X %*% beta)
  gradient <- t(X) %*% (Y - p)
  W <- diag(as.vector(p * (1 - p)))
  hessian <- -t(X) %*% W %*% X

  # Regularize Hessian to ensure it's invertible
  lambda <- 1e-5
  hessian_reg <- hessian - lambda * diag(ncol(hessian))

  # Update beta
  beta_new <- beta - solve(hessian_reg) %*% gradient

  # Update Beta Vector
  beta_vec <- append(beta_vec,beta_new)
```

```r
  # Check convergence
  if (sum(abs(beta_new - beta)) < tol) {
    beta <- beta_new
    break
  }

  beta <- beta_new
}

# Beta Matrix:
b.mat <- matrix(beta_vec,ncol = 2,byrow = T)

# Compute final log-likelihood
optimized_log_likelihood <- log_likelihood(Y, X, beta)
cat("Optimized Log-Likelihood (after Newton's Method):", optimized_log_likelihood)
```

```
## Optimized Log-Likelihood (after Newton's Method): -244.4032
```

Accuracy Improvement:

```r
# Predictions and accuracy at initial parameters
initial_pred <- ifelse(logistic(X %*% beta_init) > 0.5, 1, 0)
initial_accuracy <- mean(initial_pred == Y)
cat("Initial Accuracy:", initial_accuracy, "\n")
```

```
## Initial Accuracy: 0.4759036
```

```r
# Predictions and accuracy after Newton's Method
optimized_pred <- ifelse(logistic(X %*% beta) > 0.5, 1, 0)
optimized_accuracy <- mean(optimized_pred == Y)
cat("Optimized Accuracy:", optimized_accuracy, "\n")
```

```
## Optimized Accuracy: 0.7590361
```

With an increase in accuracy and a decrease in likelihood means Newton's method is beneficial for improving model accuracy.