ETC3555 2018 - Lab 5

Linear models and gradient descent (II/II)

Cameron Roach and Souhaib Ben Taieb

17 August 2018

Solutions

14 marks total.

Assignment - Question 1

1. (a)

(2 marks)

Assume $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ is linearly separable and $y_n \in \{-1, +1\} \ \forall \ n$. For an incorrectly classified point, $y_n \mathbf{w}^T \mathbf{x}_n < 0 \Rightarrow e_n(\mathbf{w}) = -y_n \mathbf{w}^T \mathbf{x}_n$ and $\nabla e_n(\mathbf{w}) = -y_n \mathbf{x}_n$. If a point is correctly classified then $y_n \mathbf{w}^T \mathbf{x}_n > 0 \Rightarrow e_n(\mathbf{w}) = \nabla e_n(\mathbf{w}) = 0$.

Applying SGD with this error measure gives Algorithm 1.

Data: $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$
Result: final weights: $\mathbf{w}(t+1)$
initialise weights at $t = 0$ to $\mathbf{w}(0)$;
for $t = 0, 1, 2, \ldots$ **do**
    select a training point $(\mathbf{x}_n, y_n)$ uniformly at random;
    compute the gradient of the error for this single data point, $\mathbf{g}_t = \nabla e_n(\mathbf{w}(t))$;
    set the direction to move, $\mathbf{v}_t = -\mathbf{g}_t$;
    update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$;
    iterate until $e_n(\mathbf{w}(t+1)) = 0 \ \forall \ (\mathbf{x}_n, y_n)$;
end

Algorithm 1: SGD on $e_n(\mathbf{w})$

If $(\mathbf{x}_n, y_n)$ is correctly classified the gradient will equal zero and no change to the weights will occur. Alternatively, if $(\mathbf{x}_n, y_n)$ is an incorrectly classified point the gradient will be $\mathbf{g}_t = \nabla e_n(\mathbf{w}(t)) = -y_n \mathbf{x}_n$. We can restrict our selection of points at each iteration to those that are misclassified because correctly classified points will not affect weights. Substituting $\mathbf{v}_t = y_n \mathbf{x}_n$, choosing a learning rate $\eta = 1$ and noting that $e_n(\mathbf{w}) = 0$ only when $y_n \mathbf{w}^T \mathbf{x}_n > 0$, we see that this is equivalent to the PLA (see Algorithm 2).

Data: $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$
Result: final weights: $\mathbf{w}(t+1)$
initialise weights at $t = 0$ to $\mathbf{w}(0)$;
for $t = 0, 1, 2, \ldots$ **do**
    select a misclassified training point $(\mathbf{x}_n, y_n)$ uniformly at random;
    update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + y_n \mathbf{x}_n$;
    iterate until $y_n \mathbf{w}^T(t+1)\mathbf{x}_n > 0 \ \forall \ n$;
end

Algorithm 2: PLA

1. (b)

(2 marks)

Let $\mathbf{w}$ be a non-zero vector. If a point is misclassified then $y_n \mathbf{w}^T x_n < 0$ and

$$\nabla e_n(\mathbf{w}) = \frac{-y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}} \xrightarrow{\|\mathbf{w}\| \to \infty} -y_n \mathbf{x}_n.$$

Alternatively, if a point is correctly classified then $y_n \mathbf{w}^T x_n > 0$ and

$$\nabla e_n(\mathbf{w}) = \frac{-y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}} \xrightarrow{\|\mathbf{w}\| \to \infty} 0.$$

So for large $\mathbf{w}$ we can use the above limits as gradient approximations. The result follows in the same manner as for part (a).

Assignment - Question 2

2. (a)

(2 marks)

The error function can be expressed as

$$e_n(\mathbf{w}) = \begin{cases} 0 & \text{if } y_n \mathbf{w}^T \mathbf{x}_n \geq 1, \\ (1 - y_n \mathbf{w}^T \mathbf{x}_n)^2 & \text{if } y_n \mathbf{w}^T \mathbf{x}_n < 1. \end{cases}$$

This function is plotted below.

```r
library(tidyverse)

data_frame(
  y = c(rep(1, 41), rep(-1, 41)),
  x = rep(1, 82),
  w = rep(seq(-2, 2, 0.1), 2)
) %>%
  mutate(e_n = if_else(y*w*x >= 1, 0, (1-y*w*x)^2)) %>%
  ggplot(aes(x = w, y = e_n, colour = factor(y))) +
  geom_line()
```

When $y_n \mathbf{w}^T \mathbf{x}_n < 1$ the error function $e_n(\mathbf{w})$ is continuous with respect to $\mathbf{w}$ as it is a polynomial function which are continuous (and differentiable) everywhere. Alternatively, when $y_n \mathbf{w}^T \mathbf{x}_n \geq 1$ the error function $e_n(\mathbf{w})$ is again continuous with respect to $\mathbf{w}$ as it is a constant function which are continuous (and differentiable) everywhere. Taking the limits from above and below at $y_n \mathbf{w}^T \mathbf{x}_n = 1$ we see that

$$\lim_{y_n \mathbf{w}^T \mathbf{x}_n \to 1^+} e_n(\mathbf{w}) = \lim_{y_n \mathbf{w}^T \mathbf{x}_n \to 1^+} (1 - y_n \mathbf{w}^T \mathbf{x}_n)^2 = 0,$$
$$\lim_{y_n \mathbf{w}^T \mathbf{x}_n \to 1^-} e_n(\mathbf{w}) = \lim_{y_n \mathbf{w}^T \mathbf{x}_n \to 1^-} 0 = 0,$$

and so $e_n(\mathbf{w})$ is continuous here as well. We can conclude that $e_n(\mathbf{w})$ is a continuous function. It is also differentiable at $y_n \mathbf{w}^T \mathbf{x}_n = 1$, because the left hand derivative and right hand derivative are both equal to zero.

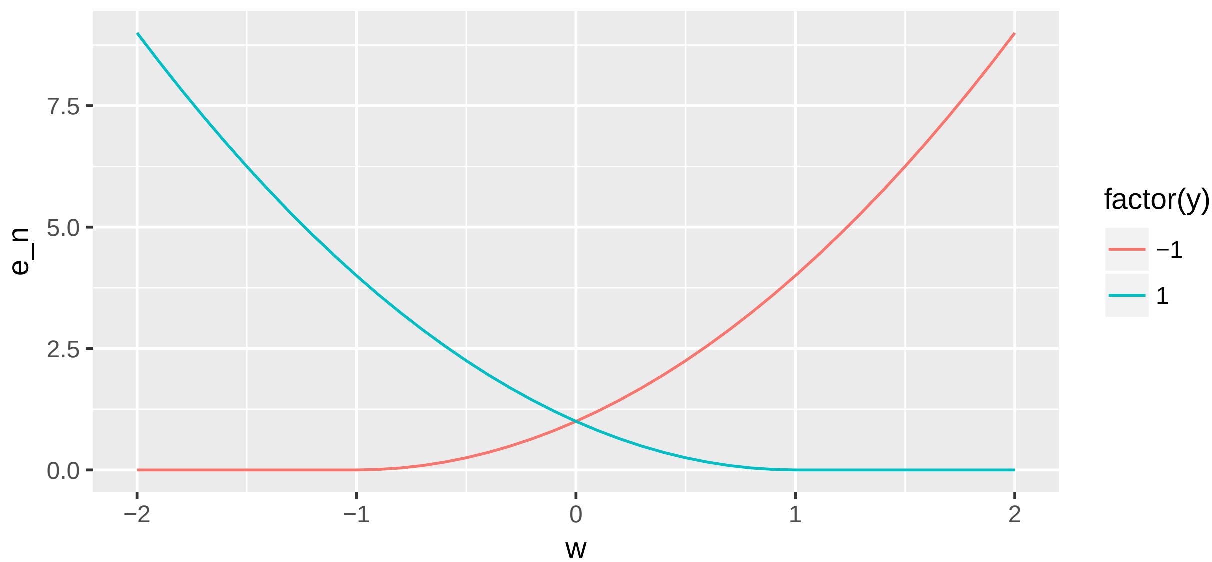


Figure 1: $e_n(\mathbf{w})$ for $y \in \{1, -1\}$ and $\mathbf{x}_n = 1$.

The deriviative is

$$\nabla e_n(\mathbf{w}) = \begin{cases} 0 & \text{if } y_n \mathbf{w}^T \mathbf{x}_n \geq 1, \\ -2y_n \mathbf{x}_n (1 - y_n \mathbf{w}^T \mathbf{x}_n) & \text{if } y_n \mathbf{w}^T \mathbf{x}_n < 1, \end{cases}$$

where we have used the chain rule to obtain the second line.

2. (b)

(2 marks)

When $\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n$ we have $y_n \mathbf{w}^T \mathbf{x}_n < 0$ and it follows that

$$e_n(\mathbf{w}) = \left(1 - y_n \mathbf{w}^T \mathbf{x}_n\right)^2 \geq (1 - 0)^2 = 1 = [\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!].$$

Alternatively, when $\text{sign}(\mathbf{w}^T \mathbf{x}_n) = y_n$ we have $y_n \mathbf{w}^T \mathbf{x}_n > 0$. If $y_n \mathbf{w}^T \mathbf{x}_n \in (0, 1)$ then

$$e_n(\mathbf{w}) = \left(1 - y_n \mathbf{w}^T \mathbf{x}_n\right)^2 \geq (1 - 1)^2 = 0 = [\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!],$$

and if $y_n \mathbf{w}^T \mathbf{x}_n \geq 1$ then

$$e_n(\mathbf{w}) = 0 = [\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!].$$

Hence, $e_n(\mathbf{w})$ is an upper bound for $[\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) = y_n]\!]$. Using this result we can show the in sample classification error has the upper bound

$$\begin{aligned} E_{\text{in}} &= \frac{1}{N} \sum_{n=1}^{1} [\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!] \\ &\leq \frac{1}{N} \sum_{n=1}^{1} e_n(\mathbf{w}). \end{aligned}$$

2. (c)

(2 marks)

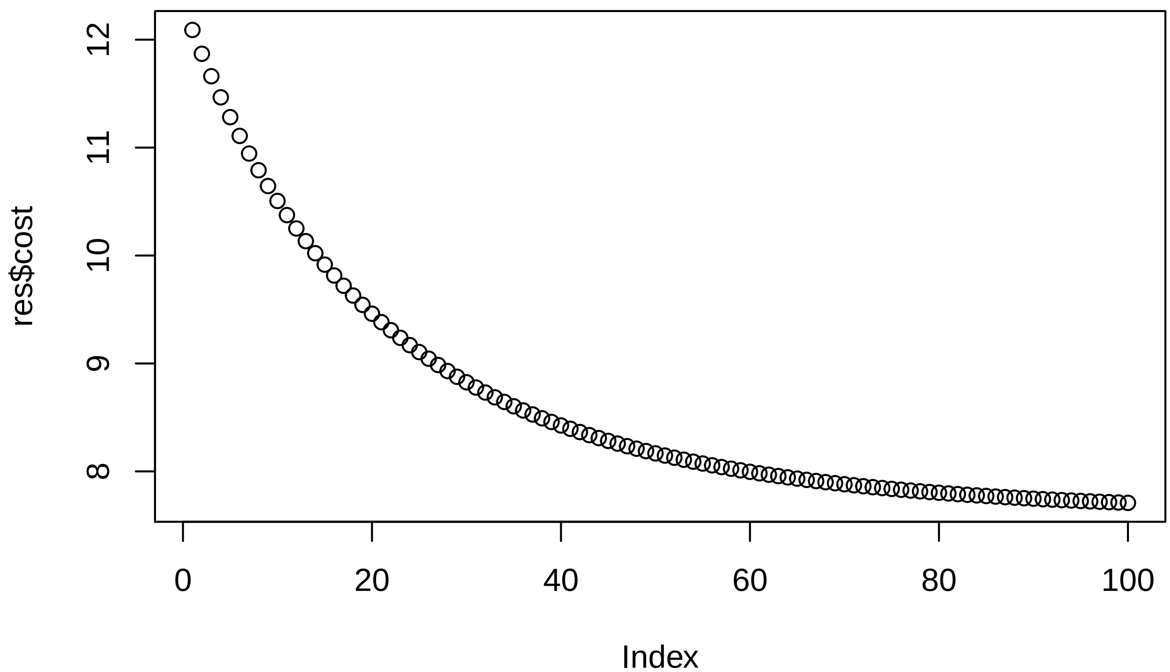
Use a similar argument as in Q1. (a). Use the fact that $y_n^2 = 1$ when rearranging gradient.

Assignment - Question 3

(4 marks)

```
library(magrittr)

#### Helper functions ---------------------------------------------------
Ein_linreg <- function(X, y, w){
  sum((y-X%*%w)^2)/length(y)
}

Ein_logreg <- function(X, y, w){
  sum(log(1+exp(-y*X%*%w)))/length(y)
}

gEin_linreg <- function(X, y, w){
  1:length(y) %>%
    sapply(function(n) { -2*X[n,]*c(y[n]-X[n,]%*%w) }) %>%
    rowMeans()
}

gEin_logreg <- function(X, y, w){
  1:length(y) %>%
    sapply(function(n) { -y[n]*X[n,]/c(1+exp(y[n]*w%*%X[n,])) }) %>%
    rowMeans()
}

GD <- function(X, y, Ein, gEin, w0, eta, precision, nb_iters){
  allw <- vector("list", nb_iters)
  allgrad <- vector("list", nb_iters)
  cost <- numeric(nb_iters)
  allw[[1]] <- w0
  t <- 1

  repeat {
    cost[t] <- Ein(X, y, allw[[t]])
    allgrad[[t]] <- gEin(X, y, allw[[t]])
    allw[[t+1]] <- allw[[t]] - eta*allgrad[[t]]
    t <- t + 1

    if (abs(cost[t]-cost[t-1]) < precision | t > nb_iters) { break }
  }

  list(allw = allw, cost = cost, allgrad = allgrad)
}


#### Linear regression ---------------------------------------------------
```

```r
set.seed(1900)
# Function taken from Friedman et al.
genx <- function(n,p,rho){
  #    generate x's multivariate normal with equal corr rho
  # Xi = b Z + Wi, and Z, Wi are independent normal.
  # Then Var(Xi) = b^2 + 1
  #  Cov(Xi, Xj) = b^2  and so cor(Xi, Xj) = b^2 / (1+b^2) = rho
  z <- rnorm(n)
  if(abs(rho) < 1){
    beta <- sqrt(rho/(1-rho))
    x <- matrix(rnorm(n*p), ncol=p)
    A <- matrix(rnorm(n), nrow=n, ncol=p, byrow=F)
    x <- beta * A + x
  }
  if(abs(rho)==1){ x=matrix(rnorm(n),nrow=n,ncol=p,byrow=F)}

  return(x)
}


N <- 100
p <- 10
rho <- 0.2
X <- genx(N, p, rho)
w_true <- ((-1)^(1:p))*exp(-2*((1:p)-1)/20)
eps <- rnorm(N)
k <- 3
y <- X %*% w_true + k * eps

res <- GD(X, y, Ein_linreg, gEin_linreg, rep(0, p), 0.01, 0.0001, 100)
plot(res$cost)
```



```r
print(w_true)
```

```
##  [1] -1.0000000  0.9048374 -0.8187308  0.7408182 -0.6703200  0.6065307
##  [7] -0.5488116  0.4965853 -0.4493290  0.4065697
```

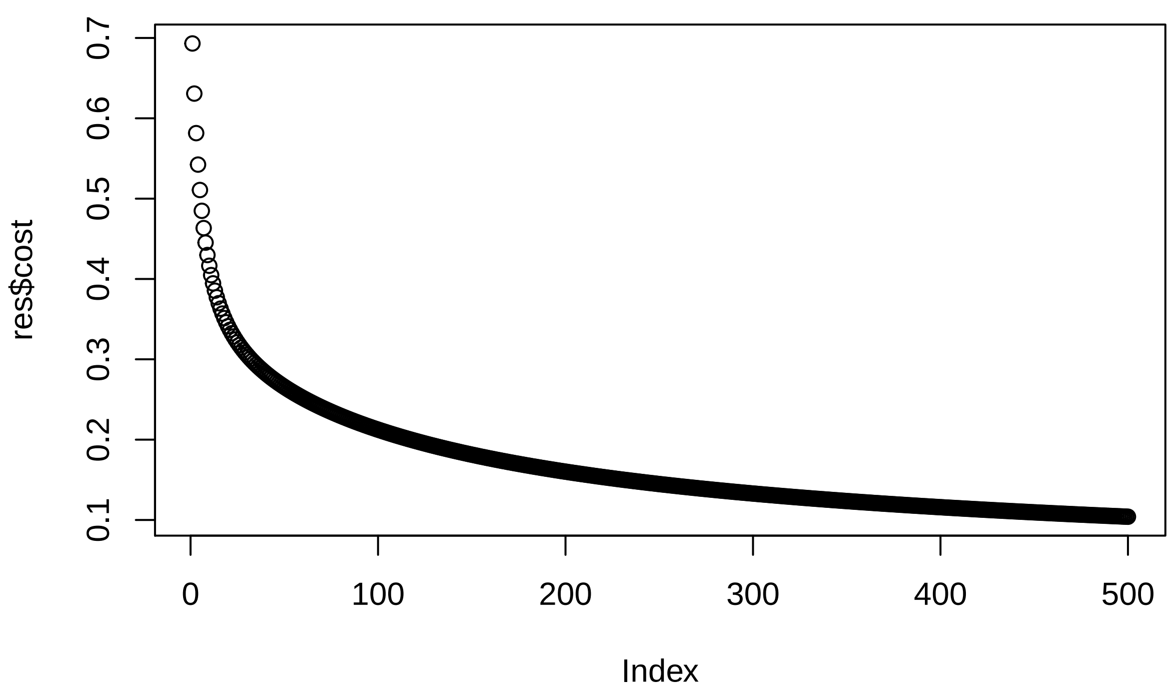
```r
print(unlist(tail(res$allw, 1)))
```

```
##  [1] -0.8974398  0.7291112 -0.4577217  0.2107578 -0.6133777  0.3827985
##  [7] -0.5230292  0.6739643 -0.4416187  0.1658106
```

```r
data <- data.frame(X)
data$y <- y
print(as.numeric(coef(lm(y ~ . - 1, data))))
```

```
##  [1] -1.0664991  0.9281946 -0.5811486  0.1682512 -0.6704339  0.4158793
##  [7] -0.5952950  0.7382885 -0.4633203  0.3327759
```

```r
#### Logistic regression ---------------------------------------------------------
set.seed(1900)
N <- 100
l <- -5; u <- 5
x <- seq(l, u, by = 0.1)
w_true <- matrix(c(-3, 1, 1), ncol = 1)
a <- -w_true[2]/w_true[3]
b <- -w_true[1]/w_true[3]

X0 <- matrix(runif(2 * N, l, u), ncol = 2)
X <- cbind(1, X0)
y <- sign(X %*% w_true)

res <- GD(X, y, Ein_logreg, gEin_logreg, rep(0, 3), 0.05, 0.0001, 500)
plot(res$cost)
```



```r
print(w_true)
```

```
##      [,1]
## [1,]   -3
## [2,]    1
## [3,]    1
```

```r
w_best <- unlist(tail(res$allw, 1))
print(w_best)
```

```
## [1] -2.0863827  1.0027444  0.9425643
```

```r
plot(c(l, u), c(u, l), type = 'n', xlab = "x1", ylab = "x2")
lines(x, a*x +b)
points(X0, col = ifelse(y == 1, "red", "blue"))

a_best <- -w_best[2]/w_best[3]
b_best <- -w_best[1]/w_best[3]
lines(x, a_best*x + b_best, col = "red")
```