ETC3555 2018 - Lab 5

Linear models and gradient descent (II/II)

Cameron Roach and Souhaib Ben Taieb

17 August 2018

Assignment - Question 1

Solve exercise 3.10 in Learning From Data.

Assignment - Question 2

Solve problem 3.4 in Learning From Data (problem 1.5 is given below).

Assignment - Question 3

Complete the code for the function `Ein_linreg` which computes the in-sample error of a linear regression fit based on the weight vector `w`. This function will take the following arguments:

1. `X`: an input matrix of dimension $n \times p$.
2. `y`: a response vector of dimension $n \times 1$.
3. `w`: a weight vector of dimension $p \times 1$.

```
Ein_linreg <- function(X, y, w){

}
```

Write a similar function for logistic regression.

```
Ein_logreg <- function(X, y, w){

}
```

For both linear regression and logistic regression, write a function that computes the gradient of the function `Ein` at `w`.

```
gEin_linreg <- function(X, y, w){

}
```

```
gEin_logreg <- function(X, y, w){

}
```

Complete the code of the function `GD` which applies gradient descent to the function `Ein` starting from `w0`. The function will take the following arguments:

1. `X`: an input matrix of dimension $n \times p$.
2. `y`: a response vector of dimension $n \times 1$.
3. `Ein`: a function which takes arguments `X`, `y` and `w`, and computes the in-sample error for `w`.
4. `gEin`: a function which takes arguments `X`, `y` and `w`, and computes the gradient of Ein at `w`.
5. `w0`: the initial weights
6. `eta`: the learning rate
7. `precision`: a small value

8. `nb_iters`: the maximum number of iterations

The function will stop when $|Ein(w(t+1)) - Ein(w(t))| < precision$ or when $t = nb\_iters$

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



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

For linear regression, try your functions on the following example.

```
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)

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

Try different values of `eta`, `precision` and `nb_iters`, and run the example again with your best values. Compare your solution with the closed-form solution.

For logistic regression, try your functions on the following example.

```r
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)

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

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")
```

Exercise

Update your functions to implement stochastic gradient descent, and include L_2 regularization for both linear and logistic regression

TURN IN

- Your `.Rmd` file (which should knit without errors and without assuming any packages have been pre-loaded)
- Your Word (or pdf) file that results from knitting the Rmd.
- DUE: August 28, 11:55pm (late submissions not allowed), loaded into moodle

Exercise 3.10

(a) Define an error for a single data point $(\mathbf{x}_n, y_n)$ to be

$$\mathrm{e}_n(\mathbf{w}) = \max(0, -y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n).$$

Argue that PLA can be viewed as SGD using e_n.

(b) For logistic regression with a very large $\mathbf{w}$, argue that minimizing E_{in} using SGD is similar to PLA. This is another indication that the logistic regression weights can be used as a good approximation for classification.

Figure 1: Source: Abu-Mostafa et al. Learning from data. AMLbook.

Problem 3.4 In Problem 1.5, we introduced the Adaptive Linear Neuron (Adaline) algorithm for classification. Here, we derive Adaline from an optimization perspective.

(a) Consider $E_n(\mathbf{w}) = \max(0, 1 - y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n)^2$. Show that $E_n(\mathbf{w})$ is continuous and differentiable. Write down the gradient $\nabla E_n(\mathbf{w})$.

(b) Show that $E_n(\mathbf{w})$ is an upper bound for $[\![\mathrm{sign}(\mathbf{w}^{\mathrm{T}} \mathbf{x}_n) \neq y_n]\!]$. Hence, $\frac{1}{N} \sum_{n=1}^{N} E_n(\mathbf{w})$ is an upper bound for the in-sample classification error $E_{\mathrm{in}}(\mathbf{w})$.

(c) Argue that the Adaline algorithm in Problem 1.5 performs stochastic gradient descent on $\frac{1}{N} \sum_{n=1}^{N} E_n(\mathbf{w})$.

Figure 2: Source: Abu-Mostafa et al. Learning from data. AMLbook.

Problem 1.5 We know that the perceptron learning algorithm works like this: In each iteration, pick a random $(\mathbf{x}(t), y(t))$ and compute $\rho(t) = \mathbf{w}^{\mathrm{T}}(t)\mathbf{x}(t)$. If $y(t) \cdot \rho(t) \leq 0$, update $\mathbf{w}$ by

$$\mathbf{w}(t+1) \longleftarrow \mathbf{w}(t) + y(t) \cdot \mathbf{x}(t) \ ;$$

One may argue that this algorithm does not take the 'closeness' between $\rho(t)$ and $y(t)$ into consideration. Let's look at another perceptron learning algorithm: In each iteration, pick a random $(\mathbf{x}(t), y(t))$ and compute $\rho(t)$. If $y(t) \cdot \rho(t) \leq 1$, update $\mathbf{w}$ by

$$\mathbf{w}(t+1) \longleftarrow \mathbf{w}(t) + \eta \cdot (y(t) - \rho(t)) \cdot \mathbf{x}(t) \ ,$$

where η is some constant. That is, if $\rho(t)$ agrees with $y(t)$ well (their product is > 1), the algorithm does nothing. On the other hand, if $\rho(t)$ is further from $y(t)$, the algorithm changes $\mathbf{w}(t)$ more. In this problem, you are asked to implement this algorithm and check its performance.

(a) Generate a training data set of size 100 similar to that used in Exercise 1.4. Generate a test data set of size $10,000$ from the same process. To get g, run the algorithm above with $\eta = 100$ on the training data set, until it converges (no more possible updates) or a maximum of $1,000$ updates has been reached. Plot the training data set, the target function f, and the final hypothesis g on the same figure. Report the error on the test set.

(b) Use the data set in (a) and redo everything with $\eta = 1$.

(c) Use the data set in (a) and redo everything with $\eta = 0.01$.

(d) Use the data set in (a) and redo everything with $\eta = 0.0001$.

(e) Compare the results that you get from (a) to (d).

The algorithm above is a variant of the so-called Adaline (*Adaptive Linear Neuron*) algorithm for perceptron learning.

Figure 3: Source: Abu-Mostafa et al. Learning from data. AMLbook.