Language Classification

Benjamin Panny

2023-10-05

Part 1: Learning weights in logistic regression

You are training a classifier for reviews of a new product recently released by a company. You design a couple of features,

$$\text{Initialize } w_1 = w_2 = b = 0$$
$$\eta = 0.2$$
$$\hat{y}_i = w_1 x_{1i} + w_2 x_{2i} + b$$
$$\text{Minimize } L_{CE}(\hat{y}, y) = -\log p\left(y \mid x\right) = -\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]$$

For $x_1 = 2, x_2 = 1, y = 1$

$$\begin{aligned} L &= -\left[1 \log \sigma\left(0\right)) + (1-y)\log(1-\sigma\left(0\right))\right] \\ &= -\left[\log(0.5) + 0\right] = 0.69 \end{aligned}$$

The update or recurrence equation is

$$w_{t+1} = w_t - \eta \frac{d}{dw} L_{CE}$$
$$b_{t+1} = b_t - \eta \frac{d}{db} L_{CE}$$

$$\frac{dL}{dw_j} = \left[\sigma(wx+b) - y\right] x_j$$
$$\frac{dL}{db} = \left[\sigma(wx+b) - y\right]$$

Let's implement the sigmoid equation and the update equation in a function `update_lr_weights`

```
sigmoid <- function(x) {
  1 / (1 + exp(-x))
}
```

```
update_lr_weights <- function(data, lr){
  # assumes weights start with w
  weight_cols <- data %>%
    select(starts_with('w')) %>%
    colnames()

  # assumes features start with x
  feature_cols <- data %>%
    select(starts_with('x')) %>%
    colnames()

  # look at each x1, x2, y tuple
  for (i in 1:nrow(data)){
    weights <- data[,weight_cols]
    x <- data[,feature_cols]
    y <- data[i,'y'] %>% pull(y)
    j <- 1
    b <- data[i,'b'] %>% pull(b)

    # update weights and gradients for each tuple, for each weight
    for (weight_col in weight_cols){
      covariate <- x[i,j] %>% pull()
      weight <- data[i, weight_col] %>% pull()
      data[i+1, weight_col] <- weight - lr*((sigmoid(sum(weights[i,] * x[i,]) + b) - y)*covariate)
      data[i, paste0('eta_dl.d', weight_col)] <- lr*((sigmoid(sum(weights[i,] * x[i,]) + b) - y)*covariate)
      j <- j + 1
    }
    # update bias, gradient for each tuple
    data[i+1, 'b'] <- b - lr*(sigmoid(sum(weights[i,] * x[i,]) + b) - y)
    data[i, 'eta_dl.db'] <- lr*(sigmoid(sum(weights[i,] * x[i,]) + b) - y)

    # get error for each tuple
    data[i, 'error'] <- -(y*log(sigmoid(sum(weights[i,] * x[i,]) + b)) + (1-y) * log(1 - sigmoid(sum(weights[i,] * x[i,]) +
b)))
  }
  return(data)
}

data <- tibble(x1 = c(2,1,0), x2 = c(1,3,4), y = c(1,0,0),
               w1 = 0, w2 = 0, b = 0, error = 0, eta_dl.dw1 = 0, eta_dl.dw2 = 0, eta_dl.db = 0)


update <- update_lr_weights(data, lr = 0.2)
```

```
update %>%
  mutate(t = seq(0,nrow(update)-1,1)) %>%
  relocate(t) %>%
  kableExtra::kable() %>%
  kableExtra::kable_styling()
```

t	x1	x2	y	w1	w2	b	error	eta_dl.dw1	eta_dl.dw2	eta_dl.db
0	2	1	1	0.0000000	0.0000000	0.0000000	0.6931472	-0.2000000	-0.1000000	-0.1000000
1	1	3	0	0.2000000	0.1000000	0.1000000	1.0374880	0.1291313	0.3873938	0.1291313
2	0	4	0	0.0708687	-0.2873938	-0.0291313	0.2682519	0.0000000	0.1882279	0.0470570
3	NA	NA	NA	0.0708687	-0.4756217	-0.0761882	NA	NA	NA	NA

This table shows the weights at each timestep, the learning rate adjusted gradients at each timestep, and the error at each timestep. Timestep indexes the x, y, w, b tuples, starting at 0 and ending at 3. Interestingly, we get the largest updates when error is largest and smaller updates when error is smallest. We can also observe that the update for a weight is 0 when the covariate on a given observation is 0. In these three observations, it appears estimates are nudged upwards when the outcome is 1 and downwards when the outcome is 0. However, whether or not the updates will be positive or negative will always depend on the current values of the covariates, the weights and bias, and the outcome. The updates will always follow the opposite direction of the steepest increase in the Loss function with respect to the parameters (and therefore the values of the covariates and outcomes that influence their derivative).