The Perceptron Learning Algorithm

Jing-Mao Ho

1 The Idea of a Perceptron

The perceptron learning algorithm is one of the most basic algorithms in machine learning, and it was originally created by Frank Rosenblatt in 1962. It is a type of supervised machine learning algorithms, and is a linear classifier. The perceptron algorithm is very basic as well as important because it can be applied to the design of other machine learning algorithms. For example, one of the ways in which the neural networks can be implemented is the multilayer perceptron. The goal of a perceptron is to find separating hyperplane that successfully divides the data into two groups. To make the algorithm work, we need assume that the given data are linearly separable. This assumption is called "linear separability."

$\mathbf{X} = [x_{ij}]_{m\times n} \quad (x_{ij} \in \mathbf{R}^m)$ and $\mathbf{Y} = [y_j]_{n\times 1} \quad (y_j \in (0,1))$, where $i = (1,2,3,...,m)$ and $j = (1,2,3,...,n)$. Denote each data point $(\mathbf{X}_j, y_j)$ where $j = (1,2,3,...,n)$ and $\mathbf{X}_j = [x_{1j}, x_{2j}, x_{2j}, \cdots, x_{mj}]_{m\times 1}$. Note that n is the input size and m is the number of features. Assume that there is a separating hyperplane $\beta_0 + \sum_{i=1}^{m} \beta_i x_i = 0$ (based on the assumption of linear separability) that separates $\{\mathbf{X}_j | y_j = 1\}$ and $\{\mathbf{X}_j | y_j = 0\}$. The hyperplane is k-1 dimensional. We can write a function to represent this idea:

$$f(\mathbf{X}_j) = y_j = \begin{cases} 1, & \text{if} \sum_{i=1}^{m} \beta_i x_{ij} \geq \beta_0 \\ 0, & \text{if} \sum_{i=1}^{m} \beta_i x_{ij} < \beta_0 \end{cases}$$

β_0 is simply a threshold. The logic behind this function is that we can dichotomize the data based on the hyperplane the $\beta_0 + \sum_{i=1}^{m} \beta_i x_{ij} = 0$ where β_0 is the threshold. Note that y_i is equal to 0 or 1. Thus we can rewrite the function into another simpler format:

$$\begin{aligned} f(\mathbf{X}_j) &= \text{sign}\left[\left(\sum_{i=1}^{m} \beta_i x_{ij}\right) - \beta_0\right] \\ &= \text{sign}\left[\left(\sum_{i=1}^{m} \beta_i x_{ij}\right) + \beta_0 \cdot (-1)\right] \\ &= \text{sign}\left(\sum_{i=0}^{m} \beta_i x_{ij}\right) \end{aligned}$$

Now the job of a perceptron is to perform $f(x_i)$ to correctly separate the data. In other words, by learning the data, a perceptron should be able to find a hyperplane $\widehat{\beta}_0 + \sum_{i=1}^{m} \widehat{\beta}_i x_{ij} = 0$. Therefore, we can write a

function that a perceptron is going to perform:

$$\begin{aligned}\widehat{f(\mathbf{X}_j)} = \widehat{y_j} &= \text{sign}\left(\sum_{i=0}^{m} \widehat{\beta}_i x_i\right) \\ &= \text{sign}\left(\mathbf{X}^T \cdot \widehat{\boldsymbol{\beta}}_{\text{PLA}}\right)\end{aligned}$$

In summary, a perceptron can find $\widehat{\boldsymbol{\beta}}_{\text{PLA}}$ so that $\widehat{y_j}$ is equal to y_j. In this sense, a perceptron is a technique of nonstatistical, nonparametric estimation. This is why a perceptron is a machine learning algorithm, although we use the language of statistics to present it.

2 The Algorithm

An algorithm implementing the idea of perceptron is a trial-and-error approach. This means that the perceptron learning algorithm makes mistakes until it finds a separating hyperplane. Assume

$$\mathbf{X}^T_{(n+1)\times(m+1)} = \begin{bmatrix} x_{00} & x_{01} & x_{02} & \cdots & x_{0m} \\ x_{10} & x_{11} & x_{12} & \cdots & x_{1m} \\ x_{20} & x_{21} & x_{32} & \cdots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n0} & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}, \widehat{\boldsymbol{\beta}}_{(m+1)\times 1} = \begin{bmatrix} \widehat{\beta}_0 \\ \widehat{\beta}_1 \\ \widehat{\beta}_2 \\ \vdots \\ \widehat{\beta}_m \end{bmatrix}, \mathbf{Y}_{(n+1)\times 1} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Suppose a perceptron learning algorithm will run t times until $\widehat{\boldsymbol{\beta}}_{\text{PLA}}$ is obtained. Denote $\mathbf{X}_{(t)}$ the t-th input of the data: $[x_{0(t)}, x_{1(t)}, x_{2(t)}, \cdots, x_{k(t)}]$, and the size of the training set is n. So $0 \leq t \leq n$. Let $\widehat{\boldsymbol{\beta}}_{(\boldsymbol{t})} = [\beta_{0(t)}, \beta_{1(t)}, \beta_{2(t)}, \cdots, \beta_{k(t)}]^T$ and $\mathbf{Y}_{(t)} = \text{y}_t$. Let $x_{i(t)} = 1$. The initial value of $\beta_{0(0)}$ is set to be 0. We specify a perceptron learning algorithm as follows:

Data: X, Y
Result: $\widehat{\boldsymbol{\beta}}$
initialization;
while $t \leq n$ **do**
  if $\left[sign\left(\mathbf{X}_{(t)}\widehat{\boldsymbol{\beta}}_{(t)}\right) \cdot \mathbf{Y}_{(t)}\right] > 0$ **then**
    $\widehat{\boldsymbol{\beta}}_{(t+1)} = \widehat{\boldsymbol{\beta}}_{(t)}$;
  else
    $\widehat{\boldsymbol{\beta}}_{(t+1)} = \widehat{\boldsymbol{\beta}}_{(t)} + \left(\mathbf{Y}_{(t)} \cdot \mathbf{X}_{(t)}\right)$;
  end
end
return $\widehat{\boldsymbol{\beta}}_{\text{PLA}} = \widehat{\boldsymbol{\beta}}_{(t+1)}$

3 Implementation in R

The implementation of the perceptron algorithm is simple in R. As we can see the algorithm above, the code will be very straightforward.

```
perceptron <- function(df,X,Y){
y <- Y
len_X <- length(X)
len_df <- nrow(df)
V_X <- c(rep(1,len_df))

##---- To make the vector of X: V_X ----##
for (i in 1:len_X)
{
  x_current <- X[i]
  V_X <- c(V_X, eval(parse(text=paste("df$",x_current,sep=""))))
}

##---- To make the matrix of X: M_X -----##
M_X <- matrix(V_X,ncol=(length(X)+1), byrow=FALSE)

##---- To make the initial matrix of coefficients: B -----##
B <- matrix(c(0,rep(1,len_X)),ncol=1, byrow=FALSE)

##---- The Algorithm ----##
do_not_match <- 1
while(do_not_match == 1)
{
  do_not_match <- 0
  for (i in 1:nrow(df))
  {
    do_not_match <- 0
    if ( ((M_X[i,] %*% B)*(eval(parse(text=paste("df$",y,sep=""))))[i])<=0 )
       {
        B <- B + ( M_X[i,]*(eval(parse(text=paste("df$",y,sep=""))))[i])
        do_not_match <- 1
       }
  }
}
return(B)
}
```

To run the this function, we can simple call

```
Perceptron(df,X,Y)
```

The parameter of "df" is the data that we would like to train, "X" is a list of the features we include, and "Y" is a vector of the labels corresponding to each data point. The function will return a vector "B," of which the first component is $\widehat{\boldsymbol{\beta}}_0$ and the rest are $\widehat{\boldsymbol{\beta}}_i$ where $i = (1, 2, 3, ..., m)$.