Machine Learning 2019

袁欣

2019 年 4 月 25 日

1 神经网络

1.1 西瓜数据 3.0

读入西瓜数据 3.0，对分类变量进行独热编码（One-hot encoding）。

```
wmdaorign <- read.csv("西瓜数据3.0.csv")
wmdaorign <- wmdaorign[, -1]
dmy <- dummyVars(~., data = wmdaorign)
wmda <- data.frame(predict(dmy, newdata = wmdaorign))
```

- 数据展示

色泽浅白	色泽青绿	色泽乌黑	根蒂蜷缩	根蒂稍蜷	根蒂硬挺	敲声沉闷	敲声清脆	敲声浊响
0	1	0	1	0	0	0	0	1
0	0	1	1	0	0	1	0	0
0	0	1	1	0	0	0	0	1
0	1	0	1	0	0	1	0	0
1	0	0	1	0	0	0	0	1

纹理模糊	纹理清晰	纹理稍糊	脐部凹陷	脐部平坦	脐部稍凹	触感软粘	触感硬滑	密度
0	1	0	1	0	0	0	1	0.697
0	1	0	1	0	0	0	1	0.774
0	1	0	1	0	0	0	1	0.634
0	1	0	1	0	0	0	1	0.608
0	1	0	1	0	0	0	1	0.556

含糖率	好瓜否	好瓜是
0.460	0	1
0.376	0	1
0.264	0	1
0.318	0	1
0.215	0	1

1.2 神经网络简介

神经网络或连接系统是由构成动物大脑的生物神经网络模糊地启发式计算系统。神经网络本身不是算法，而是许多不同机器学习算法的框架，它们协同工作并处理复杂的数据输入。现在神经网络已经用于各种任务，包括计算机视觉、语音识别、机器翻译、社交网络过滤，游戏板和视频游戏以及医学诊断。

1.2.1 逻辑回归的反向传播算法推导

激活函数为 sigmoid 函数的单一神经元模型就是逻辑回归模型。为了通俗易懂的推导反向传播算法，我利用最简单的逻辑回归模型实现，以下为推导过程。

- 符号解释

$$X = [x^{(1)}, x^{(2)}, ..., x^{(m)}]_{n\times m}$$
$$Y = [y^{(1)}, y^{(2)}, ..., y^{(m)}]_{1\times m}$$

其中上标表示第 i 个样本，共 m 个样本，n 个指标。

$$W \in R^{n\times 1} \qquad b \in R$$

W 为参数向量，由于是 n 个指标，所以 W 为 $n \times 1$ 的矩阵。b 为截距项。

- 逻辑回归表达式

$$\hat{y}^{(i)} = \sigma(W^T x^{(i)} + b)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- 损失函数

$$L(\hat{y}, y) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y}))$$

- 代价函数

$$J(W,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})$$

我们的目标是利用梯度下降法最小化代价函数。反向传播法其实就是根据链式法则计算偏导数的，下面我们进行推导。

当 $n=2$ 时，我们对单个样本进行推导：

$$\begin{aligned}z&=w_1x_1+w_2x_2+b\\\hat{y}&=a=\sigma(z)\\L(a,y)\end{aligned}$$

$$dz=\frac{dL}{dz}=\frac{dL(a,y)}{dz}=\frac{dL}{da}\frac{da}{dz}=a-y$$

其中

$$\begin{aligned}\frac{dL}{da}&=-\frac{y}{a}+\frac{1-y}{1-a}\\\frac{da}{dz}&=a(1-a)\end{aligned}$$

显然

$$\begin{aligned}\frac{dL}{dw_1}&=x_1dz=x_1(a-y)\\\frac{dL}{dw_2}&=x_2dz=x_2(a-y)\\db&=dz=a-y\end{aligned}$$

当 $n=2$ 时，有 m 个样本时我们仅需要 `for` 循环计算 m 个偏导数的和即可。其实可以看出最后得到的偏导数与我们在第二章中的一样，这里的推导过程仅为了后续更容易理解多层神经网络的反向传播算法。

- 矩阵表示

$$\begin{aligned}dZ&=[dz^{(1)},dz^{(2)},.....,dz^{(m)}]\\A&=[a^{(1)},a^{(2)},.....,a^{(m)}]\end{aligned}$$

$$dZ=A-Y$$

则

$$\begin{aligned}dW&=\frac{1}{m}X\,dZ^T\\db&=\frac{1}{m}\sum_{i=1}^{m}dZ^{(i)}\end{aligned}$$

1.2.2 单隐层神经网络

根据上一节的推导过程，我们推导了单隐层神经网络向前传播求 $\hat{Y}$, 以及反向传播求偏导数的过程。

- 符号解释

上角标 $^{[i]}$ 表示第 i 层，其中输入层 i 为 0。

- 前向传播

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= \sigma(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= \sigma(Z^{[2]})
\end{aligned}
$$

其中 $W^{[i]}$ 为第 $i-1$ 层到第 i 层的参数矩阵，为 $l^{[i]} \times l^{[i-1]}$（l 为层的神经元个数）。

- 反向传播

$$
\begin{aligned}
dZ^{[2]} &= A^{[2]} - Y \\
dW^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]T} \\
db^{[2]} &= \frac{1}{m} \sum_{i=1}^{m} dZ_{(i)}^{[2]} \\
dZ^{[1]} &= W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} X^{T} \\
db^{[1]} &= \frac{1}{m} \sum_{i=1}^{m} dZ_{(i)}^{[1]}
\end{aligned}
$$

- 随机初始化

初始化参数时，不能像逻辑回归一样直接将 W 设为 0。如果这样，那么所有隐层单元学习到的参数将是一样的（可通过前向传播、反向传播过程证明）。在初始化的过程中也要经可能的使 W 较小，这样可以加快学习速率。

- 代码实现

```
InitParam <- function(nx, nh, ny, seed = 2){
  #
  set.seed(seed)
  W1 <- matrix(rnorm(nx * nh), ncol = nx) * 0.01
```

```r
  b1 <- matrix(rep(0, nh), ncol = 1)
  W2 <- matrix(rnorm(ny * nh), ncol = nh) * 0.01
  b2 <- matrix(rep(0, ny), ncol = 1)
  return(list(W1 = W1, b1 = b1, W2 = W2, b2 = b2))
}
InitParamDeep <- function(Ldims, seed = 2){
  #
  set.seed(seed)
  Paramlist <- list()
  L = length(Ldims)
  for(i in 2:L){
    Wi <- matrix(rnorm(Ldims[i-1] * Ldims[i]), ncol = Ldims[i-1]) * 0.01
    bi <- matrix(rep(0, Ldims[i]), ncol = 1)
    Paramlist[[i-1]] <- list(Wi = Wi, bi = bi)
  }
  return(Paramlist)
}

LinearForward <- function(A, W, b){
  #
  Z <- W %*% A + matrix(rep(b, ncol(A)), ncol = ncol(A))
  return(list(Z = Z, A = A, W = W, b = b))
}
sigmoid <- function(Z){
  #
  A <- 1 / (1 + exp(-Z))
  return(A)
}
LinActForward <- function(Aprev, W, b){
  #
  templist <- LinearForward(Aprev, W, b)
  Z <- templist$Z
  A <- sigmoid(Z)
  return(list(Z = Z, A = A, W = W, b = b, Aprev = Aprev))
}

LModelForward <- function(X, Paramlist){
  #
  cacheslist <- list()
```

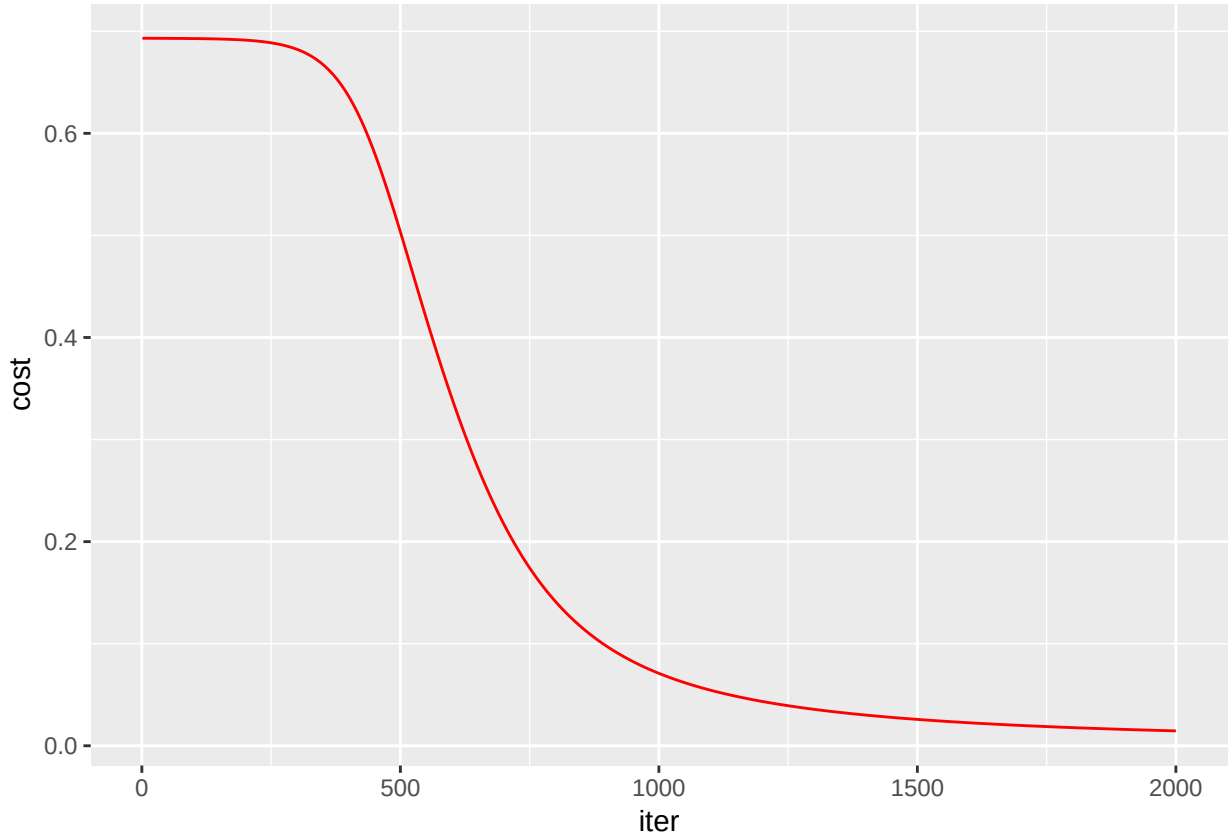
```
  A <- X
  L <- length(Paramlist)

  for(i in 1:L){
    Aprev <- A
    W <- Paramlist[[i]]$W
    b <- Paramlist[[i]]$b
    templist <- LinActForward(Aprev, W, b)
    A <- templist$A
    cacheslist[[i]] <- templist
  }
  return(list(A = A, cacheslist = cacheslist))
}

ComputeCost <- function(AL, Y){
  #
  m <- length(Y)
  cost <- -1/m * sum(Y * log(AL) + (1-Y) * log(1-AL))
  return(cost)
}

LinActBackward <- function(dA, cache){
  #
  APrev <- cache$Aprev
  W <- cache$W
  b <- cache$b
  Z <- cache$Z
  m <- ncol(APrev)
  #
  s <- 1/(1+exp(-Z))
  dZ <- dA * (s * (1 - s))
  dAprev <- t(W) %*% dZ
  dW <- 1/m * dZ %*% t(APrev)
  db <- 1/m * rowSums(dZ)
  return(list(dAprev = dAprev, dW = dW, db = db))
}
LModelBackward <- function(AL, Y, cacheslist){
  #
  grads <- list()
```

```
  L <- length(cacheslist)
  m <- length(AL)
  #
  dAL <- -Y/AL + (1-Y)/(1-AL)
  dA <- dAL

  for(i in L:1){
    #
    grads[[i]] <- LinActBackward(dA, cacheslist[[i]])
    dA <- grads[[i]]$dAprev

  }
  return(grads)
}
UpdateParams <- function(Paramlist, grads, learnrate = 0.1){
  #
  L <- length(Paramlist)
  for(i in 1:L){
    Paramlist[[i]]$Wi <- Paramlist[[i]]$Wi - learnrate * grads[[i]]$dW
    Paramlist[[i]]$bi <- Paramlist[[i]]$bi - learnrate * grads[[i]]$db
  }
  return(Paramlist)
}

NNmodel <- function(X, Y, Ldims, seed = 2, learnrate = 0.1, numiter = 10000, printcost = FALSE){
  #
  costs <- vector(length = numiter)

  #
  Paramlist <- InitParamDeep(Ldims, seed = seed)
  #
  for(i in 1:numiter){
    #
    Forward <- LModelForward(X, Paramlist)
    AL <- Forward$A
    cacheslist <- Forward$cacheslist
    #
    cost <- ComputeCost(AL, Y)
    #
```

```
    grads <- LModelBackward(AL, Y, cacheslist)
    #
    Paramlist <- UpdateParams(Paramlist, grads, learnrate)
    if(printcost == TRUE & i%%100==0){
      print(paste("Cost after iteration", i, ":", cost))
    }
    costs[i] <- cost
  }
  return(list(Paramlist = Paramlist, costs = costs))
}

NNmodelSGD <- function(X, Y, Ldims, seed = 2, learnrate = 0.1, numiter = 100, printcost = FALSE){
  #
  m <- ncol(Y)
  costs <- vector(length = numiter*m)

  #
  Paramlist <- InitParamDeep(Ldims, seed = seed)
  Xall <- X
  Yall <- Y
  #
  for(i in 1:numiter){
    #
    for(j in 1:m){
      X <- Xall[, j, drop = FALSE]
      Y <- Yall[, j, drop = FALSE]
      Forward <- LModelForward(X, Paramlist)
      AL <- Forward$A
      cacheslist <- Forward$cacheslist
      #
      cost <- ComputeCost(AL, Y)
      #
      grads <- LModelBackward(AL, Y, cacheslist)
      #
      Paramlist <- UpdateParams(Paramlist, grads, learnrate)
      if(printcost == TRUE & i%%100==0){
        print(paste("Cost after iteration", i, ":", cost))
      }
      costs[m*(i-1) + j] <- cost
```

```
    }

  }
  return(list(Paramlist = Paramlist, costs = costs))
}
NNpredict <- function(NewX, NewY, Model){
  #
  PreY <- LModelForward(NewX, Model$Paramlist)$A
  PreY <- ifelse(PreY >= 0.5, 1, 0)
  tb <- table(PreY, NewY)
  accuracy <- sum(diag(tb)) / sum(tb)
  return(list(PreY = PreY, tb = tb, accuracy = accuracy))
}
```

```
X <- t(as.matrix(wmda[, 1:19]))
Y <- t(as.matrix(wmda[, 21]))

set.seed(seed = 1)
ind <- sample(ncol(Y), 12)
X.train <- X[, ind]
Y.train <- Y[, ind, drop = FALSE]
X.test <- X[, -ind]
Y.test <- Y[, -ind, drop = FALSE]

Model <- NNmodel(X.train, Y.train, Ldims = c(19, 10, 1), seed = 2, learnrate = 0.1, numiter = 2000)

# plot cost
costs <- data.frame(iter = 1:2000, cost = Model$costs)
ggplot(data = costs, aes(x = iter, y = cost)) + geom_line(color = "red")
```




```
# Predict
Predict.train <- NNpredict(X.train, Y.train, Model)
print(paste("The Train accuracy is : ", Predict.train$accuracy * 100, "%"))
```

```
## [1] "The Train accuracy is :  100 %"
```

```
print("The confusion matrix is : ")
```

```
## [1] "The confusion matrix is : "
```

```
Predict.train$tb
```

```
##     NewY
## PreY 0 1
##    0 6 0
##    1 0 6
```

```
# Test
Predict.test <- NNpredict(X.test, Y.test, Model)
print(paste("The Test accuracy is : ", Predict.test$accuracy * 100, "%"))
```

```
## [1] "The Test accuracy is :  80 %"
```

```
print("The confusion matrix is : ")
```

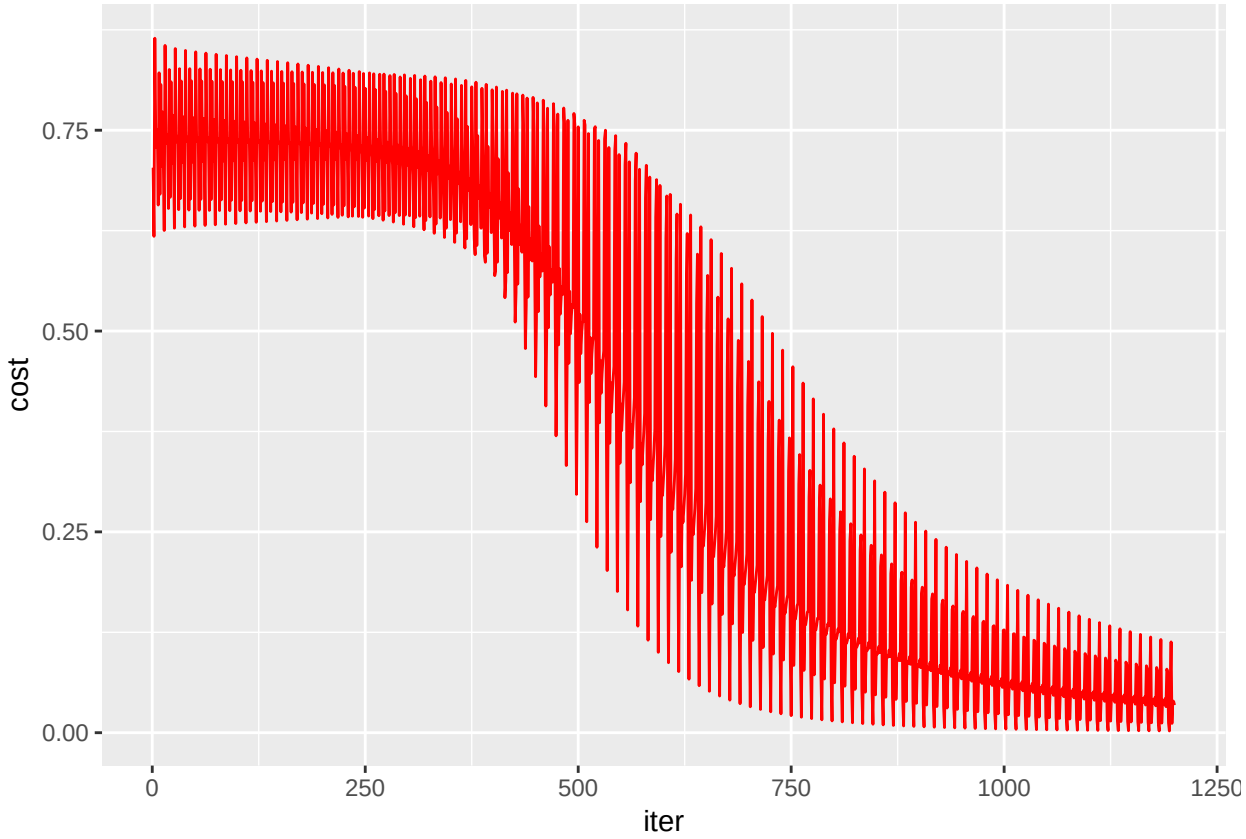
```
## [1] "The confusion matrix is : "
```

```
Predict.test$tb
```

```
##      NewY
## PreY 0 1
##    0 2 0
##    1 1 2
```

- 随机梯度下降算法

```
ModelSGD <- NNmodelSGD(X.train, Y.train, Ldims = c(19, 10, 1),
                       seed = 2, learnrate = 0.1, numiter = 100)

# plot cost
costs <- data.frame(iter = 1:(12 * 100), cost = ModelSGD$costs)
ggplot(data = costs, aes(x = iter, y = cost)) + geom_line(color = "red")
```



```r
# Predict
Predict.train <- NNpredict(X.train, Y.train, ModelSGD)
print(paste("The Train accuracy is : ", Predict.train$accuracy * 100, "%"))
```

```
## [1] "The Train accuracy is :  100 %"
```

```r
print("The confusion matrix is : ")
```

```
## [1] "The confusion matrix is : "
```

```r
Predict.train$tb
```

```
##      NewY
## PreY 0 1
##    0 6 0
##    1 0 6
```

```r
# Test
Predict.test <- NNpredict(X.test, Y.test, ModelSGD)
print(paste("The Test accuracy is : ", Predict.test$accuracy * 100, "%"))
```

```
## [1] "The Test accuracy is :  80 %"
```

```r
print("The confusion matrix is : ")
```

```
## [1] "The confusion matrix is : "
```

```r
Predict.test$tb
```

```
##      NewY
## PreY 0 1
##    0 2 0
##    1 1 2
```

1.3 小结

神经网络是一个庞大的体系，本文仅展示了经典的方法——反向传播算法。后续内容可能涉及到的有如何划分训练集与测试集、方差偏差法判断模型拟合程度、正则化（L_i 范数、随机失活等）、小批量梯度下降法、RMSprop 算法、Adam 优化算法、超参数的调试等内容。