penalized_logistic_regression.R

Erica Zhou

Nov 16, 2022

```r
Evaluate <- function(true_label, pred_label) {
  #  Compute the 0-1 loss between two vectors
  #
  #  @param true_label: A vector of true labels with length n
  #  @param pred_label: A vector of predicted labels with length n
  #  @return: fraction of points get misclassified

  #####################################################################
  #  TODO                                                             #
  #####################################################################

  error <- sum(true_label != pred_label)/length(true_label)

  #####################################################################
  #                       END OF YOUR CODE                            #
  #####################################################################
  return(error)
}



Predict_logis <- function(data_feature, beta, beta0, type) {
  # Predict by the logistic classifier.
  #
  # Note: n is the number of examples
  #       p is the number of features per example
  #
  # @param data_feature: A matrix with dimension n x p, where each row corresponds to
  #   one data point.
  # @param beta: A vector of coefficients with length equal to p.
  # @param beta0: the intercept.
  # @param type: a string value within {"logit", "prob", "class"}.
  # @return: A vector with length equal to n, consisting of
  #   predicted logits,         if type = "logit";
  #   predicted probabilities,  if type = "prob";
  #   predicted labels,         if type = "class".

  n <- nrow(data_feature)
  pred_vec <- rep(0, n)

  #####################################################################
  #  TODO                                                             #
```

```r
  ####################################################################

  if (type == "logit"){
    pred_vec <- as.vector(beta%*%data_feature + beta0)
  }

  if (type == "prob"){
    pred_vec <- as.vector((exp(
      beta0 + data_feature%*%beta))/(1 + exp(
        beta0 + data_feature%*%beta)))
  }

  if (type == "class"){
    pred_vec <- as.numeric(((exp(
      (beta0 + data_feature%*%beta))) /(1 + exp(
        (beta0 + data_feature%*%beta))) >= .5))
  }

  #sample.data  sample.data <- matrix (c(1,3,4,1, 5,8,9,6, 1,3,3,8), 4,3)

  ####################################################################
  #                       END OF YOUR CODE                           #
  ####################################################################

  return(pred_vec)
}



Comp_gradient <- function(data_feature, data_label, beta, beta0, lbd) {
  # Compute and return the gradient of the c
  #
  # Note: n is the number of examples
  #       p is the number of features per example
  #
  # @param data_feature: A matrix with dimension n x p, where each row corresponds to
  #   one data point.
  # @param data_label: A vector of labels with length equal to n.
  # @param beta: A vector of coefficients with length equal to p.
  # @param beta0: the intercept.
  # @param lbd: the regularization parameter
  #
  # @return: a (p+1) x 1 vector of gradients, the first coordinate is the gradient
  #   w.r.t. the intercept.

  n <- nrow(data_feature)
  p <- ncol(data_feature)
  grad <- rep(0, 1 + p)

  ####################################################################
  # TODO:                                                            #
  ####################################################################
```

```r
  grad0 <- exp(beta0)/(1 + exp(beta0))/n
  grad1 <- 1/n*((-data_label + t(exp(beta0 + data_feature%*%beta)/(1 + exp(
    beta0 + data_feature%*%beta)))) %*%data_feature + lbd%*%beta)
  grad <- c(grad0, grad1)

  ####################################################################
  #                       END OF YOUR CODE                           #
  ####################################################################
  return(grad)
}


Comp_loss <- function(data_feature, data_label, beta, beta0, lbd) {
  # Compute and return the loss of the penalized logistic regression
  #
  # Note: n is the number of examples
  #       p is the number of features per example
  #
  # @param data_feature: A matrix with dimension n x p, where each row corresponds to
  #   one data point.
  # @param data_label: A vector of labels with with length equal to n.
  # @param beta: A vector of coefficients with length equal to p.
  # @param beta0: the intercept.
  # @param lbd: the regularization parameter
  #
  # @return: a value of the loss function

  ####################################################################
  # TODO:                                                            #
  ####################################################################
  n <- length(data_label)
  p <- exp(beta0 + data_feature%*%beta)/(1 + exp(beta0 + data_feature%*%beta))

  loss <- as.numeric((data_label%*%log(p)+(1-data_label)%*%log(1-p))/(-n)
                     + lbd/2*(norm(beta,"2"))^2)

  ####################################################################
  #                       END OF YOUR CODE                           #
  ####################################################################
  return(loss)
}




Penalized_Logistic_Reg <- function(x_train, y_train, lbd, stepsize, max_iter) {
  # This is the main function to fit the Penalized Logistic Regression
  #
  # Note: n is the number of examples
  #       p is the number of features per example
  #
  # @param x_train: A matrix with dimension n x p, where each row corresponds to
  #   one training point.
```

```r
# @param y_train: A vector of labels with length equal to n.
# @param lbd: the regularization parameter.
# @param stepsize: the learning rate.
# @param max_iter: a positive integer specifying the maximal number of
#   iterations.
#
# @return: a list containing four components:
#   loss: a vector of loss values at each iteration
#   error: a vector of 0-1 errors at each iteration
#   beta: the estimated p coefficient vectors
#   beta0: the estimated intercept.

p <- ncol(x_train)

# Initialize parameters to 0
beta_cur <- rep(0, p)
beta0_cur <- 0

# Create the vectors for recording values of loss and 0-1 error during
# the training procedure
loss_vec <- rep(0, max_iter)
error_vec <- rep(0, max_iter)

####################################################################
# TODO:                                                            #
# Modify this section to perform gradient descent and to compute   #
# losses and 0-1 errors at each iterations.                        #
####################################################################


for (i in 1:max_iter){

  beta_cur <- beta_cur - stepsize * Comp_gradient(x_train, y_train,
                                                  beta_cur, beta0_cur,
                                                  lbd)[-1]
  beta0_cur <- beta0_cur - stepsize * Comp_gradient(x_train, y_train,
                                                    beta_cur, beta0_cur,
                                                    lbd)[1]

  # 0-1 error: the true vs. the predicted
  # First we need the predicted label for each beta and beta0 starting from (0,0)
  y_pred <- Predict_logis(x_train,beta_cur,beta0_cur, "class" )
  # Second we calculate and add the 0-1 error for each iteration to the vector
  error_vec[i] <- Evaluate(y_train, y_pred)
  # loss of each iteration
  loss_vec[i] <- Comp_loss(x_train, y_train, beta_cur, beta0_cur, lbd)

}

####################################################################
#                        END OF YOUR CODE                          #
####################################################################
```

```
  return(list("loss" = loss_vec, "error" = error_vec,
              "beta" = beta_cur, "beta0" = beta0_cur))
}
```