XGBoost

2022-12-04

Data Generation

```r
set.seed(1)
n_all <- 120
n_pred = 10
R = diag(n_pred)
mu <- rep(0, n_pred)
X = mvtnorm::rmvnorm(n_all, mean = mu, sigma = R)
df = 10
median = df - 0.7

sum_sqr = function(vec) {
  return(sum(vec^2))
}
sum_X = apply(X,1,sum_sqr)
y = ifelse(sum_X>=median,1,0)

X_test = X[101:120,]
y_test = y[101:120]
X_train = X[1:100,]
y_train = y[1:100]
n = nrow(X_train)
```

Algorithm

For classification problem, likelihood function is $L = -y_i log(p_i) + (1 - y_i) log(1 - p_i)$, the gradient of loss function is $g_i = -(y_i - p_i)$ and the Hession function is $h_i = p_i * (1 - p_i)$.

Similarity score $= \frac{(\sum g_i)^2}{\sum h_i + \lambda}$;

Output value for each node $= \frac{\sum g_i}{\sum h_i + \lambda}$;

Gain = Left Similarity score + Right Similarity score - Parent node Similarity score;

For each tree, find the (variable, threshold) combination that maximize the Gain.

We initialized the probability to be 0.5 so that the logodds = 0.

Once a tree is developed, we updated the predicted probability by updating the logodds = logodds_previous + learning_rate * output_value_of_the_node. Then, we convert the logodds back to probability $p = exp(logodds)/(1 + exp(logodds)$. Using the updated probability, we can construct the second tree.

We build tree until convergence or the maximum number of tree reached.

```r
similarity = function(index, true_y, pred_prob, lambda = 1){
  true_y = true_y[index]
  pred_prob = pred_prob[index]
  score = sum(true_y-pred_prob)^2 / (sum(pred_prob * (1-pred_prob)) + lambda)

  return(score)
}

output = function(index, true_y, pred_prob, lambda = 1){
  true_y = true_y[index]
  pred_prob = pred_prob[index]
  score = sum((true_y-pred_prob)) / (sum(pred_prob * (1-pred_prob)) + lambda)

  return(score)
}
```

```r
XGB = function(X_inp, y_inp, n_tree=100, prob_initial = rep(0.5,n), lambda = 1, epsilon = 0.1) {
  similarity_all = similarity(seq(1:n), y, pred_prob = prob_initial, lambda)

  tree = list()
  prob = prob_initial
  for (t in 1:n_tree) {
    Gain_pred = c()
    threshold = c()
    out_left = c()
    out_right = c()
    for (i in 1:n_pred) {
      x = X_inp[,i]

      thre_list = sort(x)
      Gain = -100

      for (j in 1:(n-1) ) {
        thres = thre_list[j]
        left_ind = which(x <= thres)
        right_ind = which(x > thres)

        similarity_left = similarity(left_ind, y, pred_prob = prob, lambda)
        similarity_right = similarity(right_ind, y, pred_prob = prob, lambda)
        Gain_new = similarity_left + similarity_right - similarity_all

        if (Gain_new > Gain) {
          Gain = Gain_new
          Gain_pred[i] = Gain_new
          threshold[i] = thres
          out_left[i] = output(left_ind, y_inp, pred_prob = prob, lambda = 1)
          out_right[i] = output(right_ind, y_inp, pred_prob = prob, lambda = 1)
        }
      }
    }
    which_index = min(which(Gain_pred == max(Gain_pred)))
    tree[[t]] = list(variable = which_index, threshold = threshold[which_index],
         out_left = out_left[which_index], out_right = out_right[which_index])
    # update_prediction
    output_leaf = ifelse(X_inp[,tree[[t]]$variable] <= tree[[t]]$threshold, tree[[t]]$out_left, tree[[t
    pred_logodds = log(prob/(1-prob)) + epsilon*output_leaf
    prob = exp(pred_logodds) / (1+exp(pred_logodds))
  }


  return(tree)
}
```

Prediction

```r
prediction = function(X_inp, tree, epsilon = 1) {
  pred_logodds = 0
  for (i in 1:length(tree)) {
    output_leaf = ifelse(X_inp[,tree[[i]]$variable] <= tree[[i]]$threshold, tree[[i]]$out_left, tree[[i
    pred_logodds = pred_logodds + epsilon*output_leaf
  }
  prob = exp(pred_logodds) / (1+exp(pred_logodds))
  out = ifelse(prob >= 0.5, 1, 0)
}
```

Train and test

```r
trees = XGB(X_train, y_train, n_tree=100, prob_initial = rep(0.5,n), lambda = 0, epsilon = 0.1)

# training accuracy
y_pred = prediction(X_train, trees, epsilon = 0.1)
print(paste('training accuracy:', sum(y_train == y_pred)/n))
```

```
## [1] "training accuracy: 0.95"
```

```r
# testing accuracy
y_pred = prediction(X_test, trees, epsilon = 0.1)
print(paste('test accuracy:', sum(y_test == y_pred)/length(y_test)))
```

```
## [1] "test accuracy: 0.8"
```