LB3 机器学习概论

PB19151769　　马宇骁

目录

1 实验要求

1. 完成决策树（回归树）的算法
2. 完成 XGBoost 的算法
3. 书写你的报告

- 禁止使用 sklearn 或者其他的机器学习库，你只被允许使用 numpy, pandas, matplotlib, 和 Standard Library, 你需要从头开始编写这个项目。
- 你可以和其他同学讨论，但是你不可以剽窃代码，我们会用自动系统来确定你的程序的相似性，一旦被发现，你们两个都会得到这个项目的零分。

2 实验原理

2.1 决策树（回归树）

决策树是一种基本的分类与回归方法。决策树由结点 (node) 和有向边 (diredcted edge) 组成。结点有两种类型：内部结点 (internal node) 和叶结点 (leaf node)。内部结点表示一个特征或属性，叶结点表示一个类别或者某个值。用决策树做分类或回归任务时，从根节点开始，对样本的某一特征进行测试，根据测试结果，将样本分配到其子结点；这时，每一个子节点对应着该特征的一个取值。如此递归地对样本进行测试并分配，直至到达叶结点。

2.1.1 理论解释

假设 X 和 Y 分别为输入和输出变量，并且 Y 是连续变量，给定训练数据集：

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

一个回归树对应着输入空间（即特征空间）的一个划分以及在划分的单元上的输出值。假设已将输入空间划分为 M 个单元 $R_1, R_2, \ldots, R_M$，并且在每个单元 R_i 上有一个固定的输出值 c_i ，于是回归树模型可以表示为：

$$f(x) = \sum_{i=1}^{M} c_i I\left(x \in R_i\right)$$

用平方误差 $\sum_{x_i \in R_m} \left(y_i - f\left(x_i\right)\right)^2$ 来表示回归树对于训练数据的预测误差，用平方误差最小的准则求解每个单元上的最优输出值。易知，单元 R_i 上的 c_i 的最优值 $\hat{c}_i$ 是 R_i 上的所有输入实例 x_j 对应的输出 y_j 的均值，即：

$$\hat{c}_i = \operatorname{ave}\left(y_j \mid x_j \in R_i\right)$$

2.1.2 划分点

CART 回归树采用启发式的方法对输入空间进行划分，选择第 j 个变量 $x^{(j)}$ 和它取的值 s，作为切分变量（splitting variable）和切分点（splitting point），并定义两个区域：

$$R_1(j,s) = \left\{x \mid x^{(j)} \leq s\right\} \text{ 和} R_2(j,s) = \left\{x \mid x^{(j)} > s\right\}$$

然后寻找最优切分变量 j 和最优切分点 s。即求解：

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}\left(y_i - c_1\right)^2 + \min_{c_2}\sum_{x_2\in R_2(j,s)}\left(y_i - c_2\right)^2\right]$$

对固定输入变量 j 可以找到最优切分点 s。

2.1.3 叶节点的输出值

用选定的最优切分变量 j 和最优切分点 s 划分区域并决定相应的输出值：

$$\hat{c}_1 = \text{ave}\left(y_i \mid x_i \in R_1(j,s)\right) \text{ 和} \hat{c}_2 = \text{ave}\left(y_i \mid x_i \in R_2(j,s)\right)$$

遍历所有输入变量，找到最优的切分变量 j，构成一个对（j, s）。依此将输入空间划分为两个区域。接着，对每个区域重复上述划分过程，直到满足停止条件为止。对于已经划分输入空间好的，模型表示为：

$$f(x) = \sum_{i=1}^{M}\hat{c}_i I\left(x \in R_i\right)$$

2.1.4 Obj

$$Obj^{(t)} = \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}\left(H_j + \lambda\right)w_j^2\right] + \gamma T$$

- 权重

$$\frac{\partial Obj^{(t)}}{\partial w} = \sum_{j=1}^{T}\left[G_j + \left(H_j + \lambda\right)w_j\right] = 0$$

且对每一个 w_j 求偏导也等于 0.

$\Longrightarrow$

$$w_j^* = -G_j(H_j + \lambda)^{-1}$$

- Obj

$$
\begin{aligned}
Obj^{(t)*} &= \sum_{j=1}^{T}\left[G_j w_j^* + \frac{1}{2}\left(H_j+\lambda\right)w_j^{*2}\right] + \gamma T \\
&= \sum_{j=1}^{T}\left[-G_j^2(H_j+\lambda)^{-1} + \frac{1}{2}G_j^2(H_j+\lambda)^{-1}\right] + \gamma T \\
&= (\gamma - \frac{1}{2}G_j^2(H_j+\lambda)^{-1})T
\end{aligned}
$$

2.2 XGBoost

基于 boosting 集成思想的加法模型，训练时采用前向分布算法进行贪婪的学习，每次迭代都学习一棵 CART 树来拟合之前 t-1 棵树的预测结果与训练样本真实值的残差。XGBoost（eXtreme Gradient Boosting）极致梯度提升，是基于 GBDT 的一种算法。

2.2.1 原理

提升树模型可以表示为决策树的加法模型:

$$
f_M(x) = \sum_{m=1}^{M} T_m(x)
$$

其中，$T_m(x)$ 表示第 m 决策树，M 为树的个数。

2.2.2 初始化弱学习器

$$
f_0(x) = \arg\min_c \sum_{l=1}^{N} L\left(y_i, c\right)
$$

假设取损失函数为平方损失。因为平方损失函数是一个凸函数，直接对 c 求导：

$$
\sum_{i=1}^{N}\frac{\partial L\left(y_i,c\right)}{\partial c} = \sum_{i=1}^{N}\frac{\partial\left(\frac{1}{2}\left(y_i-c\right)^2\right)}{\partial c} = \sum_{i=1}^{N}\left(c-y_i\right)
$$

等于 0 时，得:

$$
c = \sum_{i=1}^{N} y_i / N
$$

所以初始化时，c 取值为所有训练样本标签值的均值。此时得到初始学习器：$f_0(x) = c$

2.2.3 迭代训练

对每个样本 i=1,2, $\cdots$, N ，计算负梯度，即残差:

$$
r_{mi} = -\left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right]_{fx)=f_{m-1}(x)}
$$

将上步得到的残差 r_{mi} 作为样本新的真实值，并将数据 $(x_i, r_{im}), i = 1, 2, \cdots, N$ 作为下树的叶子结点的个数。

对 j=1,2, $\cdots$, J 个叶子结点，计算最佳拟合值：

$$c_{mj} = \arg\min_{c} \sum_{x_i \in R_{mj}} L\left(y_i, f_{m-1}\left(x_i\right) + c\right) \quad > c_{mj}$$

更新强学习器:

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} I\left(x \in R_{mj}\right)$$

最终的模型为：

$$\hat{f}(x) = f_M(x) = f_0(x) + \sum_{m=1}^{M} \sum_{j=1}^{J} c_{mj} I\left(x \in R_{mj}\right)$$

3 实验实现

事先将 train.data 文件改写成 txt 导入至 excel 文件中后读入。

“在 train.data 文件中，有 7154 条 41 维的数据，其中前 40 列为 feature，最后一列为 label.”

将数据随机打乱，9:1 的比例划分训练集和测试集：

```
def split_train_test(data, test_ratio):
  np.random.seed(46)
  shuffled_indices = np.random.permutation(len(data))  # 生成和原数据等长的无序索引
  test_set_size = int(len(data) * test_ratio)
  test_indices = shuffled_indices[:test_set_size]
  train_indices = shuffled_indices[test_set_size:]
  return data.iloc[train_indices], data.iloc[test_indices]

train,test = split_train_test(data, 0.1)
train,test = train.to_numpy(),test.to_numpy()
X_train,X_test = train[:,0:-1],test[:,0:-1]
Y_train,Y_test = train[:,-1],test[:,-1]
```

3.1 算法流程

3.1.1 决策树（回归树）

输入：训练数据集 D;

输出：回归树 $f(x)$.

做如下步骤：

1. 选择最优切分变量 j 与切分点 s，即求解

$$\arg\min_{j,s} \left[\min_{q_1} \sum_{x_1 \in R_1(j,s)} \left(y_i - c_1\right)^2 + \min_{c_2} \sum_{x_1 \in R_2(j,s)} \left(y_i - c_2\right)^2 \right]$$

2. 用选定的对 (j, s) 划分区域并决定相应的输出值：

$$R_1(j,s)=\left\{x \mid x^{(j)} \leq s\right\}, \quad R_2(j,s)=\left\{x \mid x^{(j)}>s\right\}$$

$$\hat{c}_m=\frac{1}{N_m} \sum_{x_i \in R_m(j,s)} y_i, \quad x \in R_m, \quad m=1,2$$

3. 对子区域递归调用 1 与 2，直至满足停止条件（选择使用到最后一层停止）。

4. 将输入空间划分为 M 个区域，生成决策树：

$$f(x)=\sum_{i=1}^{M} \hat{c}_i I\left(x \in R_i\right)$$

3.1.2 XGBoost

输入: 训练数据集 $T=\left\{\left(x_1, y_1\right),\left(x_2, y_2\right), \cdots,\left(x_n, y_n\right)\right\}$ $x_i \in X \subseteq R^n$ $y_i \in Y \subseteq R$；
输出: 提升树 $f_M(x)$
做如下步骤：

1. 初始化 $f_0(x)=0$

2. 对 m=1,2, $\cdots$, M

   (a) 计算残差:

   $$r_{mi}=y_i-f_{m-1}\left(x_i\right), \quad i=1,2, \cdots, N$$

   (b) 拟合残差 r_{mi} 学习一个回归树, 得到 $T_m(x)$

   (c) 更新 $f_m(x)=f_{m-1}(x)+T_m(x)$

3. 得到：

$$f_M(x)=\sum_{m=1}^{M} T_m(x)$$

3.2 实验结果

3.2.1 决策树（回归树）

取 0-9 十个不同的最大深度进行拟合训练，训练曲线：

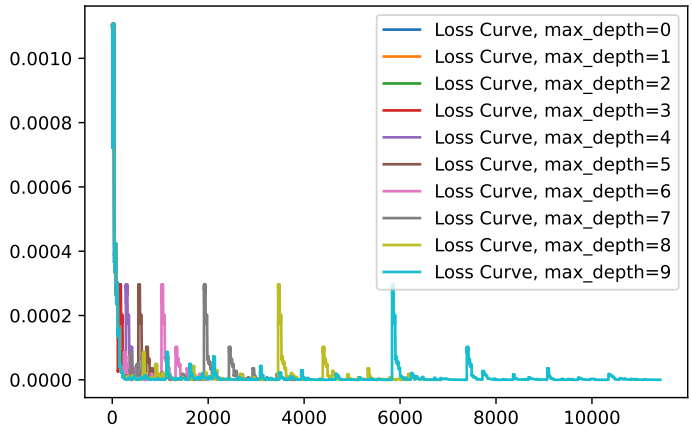

图 1: Loss Curve

对测试集预测，将 RSS（也可以写做 SSE）曲线绘制如下：

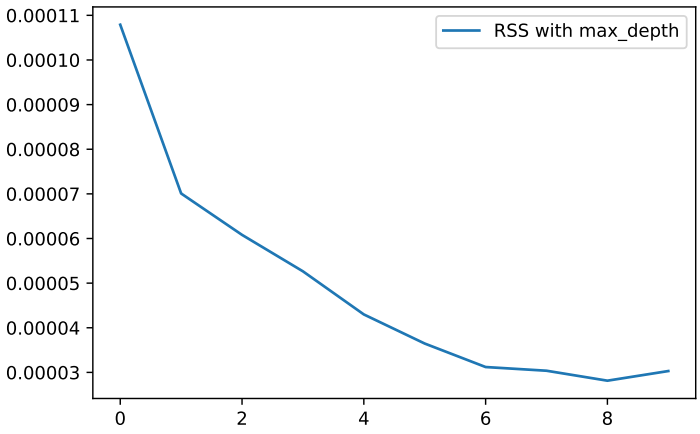

图 2: RSS

可以看出 8 的时候比较好。

3.2.2 XGBoost

展示 1-5 五个不同的最大深度进行拟合训练的训练曲线：

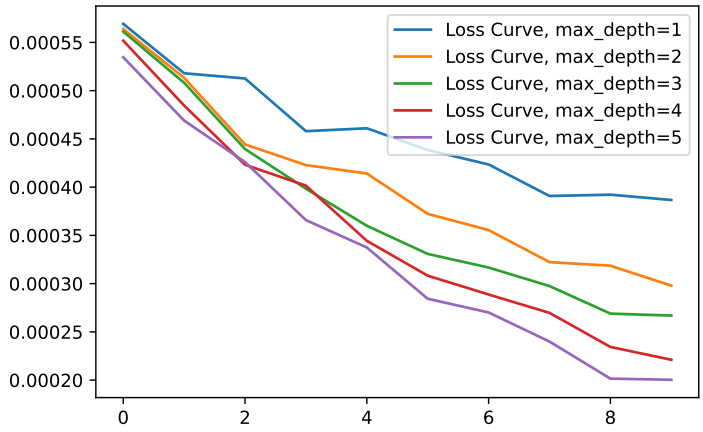

图 3: Loss Curve

这个比较明显的看出随着最大深度的增加损失下降明显。
对测试集预测，将 RSS（1-14）曲线绘制如下：

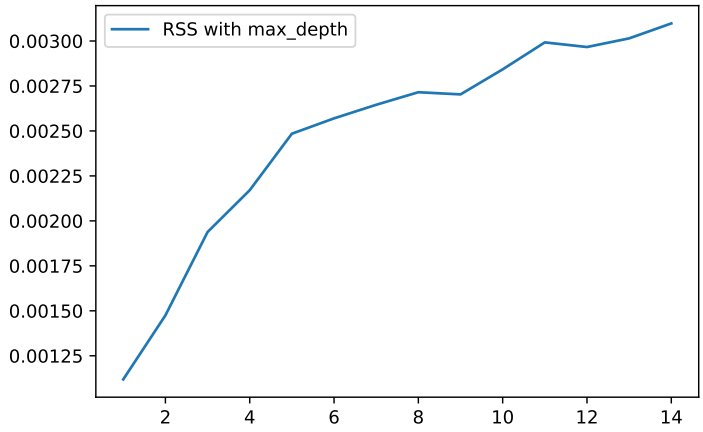

图 4: RSS

发现误差并没有因为深度的增加而显著降低。