机器学习Lab2实验报告

Xgboost实验

PB19020499 桂栋南

实验原理

Xgboost的基本原理——基模型的加法模型

Xgboost是一个面对基模型加法模型，旨在输出k个基本模型的组合模型结果，是一种顺序依次学习组合模型的方法。

我们首先假设前t-1个模型已知，那么此时我们有第k个模型的输出为

$$\hat{y_i}^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y_i}^{(t-1)} + f_t(x_i)$$

所以我们就是要用前t-1个固定模型和学习的第t个模型来优化目标函数

$$Obj^{(t)} = \sum_{i=1}^{n} loss(y_i, \hat{y_i}^{(t)}) + \sum_{k=1}^{t} penalty(f_k)$$

所以，经过推导，学习第t个模型时，要优化的目标为

$$Obj^{(t)} = \sum_{i=1}^{n} loss\left(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)\right) + penalty(f_t) + C$$

将loss项在$\hat{y_i}^{(t-1)}$泰勒展开可以得到

$$loss\left(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)\right) \approx loss\left(y_i, \hat{y_i}^{(t-_i1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)$$

其中在本次实验中，由于优化问题是回归问题，所以有

$$loss\left(y_i, \hat{y_i}^{(t-_i1)}\right) = \left(y_i - \hat{y_i}^{(t-1)}\right)^2$$

$$g_i = \frac{\partial loss\left(y_i, \hat{y_i}^{(t-1)}\right)}{\partial \hat{y_i}^{(t-1)}} = \frac{\partial \left(y_i - \hat{y_i}^{(t-1)}\right)^2}{\partial \hat{y_i}^{(t-1)}} = -2\left(y_i - \hat{y_i}^{(t-1)}\right)$$

$$h_i = \frac{\partial^2 loss\left(y_i, \hat{y_i}^{(t-1)}\right)}{\partial \left(\hat{y_i}^{(t-1)}\right)^2} = \frac{\partial g_i}{\partial \hat{y_i}^{(t-1)}} = 2$$

通过上述推导，我们已经知道了第t个模型的优化目标。

基模型的选择——决策树

在本次试验中，优化模型采用决策树模型。所以决策树采取的惩罚为

$$penalty(f) = \gamma T + \frac{1}{2}\lambda||w||_2^2$$

其中γ, λ是可调的超参数，w是学习的决策树的叶子结点的决策变量。

根据决策树的学习结果，我们有$f_t(x_i) = w_{q(x_i)}$，w_j为决策树生成的一种决策方案，所以优化目标可以简化为

$$Obj^{(t)} = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)\cdot w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)\cdot w_j^2\right] + \gamma\cdot T$$

其中T为决策树叶子总数，I_j为决策树分类后分为第j类的样本集合，所以有$1 \le j \le T$成立。

决策树的划分标准——信息增益

对上式根据w_j求导，得到w_j的最优解为

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

其中$G_j = \sum_{i\in I_j} g_i, H_j = \sum_{i\in I_j} h_i$，从而将最优解带入目标函数值，最小目标函数值为

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

上述目标函数可视为一种划分方案的好坏程度的度量，所以可以根据上述目标函数值的度量来决定决策树的划分标准，即父子结点的度量值的差可视为信息增益

$$Obj_1 = -\frac{1}{2}\cdot\frac{G^2}{H + \lambda} + \gamma T$$

$$Obj_2 = -\frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda}\right] + \gamma T$$

$$Gain = Obj_1 - Obj_2$$

上述方法给定了决策树的建立方法。

最大增益的选择——贪心算法

由于本次实验是基于回归问题的，各个属性均为连续变量，所以最大增益需要考虑所有属性的所有划分点比较最大增益，每一步采取贪心的算法——即直接选取最大的增益作为此次决策树划分的依据。

大致思想如下：

```
样本集合D,特征集合F
for feature in F:
    提取特征按照升序排列，作为新的D
    for split_point in D:
        D_L=D[:split_point],D_R=D[split_point:]
        计算gain=obj(D_L)-obj(D_R)
选取max(gain)作为划分依据
```

实验实现

实验框架

主要包括处理**数据输入输出函数**、**XgboostTree()类**模块和**Xgboost()类**模块。

代码框架如下：

```
# 先处理数据
def process_data(size=1):
    pass


# 一个决策树结点的类
class TreeNode:
    def __init__(self, feature=None, value=None,
                 threshold=None, left_child=None,
                 right_child=None):
        pass


# 定义XgboostTree类
class XgboostTree(TreeNode):
    # 继承TreeNode类
    def __init__(self, min_gain=0, min_size=10,
                 max_depth=3, loss=None, lam=0, gamma=0):
        TreeNode.__init__(self)
        pass

    # 建立决策树,调用tree_build函数
    def fit(self, data, label, pred):
        pass

    # 根据某一条数据遍历树输出结果
    def predict_value(self, treenode, data):
        pass

    # 根据输入预测结果,调用predict_value函数
    def predict(self, data):
        pass

    # 递归建树
    def tree_build(self, np_all, depth_now=1):
        pass


# 定义Xgboost()类
class Xgboost(XgboostTree):
    # 继承XgboostTree类
    def __init__(self,trees_num=5):
        XgboostTree.__init__(self)
        pass

    # 建立依据XgboostTree的加法模型
    def fit(self, data, label):
        pass

```

```
    # 根据trees_num棵XgboostTree的结果获得预测结果
    def predict(self, data):
        pass


# 主函数调用上述过程
if __name__ == "__main__":
    pass
```

停止标准

代码停止标准

采用最简洁的停止标准，可以设置trees_num，默认值为5，即建立5棵树后算法停止。

决策树停止标准

采用多重标准：

1. 可以设置树的深度max_depth，当树的深度超过max_depth时，停止划分，默认值为3；
2. 可以设置最小增益min_gain，当增益小于该值时，停止划分，默认值为0；
3. 可以设置最小样本数目min_size，当某结点样本数小于该值时，停止划分，默认值为10。

实现方法

省略数据处理部分。

树节点参数

```
class TreeNode:
    # 一个决策树结点的类
    def __init__(self, feature=None, value=None,
                 threshold=None, left_child=None,
                 right_child=None):
        # 决策树分类所用的特征
        self.feature = feature
        # 决策树分类特征的取值
        self.value = value
        # 决策树分类的阈值
        self.threshold = threshold
        # 决策树分类为true时的子节点，是一个TreeNode
        self.left_child = left_child
        # 决策树分类为false时的子节点，是一个TreeNode
        self.right_child = right_child
```

XgboostTree参数

```
def __init__(self, min_gain=0, min_size=10,
             max_depth=3, lam=0, gamma=0):
    # 继承TreeNode类
    TreeNode.__init__(self)
    # 设置划分的最小增益值
    self.min_gain = min_gain
    # 设置划分截止的最小数据量
    self.min_size = min_size
    # 设置树的最大深度
    self.max_depth = max_depth
    # 设置超参数lambda和gamma
    self.lam = lam
    self.gamma = gamma
```

XgboostTree建树

```
def tree_build(self, np_all, depth_now=1):
    '''
    采用递归方法建树，np_all是一个包含data,label,pred,G,H的大np数组
    '''
    max_gain = 0
    size_num = np_all.shape[0]
    att_num = np_all.shape[1] - 4

    # 不满足决策树停止标准，继续建树
    if size_num >= self.min_size and depth_now <= self.max_depth:
        for att in range(att_num):
            # 分别对数据的第att个属性排序，获得排序后新的np_all
            data_att = np_all[:, att]
            sort_att = np.lexsort(data_att.reshape(1,-1))
            np_all = np_all[sort_att]

            # 分别对不同属性的不同值进行划分，计算每次划分的增益
            unique_values = np.unique(np_all[:, att])
            for value in unique_values:
                # 根据不同值选取划分点
                split_i = np.searchsorted(np_all[:, att], value)

                # 根据划分点计算左右G,H
                G = sum(np_all[:, -2])
                G_left = sum(np_all[:split_i, -2])
                G_right = sum(np_all[split_i:, -2])
                H = sum(np_all[:, -1])
                H_left = sum(np_all[:split_i, -1])
                H_right = sum(np_all[split_i:, -1])

                # 如果划分有一类为空，跳过
                if not (G_left and G_right and H_left and H_right):
                    continue

                    # 获取信息增益
                    obj1 = -0.5 * (G**2/(H+self.lam)) + self.gamma
                    obj2 = -0.5 * (G_left**2/(H_left+self.lam) +
                                   G_right**2/(H_right+self.lam)) + 2 *
self.gamma
                    gain = obj1 - obj2

```

```
                    #获得最大信息增益，记录分割点和分类属性及阈值
                    if gain > max_gain:
                        max_gain = gain
                        split_point = split_i
                        feature = att
                        threshold = value
                        print("建立第{}层，选取的特征是第{}列，分类阈值是{}，信息增益是{}"
                              .format(depth_now,feature,threshold,max_gain))

                        # 根据最大收益递归划分建树
                        if max_gain > self.min_gain:
                            # 获得根据feature分类后的data
                            data_feature = np_all[:, feature]
                            sort_feature = np.lexsort(data_feature.reshape(1, -1))
                            np_all = np_all[sort_feature]
                            left_np_all = np_all[:split_point]
                            right_np_all = np_all[split_point:]

                            # 递归建树
                            left_child = self.tree_build(left_np_all,depth_now+1)
                            right_child = self.tree_build(right_np_all,depth_now+1)

                            # 返回根节点
                            return TreeNode(feature=feature,threshold=threshold,
                                            left_child=left_child,
                                            right_child=right_child)

                        # 不建树，返回叶子结点，只有value有值
                        else:
                            G = sum(np_all[:,-2])
                            H = sum(np_all[:,-1])

                            # 根据所有该分类叶子结点的样本数据取均值作为value
                            w_value = -(G/(H+self.lam))
                            return TreeNode(value=w_value)
```

XgboostTree 预测

```
def predict_value(self, treenode, data):
    # 这里的data是一行数据

    # treenode.value非空，是叶子结点，直接返回
    if treenode.value:
        return treenode.value

    # 选取划分属性
    data_feature = data[treenode.feature]
    # 根据阈值大小，在左右子树递归寻找
    if data_feature < treenode.threshold:
        return self.predict_value(treenode.left_child,data)
    else:
        return self.predict_value(treenode.right_child,data)
```

Xgboost fit

```
def fit(self,data, label):
    # 初始化预测全0
    pred = np.zeros(label.shape)
    # 建trees_num棵树
    for i in range(self.trees_num):
        print("-----正在建立第{}棵树-----".format(i+1))
        xgbt = XgboostTree()
        xgbt.fit(data, label, pred)
        print("-----第{}棵树建立完毕-----".format(i+1))

        # 下一次建树的输出为之前全部的预测结果之和
        pred = pred + xgbt.predict(data)

        #记录该树
        self.trees.append(xgbt)
```

Xgboost 预测

```
def predict(self,data):
    # 初始化pred
    pred = np.zeros(data.shape[0])

    # 根据fit记录的树，遍历每棵树输出结果
    for tree in self.trees:
        #最终的预测为trees_num棵树预测结果之和
        pred = pred + tree.predict(data)

    return pred
```

实验结果

程序输出

程序运行时会输出每棵树的划分标准等，如下所示

```
-----正在建立第1棵树-----
建立第1层，选取的特征是第6列，分类阈值是-13.0，信息增益是0.0003168107655992327
建立第2层，选取的特征是第10列，分类阈值是0.033，信息增益是8.308170522565743e-05
建立第3层，选取的特征是第2列，分类阈值是-0.08，信息增益是2.5490632563603252e-05
建立第3层，选取的特征是第19列，分类阈值是0.044，信息增益是1.2688894681280407e-05
建立第2层，选取的特征是第6列，分类阈值是-9.0，信息增益是7.909937709606575e-05
建立第3层，选取的特征是第17列，分类阈值是0.028，信息增益是3.7196010542996354e-05
建立第3层，选取的特征是第2列，分类阈值是-0.03，信息增益是8.106307164309741e-06
-----第1棵树建立完毕-----
...
...
-----正在建立第5棵树-----
建立第1层，选取的特征是第2列，分类阈值是-0.48，信息增益是3.2291204388770738e-06
建立第2层，选取的特征是第8列，分类阈值是0.005，信息增益是8.173423565954147e-07
建立第3层，选取的特征是第3列，分类阈值是-0.24，信息增益是6.454393322149959e-07
建立第3层，选取的特征是第5列，分类阈值是-0.2，信息增益是2.94263311519359e-07
建立第2层，选取的特征是第5列，分类阈值是-1.6，信息增益是2.0637070771078506e-06
建立第3层，选取的特征是第8列，分类阈值是0.05，信息增益是6.062306330697147e-07
建立第3层，选取的特征是第10列，分类阈值是0.064，信息增益是1.1764668290848997e-06
```

```
20 -----第5棵树建立完毕-----
```

指标输出

通过sklearn.model_selection.train_test_split将数据自由划分为占70%的训练集和占30%的测试集。

测试时选取可变参数trees_num为5。

根据最后预测得到的结果包括RMSE和R^2，如下所示

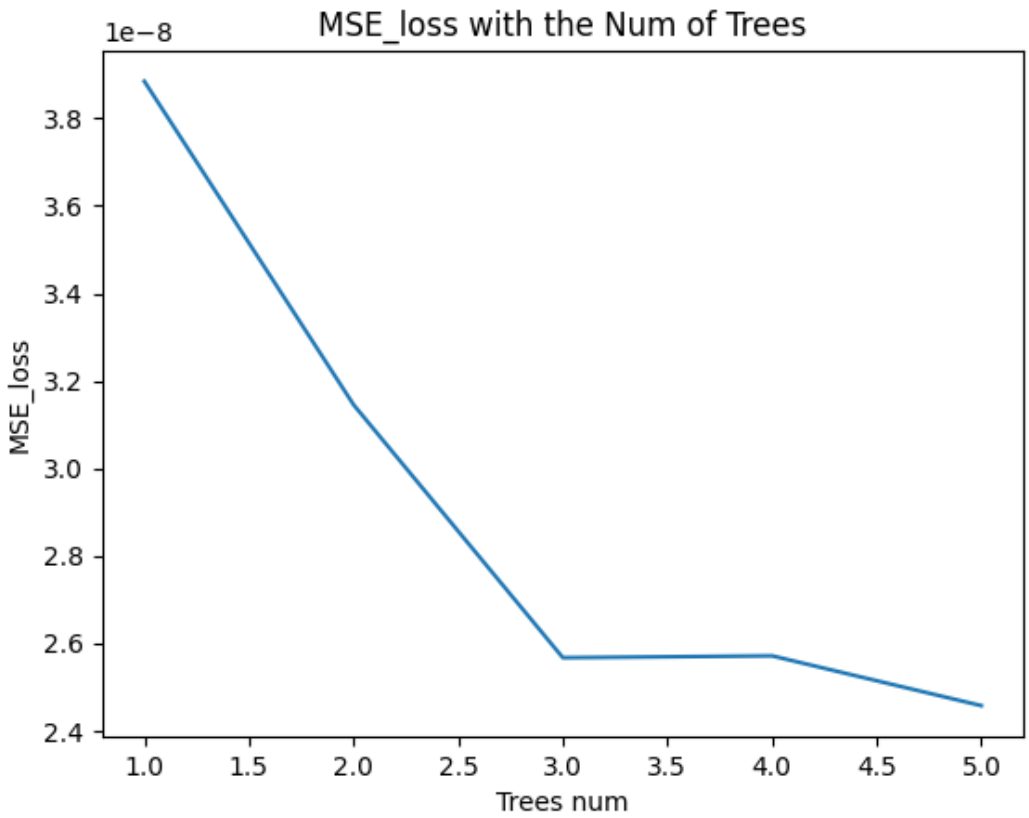
```
1 RMSE的值为：0.00021611655673910686
2 R的值为：0.7403731109941707
```

数据可视化

Loss变化

随trees_num的变化

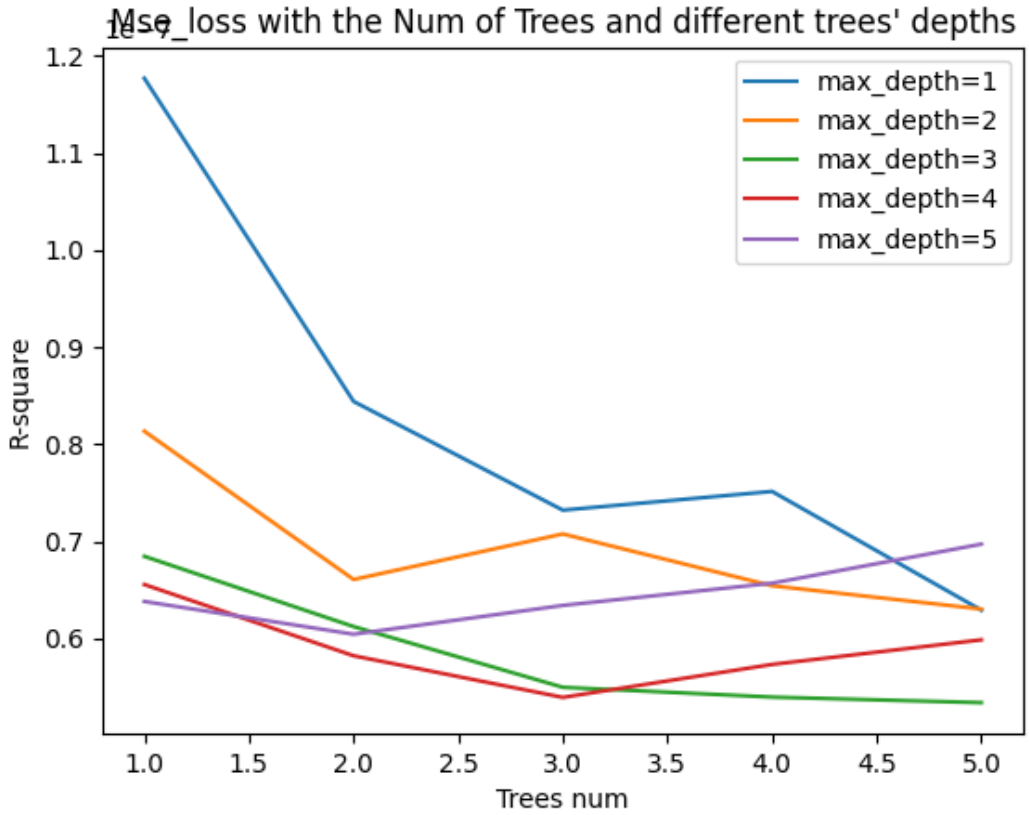
根据数据训练的结果，根据不同xgbooosttree的数量，其MSE_loss的变化如下所示（取size=0.2）



可以看到，随着树的数目的增多，loss在减少。这符合xgboost的原理，xgboost本就是用一个一个基模型来拟合预测结果与标签之间的误差。

随max_depth的变化

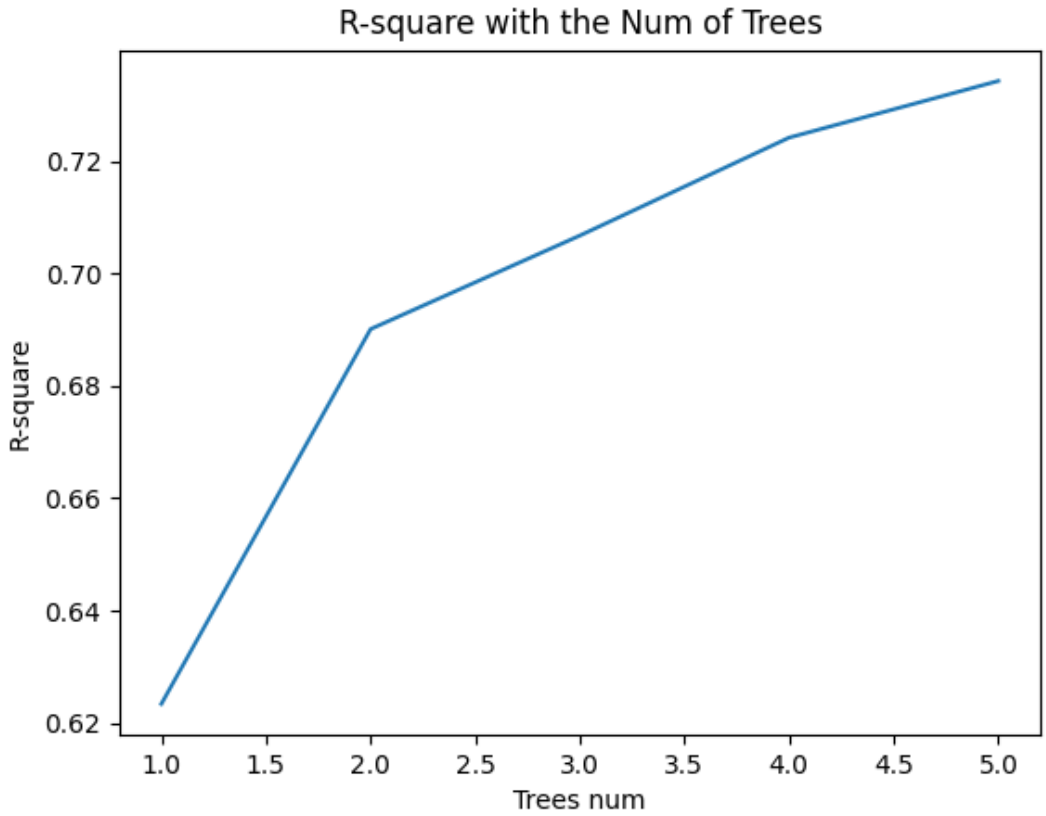
根据数据训练的结果，根据xgbooosttree最大深度的不同，其MSE_loss的变化如下所示（取size=0.2）



可以看出，xgboost模型对每棵树的深度不是很敏感，所以设置建树停止标准为简单的最大树深度是合理的。

R^2变化

根据数据训练的结果，根据不同xgbooosttree的数量，其R^2的变化如下所示（取size=1）



可以看出，随着树的数量的增长，R^2值在增加，这表明随着基模型的增加，xgboost拟合的效果在变好。且随着树数量的增加，R^2的值仍有增加的趋势。