# 机器学习概论第三次大作业

PB21000302

张成

# 1.实验原理

## 1.1XGBoost

XGBoost 是由多个基模型组成的一个加法模型，假设第 k 个基模型是 fk(x), 那么前 个模型组成的模型的输出为$f_k(x)$那么模型输出为

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

其中 xi 为第表示第 i 个训练样本，$y_i$ 表示第 i 个样本的真实标签；$\hat{y}_i(t)$ 表示前 t 个模型对第i个样本的标签最终预测值之和。

我们在学习第 t 个基模型时，优化目标为

$$Obj^{(t)} = \Sigma_{i=1}^{n} loss(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + penalty(f_t) + C$$

对于$loss(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))$使用 Taylor 展开，则有

$$loss(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) \approx loss(yi, \hat{y}_i^{(t-1)}) + g_i * f_t(x_i) + \frac{1}{2} h_i * f_t^2(x_i)$$

其中$g_i = \frac{\partial loss(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$一阶导数$h_i = \frac{\partial^2 loss(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))}{\partial (\hat{y}^{(t-1)})^2}$为二阶导数，对于回归问题$loss(y_i, \hat{y}_i^{(t-1)}) = (y_i - \hat{y}_i^{(t-1)})^2$则$g_i = -2(y_i - \hat{y}_i^{(t-1)})$,$h_i = 2$

## 1.2 决策树

对输入 xi，决策树中有 $f(x_i) = w_q(x_i)$，其中$q(x_i)$ 表示将输入 xi 映射到叶子节点的索引（如$q(x_i) = 3$ 表示 xi 对应的叶子节点为$w_3$）。

设置惩罚函数为$penalty(f) = \gamma \cdot T + 12\lambda \cdot ||w||2$其中T为节点数，λ，γ为超参数

记对应到第 j 个叶子节点的样本为 $I_j$，即$I_j = \{i|q(x_i) = j\}(1 \le j \le T)$。此时优化目标为

$$Obj^{(t)} = \Sigma_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + penalty(f_t)$$

$$= \Sigma_{j=1}^{T}[\Sigma_{i \in I_j}(g_i) w_j + \frac{1}{2}(\Sigma_{i \in I_j} h_i + \lambda) f_t^2(x_i)] + \gamma T$$

简记 $G_j = \sum_{i \in I_j} g_i$，$Hj = \sum_{i \in I_j} h_i$，则

$$Obj^{(t)} = \Sigma_{j=1}^{T}[G_j w_t + \frac{1}{2}(H_j + \lambda) w_j^2]$$

求解$Obj$的最小值$Obj = -\frac{1}{2}\Sigma_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$在$w = -\frac{G_j}{H_j+\lambda}$时取到

所以对于某个划分划分前后$Obj$的差值为

$$\Delta Obj = -\frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}] + \gamma$$

让这个差值达到最大值就可

# 2.实现步骤

## 2.1读取数据

这次试验需要记录划分，使用pandas会比numpy更方便

```
df = pd.read_csv('train.data',names=[i for i in range(41)])
m,n=df.shape#m=7k,n=14
X=df.iloc[:,:40]
y=df.iloc[:,[40]]
```

注意需要为dataframe命名，方便后续对columns操作

2.2计算函数

根据公式可以得出G,H等值的计算函数，下为G的计算函数

```
def gradient(y1,y2):
    y11 = np.transpose(np.array(y1))
    y22 = np.transpose(np.array(y2))
    return -2*sum(y11-y22)
```

## 2.3决策树的建立

决策树的数据结构是二叉树

```
class DevisionTree(object):
    def __init__(self):
        self.lchild=None
        self.rchild=None
        self.div_column=None
        self.div_value=None
        self.is_leaf=False
        self.leaf_value=None
```

其中l/rchild是它的左右子树，is_leaf是判定它是不是叶子节点，leaf_value是叶子结点的返回值，div_column是决定分割的特征，div_value是决定分割的值。

### 2.3.1停止策略

在建立过程中，我使用限制决策树深度和限制分裂时至少剩余的特征数量，防止决策树太深，计算代价过高，以及分裂时特征太少，这个分支被用到的概率太低。

```
def regress_one_tree(min_width,max_depth,train_data,train_res,results,lambda_,gamma_,depth,tree_node):
    if len(results)<min_width or depth>max_depth:
        tree_node.is_leaf=True
        tree_node.leaf_value= sum(np.array(train_res-results))/len(train_res)
        return
```

其中函数头中min_width代表最少的特征数，max_depth代表最大的深度，train_data代表特征值，train_res代表$y$，results代表$\hat{y}^{(t-1)}$ ，lambda_ 和gamma_代表λ和γ，depth代表现在的深度，tree_node，代表当前节点。

### 2.3.2分割策略

寻找分割效果最好的方法是对每个特征，遍历每个值作为分割，看谁的$\Delta Obj$最大

```
for col in train_data.columns:
        val=np.transpose(np.array(train_data.loc[:,col].drop_duplicates()))
        max_val=max(val)
        for v in val:
            if v==max_val:
                continue
            y_left=train_res.loc[train_data[train_data[col] <= v].index, :]
            y_right=train_res.loc[train_data[train_data[col] > v].index, :]
            res_left=results.loc[train_data[train_data[col] <= v].index, :]
            res_right=results.loc[train_data[train_data[col] > v].index, :]
            #分割y和y^hat
            o_now=_object(train_res,results,lambda_,gamma_)
            o_left=_object(y_left,res_left,lambda_,gamma_)
            o_right=_object(y_right,res_right,lambda_,gamma_)
            if o_now-o_right-o_left>max_obj:
                max_obj=o_now-o_right-o_left
                best_column=col
                best_value=v
```

找到最佳分割后，按照分割进行左右子树的构建

```
regress_one_tree(min_width,max_depth,x_left,y_left,res_left,lambda_,gamma_,depth+1,tree_node.lchild)
regress_one_tree(min_width,max_depth,x_right,y_right,res_right,lambda_,gamma_,depth+1,tree_node.rchild)
```

随后根据得到的决策树计算$f_t(x_i)$

```
def predict_one_example(tree,x):
    if tree.is_leaf:
        return tree.leaf_value
    col=tree.div_column
    value=tree.div_value
    if x[col]>value:
        return predict_one_example(tree.rchild,x)
    else :
        return predict_one_example(tree.lchild,x)
def predict(tree,xx):
```

```
    res=pd.DataFrame([0]*len(X.index),index=XX.index,columns=[40])
    for i in XX.index:
        x=XX.loc[i,:]
        res.loc[i,40]=predict_one_example(tree=tree,x=x)
    return res
```

随后就可以建立XGBT模型

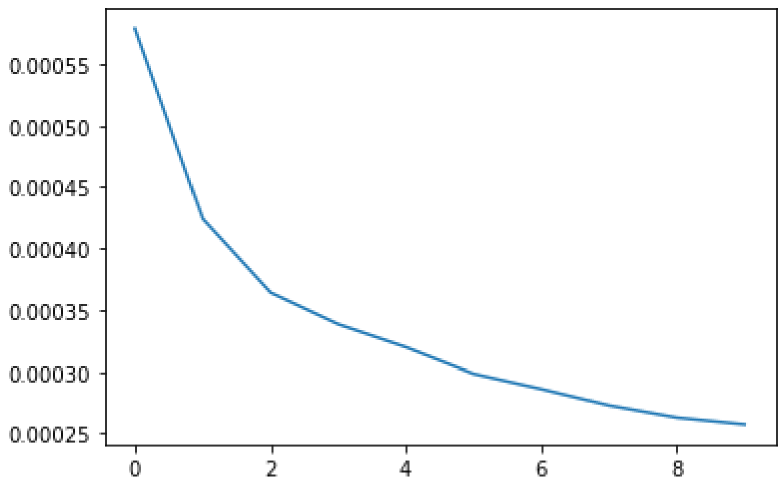
```
class xg():
    def __init__(self,X,y,lambda_=0.1,gamma_=0.000001,depth=4, least_width = 70,iter_=4):
        self.J=[]
        self.roots=[]
        self.gamma_=gamma_
        self.lambda_=lambda_
        self.trees=[]
        self.X=X
        self.y=y
        self.lr=1
        self.iter=iter_
        self.miniwidth=least_width
        self.max_depth=depth
        self.res=pd.DataFrame([0] * len(X.index), index = X.index,columns=[40])
```

训练过程中，一直将上一次训练的结果加到res中，这样y和res就成为下一次训练的目标

```
def update_RES(self,tree):
        pre=predict(tree,self.X)
        self.res+=self.lr*pre
        self.trees.append(tree)
        self.J.append(loss(self.res,self.y))

    def train_a_model(self):
        self.trees=[]
        for i in range(int (self.iter)):
            new_tree=DevisionTree()
            regress_one_tree#省略
            self.update_RES(new_tree)
```

训练过程中$loss$变化趋势如下(iter=10,max_depth=1,miniwidth=40,lr=1)



# 3.实验结果

## 3.1XGBT停止策略

策略是设定固定的循环次数停止，拟合因为残差的提升越来越小，所以固定循环次数作为一个超参数。

```
        self.iter=max_iter
```

## 3.2最佳模型展示和验证

本实验的两个检验参数

1.$RMSE$:

$$RMSE=\sqrt{\frac{1}{m}\Sigma_{i=1}^{m}(y_{test}^{(i)}-\hat{y}_{test}^{(i)})^2}$$

2.$R^2$ :

$$R^2=1-\frac{\Sigma_{i=1}^{m}(y_{test}^{(i)}-\hat{y}_{test}^{(i)})^2}{\Sigma_{i=1}^{m}(\bar{y}_{test}-y_{test}^{(i)})^2}$$

```
def split_data(row_data):
    row_data=row_data.sample(frac=1)#打乱
    m,n=row_data.shape
    a=int(0.1*m)
    Z=[]
    for i in range(10):
        Z.append(row_data.iloc[a*i:a*(i+1),:])
    return Z

def check_algorithm(df0):
    R=[]
    RMES=[]
    for i in range(1):
        S=split_data(df0)
        for j in range(1):
            X1=pd.DataFrame(columns=df0.columns)
            for k in range(10):
                if k!=j:
                    X1=pd.concat([X1,S[k]],axis=0)
            X2=S[j]
            y1=X1.loc[:,[40]]
            X1=X1.drop(labels=[40],axis=1)
            y2=X2.loc[:,[40]]
            X2=X2.drop(labels=[40],axis=1)

            moxin=xg(X1,y1)
            moxin.train_a_model()
            a=np.linspace(0,len(moxin.J)-1,endpoint=True,num=len(moxin.J))
            plt.plot(a,moxin.J)
            rres=pd.DataFrame([0] * len(X2.index), index = X2.index,columns=
[40])
            print(rres)
```

```
            for tree in moxin.trees:
                rres=rres+predict(tree,X2)
            r1=math.sqrt(loss(y2,rres)/len(y2))
            ave=np.mean(np.array(y2))
            AVE=pd.DataFrame([ave] * len(X2.index), index = X2.index,columns=[40])
            r2=1-loss(y2,rres)/loss(y2,AVE)
            print(rres)
            print(y2)
            print(r1,r2)
            R.append(r2)
            RMES.append(r1)
    return R,RMES
            #X1训练集，X2测试集
R,RMSE=check_algorithm(df)
```

以上为我实现五次十折验证的代码，主要思路就是将数据集打乱划分成十份，将九份作为训练集，一份作为测试集，返回历史R和RMSE的值，最终得到两个评价标准

## 3.3不同参数对比

本实验最终得出最好的参数是$lambda = 0.1, gamma = 0.000001, depth = 5, iter = 5$得到的评价指标为

$$RMES = 0.00018791202881863537$$
$$R^2 = 0.8018208404593465$$

下为改变一些参数得到的结果

| 参数变化 | RMSE\|\|R^2 |
|---|---|
| gamma= 3e-08 | 0.00018624432845252 86\|\| 0.7758445321161277 |
| gamma= 1e-07 | 0.00021201053114827467\|\|0.7593584059791325 |
| gamma= 3e-06 | 0.0002137353631027598\|\| 0.7538597847950195 |
| gamma= 3e-05 | 0.00021400747901045473\|\| 0.7502384513575996 |

gamma值的改变带来的评价标准变化不是很规则，有时使它变大，有时变小。

| 参数变化 | RMSE\|\|R^2 |
|---|---|
| lambda= 0.001 | 0.0001978130742269341\|\| 0.773439452682136 |
| lambda= 0.003 | 0.0001964262481324829\|\| 0.7673521915168998 |
| lambda= 0.01 | 0.00021039869116737503\|\| 0.7577091314112543 |
| lambda= 0.3 | 0.0002009096357098994 5 \|\|0.7400504543086542 |

随着lambda值变大，评价标准先变好再变差

| 参数变化 | RMSE\|\|R^2 |
| --- | --- |
| depth= 2 | 0.00021852545754406686\|\| 0.7272074965614317 |
| depth= 8 | 0.0002173352328801674\|\| 0.696967706235973 |
| depth= 12 | 0.0002241941078890356\|\| 0.6911105105539258 |
| depth= 16 | 0.00025556373944664453\|\| 0.6143385647275147 |

随着最大深度的变大，效果也是先编号再变差，说明树的深度太大会过拟合

| 参数变化 | RMSE\|\|R^2 |
| --- | --- |
| iter= 2 | 0.00021547289740691056 \|\|0.7309767773754523 |
| iter= 8 | 0.00019533965065446422 \|\|0.7830636883694284 |
| iter= 12 | 0.00020496130216064644 \|\|0.7232513654191343 |
| iter= 16 | 0.00019988307788696398 \|\|0.7646338551586132 |

同上，循环次数变大，效果也是先编号后边查，说明太多次循环会过拟合