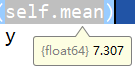
# GradientBoostingRegressor

**import** pandas **as** pd  
**import** numpy **as** np  
df = pd.DataFrame([[1,5.56],[2,5.7],[3,5.91],[4,6.4],[5,6.8],[6,7.05],[7,8.9],[8,8.7],[9,9],[10,9.05]])  
X = df.iloc[:,[0]]  
Y = df.iloc[:,-1]  
**from** sklearn.ensemble **import** GradientBoostingRegressor  
model = GradientBoostingRegressor(max\_depth=1,n\_estimators=20)  
model.fit(X,Y)  
print(model.predict(X))

## fit

*fit initial model - FIXME make sample\_weight optional*

### self.init\_.fit(X, y, sample\_weight)



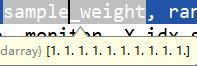
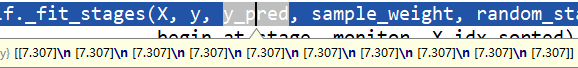
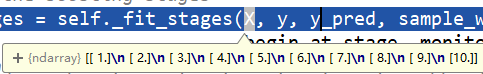
*init predictions*

### y\_pred = self.init\_.predict(X)

y.fill(self.mean)

*fit the boosting stages*

### n\_stages = self.\_fit\_stages(X, y, y\_pred, sample\_weight, random\_state,begin\_at\_stage, monitor, X\_idx\_sorted)



*# perform boosting iterations*i = begin\_at\_stage  
**for** i **in** range(begin\_at\_stage, self.n\_estimators):  
 *# fit next stage of trees* y\_pred = self.\_fit\_stage(i, X, y, y\_pred, sample\_weight,sample\_mask, random\_state, X\_idx\_sorted,X\_csc, X\_csr) **return** i + 1

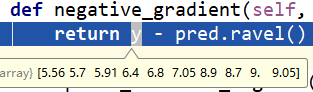
#### \_fit\_stage(self, i, X, y, y\_pred, sample\_weight, sample\_mask,random\_state, X\_idx\_sorted, X\_csc=None, X\_csr=None)

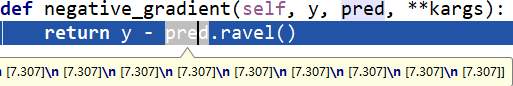
*K : int  
 The number of regression trees to be induced;  
 1 for regression and binary classification;  
 ``n\_classes`` for multi-class classification.  
"""*

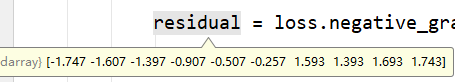
**for** k **in** range(loss.K):  
 residual = loss.negative\_gradient(y, y\_pred, k=k,  
 sample\_weight=sample\_weight)  
 *# induce regression tree on residuals* tree = DecisionTreeRegressor(  
 criterion=self.criterion,  
 splitter=**'best'**,  
 max\_depth=self.max\_depth,  
 min\_samples\_split=self.min\_samples\_split,  
 min\_samples\_leaf=self.min\_samples\_leaf,  
 min\_weight\_fraction\_leaf=self.min\_weight\_fraction\_leaf,  
 min\_impurity\_decrease=self.min\_impurity\_decrease,  
 min\_impurity\_split=self.min\_impurity\_split,  
 max\_features=self.max\_features,  
 max\_leaf\_nodes=self.max\_leaf\_nodes,  
 random\_state=random\_state,  
 presort=self.presort)  
  
 **if** X\_csc **is not None**:  
 tree.fit(X\_csc, residual, sample\_weight=sample\_weight,  
 check\_input=**False**, X\_idx\_sorted=X\_idx\_sorted)  
 **else**:  
 tree.fit(X, residual, sample\_weight=sample\_weight,  
 check\_input=**False**, X\_idx\_sorted=X\_idx\_sorted)  
  
 *# update tree leaves* **if** X\_csr **is not None**:  
 loss.update\_terminal\_regions(tree.tree\_, X\_csr, y, residual, y\_pred,  
 sample\_weight, sample\_mask,  
 self.learning\_rate, k=k)  
 **else**:  
 loss.update\_terminal\_regions(tree.tree\_, X, y, residual, y\_pred,  
 sample\_weight, sample\_mask,  
 self.learning\_rate, k=k)  
  
 *# add tree to ensemble* self.estimators\_[i, k] = tree  
  
**return** y\_pred

##### negative\_gradient(self, y, pred, \*\*kargs):

**return** y - pred.ravel()







##### loss.update\_terminal\_regions(tree.tree\_, X, y, residual, y\_pred, sample\_weight, sample\_mask, self.learning\_rate, k=k)

**def** update\_terminal\_regions(self, tree, X, y, residual, y\_pred,  
 sample\_weight, sample\_mask,  
 learning\_rate=1.0, k=0):  
 *"""Least squares does not need to update terminal regions.  
  
 But it has to update the predictions.  
 """  
 # update predictions* y\_pred[:, k] += learning\_rate \* tree.predict(X).ravel()









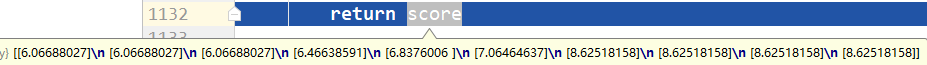
## predict

model.predict(X)

### self.\_decision\_function(X).ravel()

#### predict\_stages(self.estimators\_, X, self.learning\_rate, score)

**def** predict\_stages(\*args, \*\*kwargs): *# real signature unknown  
 """  
 Add predictions of ``estimators`` to ``out``.  
   
 Each estimator is scaled by ``scale`` before its prediction  
 is added to ``out``.  
 """* **pass**



使用gbdt做回归，loss=ls、learning\_rate=0.1、**n\_estimators** =20时

fit：

第一个y\_pred是均值

循环**estimators** 次计算负梯度，使用回归树拟合，加入加法模型

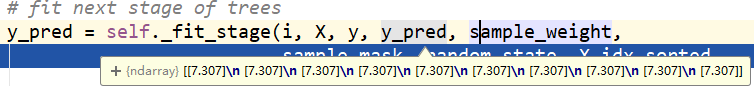
最终得到加法模型

## 数据可视化

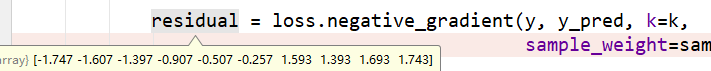
[[1,5.56],[2,5.7],[3,5.91],[4,6.4],[5,6.8],[6,7.05],[7,8.9],[8,8.7],[9,9],[10,9.05]]

### round1

第1轮加法模型会预测为均值



计算第1轮加法模型下损失函数的负梯度（残差）

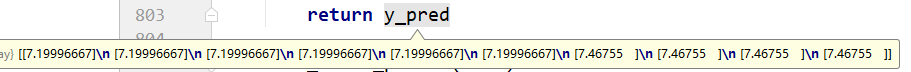
5.56-7.307=-1.747

使用第1轮加法模型下损失函数的负梯度拟合一棵回归树

对拟合好的树执行update\_terminal\_regions()，不过Least squares does not need to update terminal regions

But it has to update the predictions.

y\_pred[:, k] += learning\_rate \* tree.predict(X).ravel()

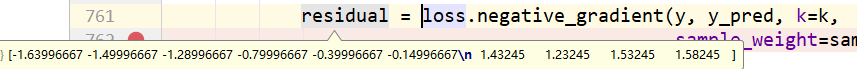


将回归树加入加法模型

self.estimators\_[i, k] = tree

### round2

计算第2轮加法模型下损失函数的负梯度（残差）



5.56-7.199=-1.639

使用第2轮加法模型下的负梯度拟合一棵回归树，更新预测值



### round3

计算第3轮加法模型下损失函数的负梯度（残差）



5.56-7.103=-1.543

使用第3轮加法模型下的负梯度拟合一棵回归树，更新预测值



### 。。。

### round20





### 预测

**def** predict\_stages(\*args, \*\*kwargs): *# real signature unknown  
 """  
 Add predictions of ``estimators`` to ``out``.  
   
 Each estimator is scaled by ``scale`` before its prediction  
 is added to ``out``.  
 """* **pass**

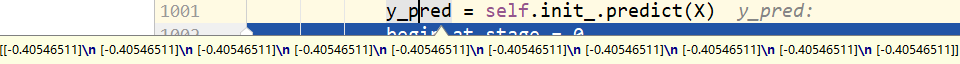
# GradientBoostingClassifier

**import** pandas **as** pd  
**import** numpy **as** np  
**import** math  
**from** sklearn.ensemble **import** GradientBoostingClassifier  
df = pd.DataFrame([[1,-1],[2,-1],[3,-1],[4,1],[5,1],  
 [6,-1],[7,-1],[8,-1],[9,1],[10,1]])  
X = df.iloc[:,[0]]  
Y = df.iloc[:,-1]  
model = GradientBoostingClassifier(n\_estimators=20, learning\_rate=1.0,  
 max\_depth=1, random\_state=0)  
model.fit(X, Y)  
print(model.predict(X))

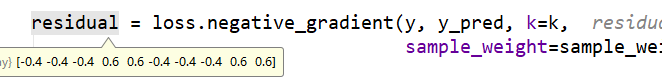
## 模型初始化

第1轮加法模型不在预测为均值

**def** fit(self, X, y, sample\_weight=**None**):  
 *# pre-cond: pos, neg are encoded as 1, 0* **if** sample\_weight **is None**:  
 pos = np.sum(y)  
 neg = y.shape[0] - pos  
 **else**:  
 pos = np.sum(sample\_weight \* y)  
 neg = np.sum(sample\_weight \* (1 - y))  
  
 **if** neg == 0 **or** pos == 0:  
 **raise** ValueError(**'y contains non binary labels.'**)  
 self.prior = self.scale \* np.log(pos / neg)



计算第1轮加法模型下损失函数的负梯度（残差）



计算负梯度的公式 expit(x) = 1/(1+exp(-x))

**def** negative\_gradient(self, y, pred, \*\*kargs):  
 *"""Compute the residual (= negative gradient). """* **return** y - expit(pred.ravel())

## 调整叶子结点

使用第1轮加法模型的负梯度拟合一棵回归树，但这里要调整叶子结点，每个叶子结点的输出值依赖于选用的损失函数

**def** update\_terminal\_regions(self, tree, X, y, residual, y\_pred,  
 sample\_weight, sample\_mask,  
 learning\_rate=1.0, k=0):  
 *"""Update the terminal regions (=leaves) of the given tree and  
 updates the current predictions of the model. Traverses tree  
 and invokes template method `\_update\_terminal\_region`.  
  
 Parameters  
 ----------  
 tree : tree.Tree  
 The tree object.  
 X : ndarray, shape=(n, m)  
 The data array.  
 y : ndarray, shape=(n,)  
 The target labels.  
 residual : ndarray, shape=(n,)  
 The residuals (usually the negative gradient).  
 y\_pred : ndarray, shape=(n,)  
 The predictions.  
 sample\_weight : ndarray, shape=(n,)  
 The weight of each sample.  
 sample\_mask : ndarray, shape=(n,)  
 The sample mask to be used.  
 learning\_rate : float, default=0.1  
 learning rate shrinks the contribution of each tree by  
 ``learning\_rate``.  
 k : int, default 0  
 The index of the estimator being updated.  
  
 """  
 # compute leaf for each sample in ``X``.* terminal\_regions = tree.apply(X)  
  
 *# mask all which are not in sample mask.* masked\_terminal\_regions = terminal\_regions.copy()  
 masked\_terminal\_regions[~sample\_mask] = -1  
  
 *# update each leaf (= perform line search)* **for** leaf **in** np.where(tree.children\_left == TREE\_LEAF)[0]:  
 self.\_update\_terminal\_region(tree, masked\_terminal\_regions,  
 leaf, X, y, residual,  
 y\_pred[:, k], sample\_weight)  
  
 *# update predictions (both in-bag and out-of-bag)* y\_pred[:, k] += (learning\_rate  
 \* tree.value[:, 0, 0].take(terminal\_regions, axis=0))

numerator = np.sum(sample\_weight \* residual)  
denominator = np.sum(sample\_weight \* (y - residual) \* (1 - y + residual))

## 可视化

[[1,-1],[2,-1],[3,-1],[4,1],[5,1],[6,-1],[7,-1],[8,-1],[9,1],[10,1]]

### round1

第1轮加法模型不是预测为均值了

self.prior = self.scale \* np.log(pos / neg)

pos 是1类数量 neg是-1类数量

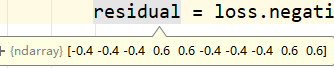
np.log(4/6)=-0.4054



计算负梯度，程序会自动将-1类变为0

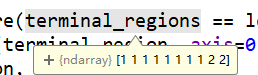
a = np.log(4/6)  
print(0-1/(1+np.exp(-a))) -0.4

**return** y - expit(pred.ravel())



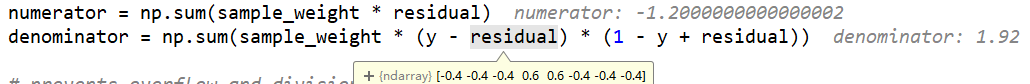
### round2

使用第1轮加法模型下的负梯度拟合一棵回归树，这里就需要对叶子结点进行调整了

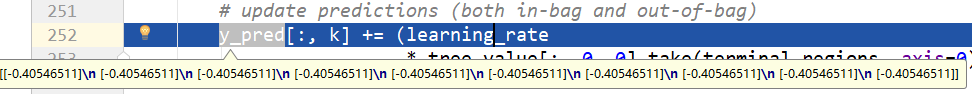


前8个样本被分在了左叶结点，后2个样本被分在了右叶结点

以左叶结点为例，该结点最终的输出值的计算如下：

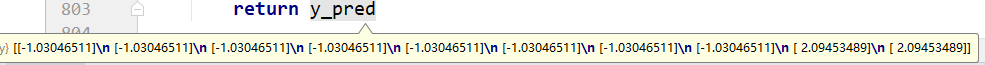


分子/分母= -0.625，这里的学习率=1

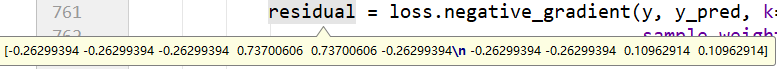


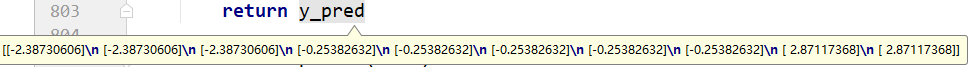
加入加法模型后

-0.4054-0.625=-1.0304



### round3





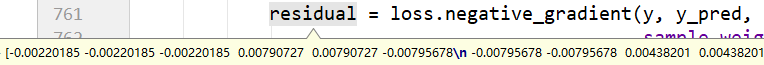
### round4





### ...

### round20





### 预测

score = self.decision\_function(X) 封装  
decisions = self.loss\_.\_score\_to\_decision(score) 输出类别  
**return** self.classes\_.take(decisions, axis=0)