GBDT算法

Gradient Boosting Decison Tree(梯度提升决策树)

一、课前准备

- 了解Boosting算法的思想
- 了解决策树算法：ID3、C4.5、CART
- 了解梯度下降

二、课堂主题

- 本节课主要讲授GBDT算法、提升的概念，以及如何计算GBDT的残差。
- 讲授提升树算法内容。
- 讲授GBDT的代码应用。

三、课堂目标

- 能够掌握传统的集成框架的类型
- 能够掌握GBDT的算法过程
- 能够掌握GBDT的残差

四、知识要点

4.1 CART预测

编号	年龄（岁）	体重（KG）	是否爱运动	身高（M）
A	5	20	否	1.1
B	7	30	是	1.3
C	22	70	否	1.7
D	30	60	是	1.9
E	25	65	是	?

使用回归决策树预测身高

CART回归树

输入：训练集D

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

输出：回归树

- Step1：针对数据集D（此时树的深度为0），遍历每一个特征Feature的每一个值Value，对于每一个Value将原数据集S分裂成2个集合：左集合S_left(<=Value的样本)、右集合S_right(>Value的样本)，每一个集合也叫做一个结点。分别计算这2个集合的均方误差（mse），找到使得（left_mse+right_mse）最小的那个Value，记录下此时的Feature名称和Value，这个就是最佳分割特征以及最佳分割值。

$$S_{left} = \{x | x < Value\}$$

$$S_{right} = \{x | x > Value\}$$

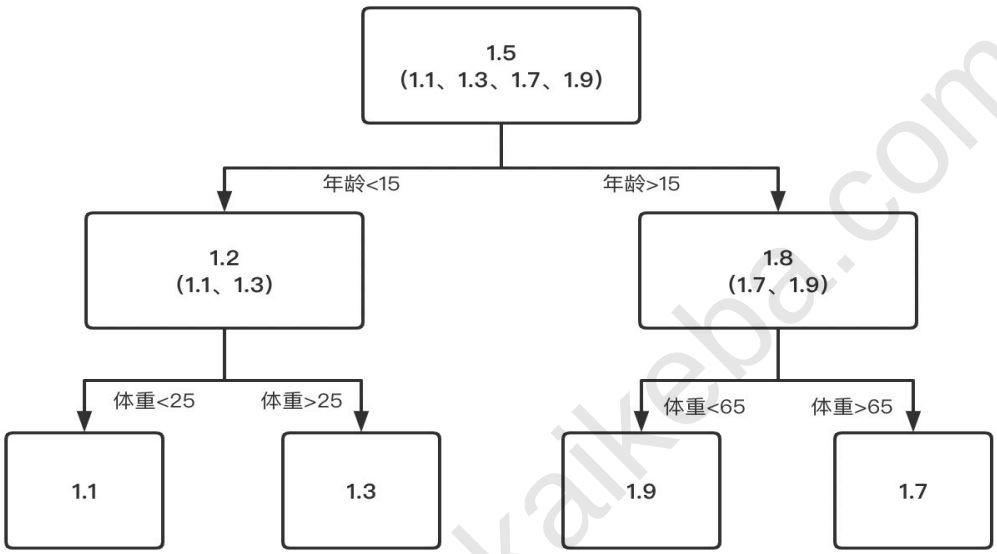
$$\min_{value}[\sum_{x_i \in s_{left}} (y - c_{left})^2 + \sum_{x_i \in s_{right}} (y - c_{right})^2]$$

（使用MSE的好处在于其一阶导数和二阶导数可求并好求。）

- Step2：找到最佳分割Feature以及最佳分割Value之后，用该Value将集合S分裂成2个集合：左集合S_left、右集合S_right，每一个集合也叫做一个结点。此时树的深度depth += 1。
- Step3：针对集合S_left、S_right分别重复步骤1,2，直到达到终止条件。
- Step4：最后生成的并且不再进行分裂的集合就叫做叶子结点。落在该叶子节点内的样本的预测值，就是该叶子结点的值。同一个叶子结点中的样本具有同一个预测值。
- 最终将空间划分成S1、S2...Sm生成的回归树模型：

$$f(x) = \sum_{m=1}^{M} c_m I(x \in S_m)$$

$$C_m = ave(y_i | x_i \in S_m)$$



4.2 提升树

使用提升树预测身高

Boosting Tree的流程

- （1）初始化$f_0(x)=0$
- （2）对m=1,2,...,M

（a）计算残差:

$r_{mi}=y_i-f_{m-1}(x_i),i=1,2,\ldots,N$

（b）拟合残差r_{mi}学习一个回归树，得到$h_m(x)$

（c）更新$f_m(x)=f_{m-1}+h_m(x)$

- （3）得到回归问题提升树

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

对于上面的流程中**残差**又代表着什么呢?

假设我们前一轮迭代得到的强学习器是:

$f_{t-1}(x)$

损失函数是:

$L(y,f_{t-1}(x))$

我们本轮迭代的目标是找到一个弱学习器:

$h_t(x)$

使得本轮的损失函数最小化:

$L(y,f_t(x))=L(y,f_{t-1}(x)+h_t(x))$

此时的损失采用平方损失函数时:

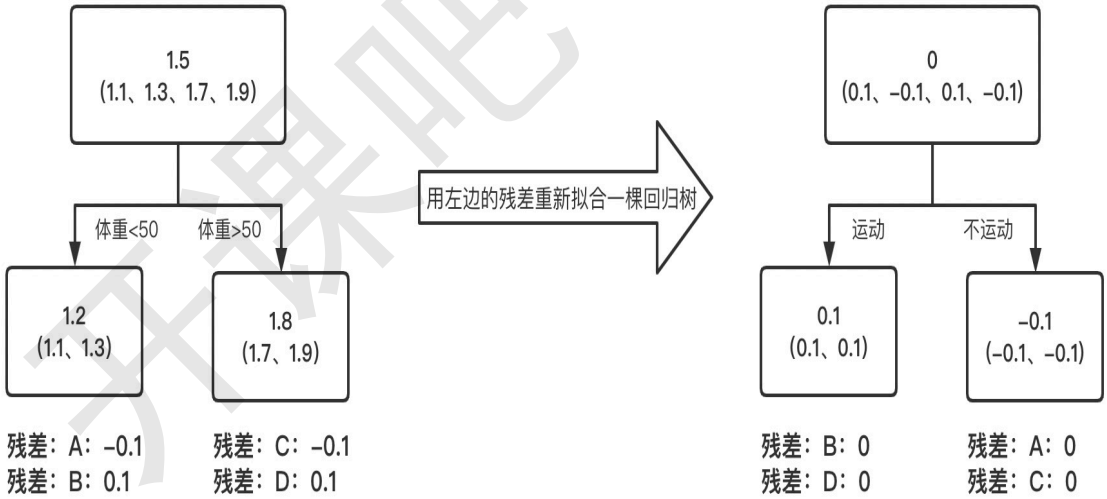
$L(y,f_{t-1}(x)+h_t(x))$

$=(y-f_{t-1}(x)-h_t(x))^2$

$=(r-h_t(x))^2$

此时有残差:

$r=y-f_{t-1}(x)$



- A的预测值：1.2 - 0.1 = 1.1
- B的预测值：1.2 + 0.1 = 1.3
- C的预测值：1.8 - 0.1 = 1.7
- D的预测值：1.8 + 0.1 = 1.9

4.3 GBDT（回归）

使用GBDT预测身高

GBDT的流程

输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$，$x_i \in X \subseteq R^n$，$y_i \in Y \subseteq R$；损失函数$L(y,f(x))$；

输出：回归树$\hat{f}(x)$

- （1）初始化弱学习器：

$$f_0(x)=\underset{c}{argmin}\sum_{i=1}^{N}L(y_i,c)$$

- （2）对m=1,2,...,M

（a）对i=1,2,...,N，计算负梯度（残差）：

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

（b）将a中得到的残差作为新样本的真实值，拟合一个回归树，得到第m棵树的叶子结点区域R_{mj}，$j=1,2,\ldots,J$。其中J为回归树的叶子结点的个数。

（c）对$j=1,2,\ldots,J$，计算最佳拟合值：

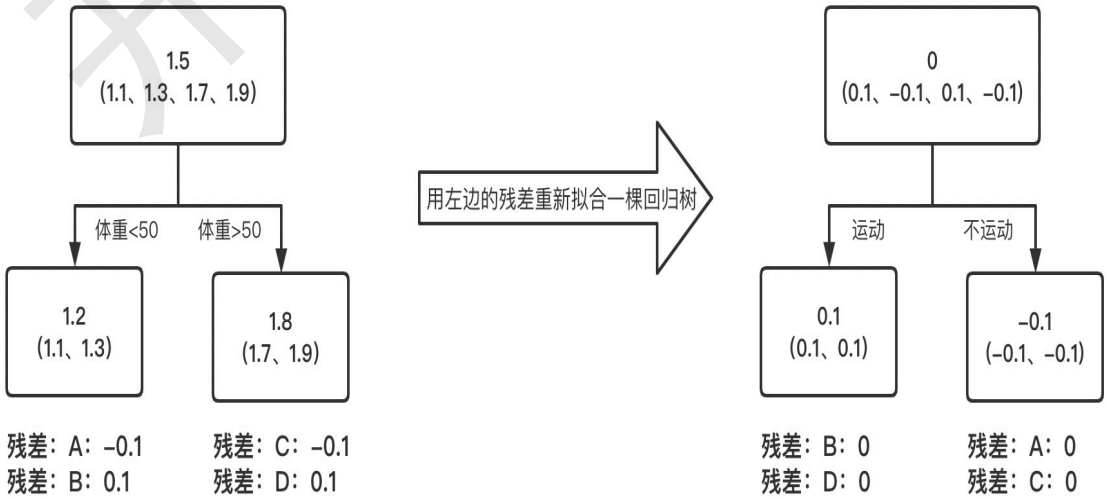
$c_{mj} = \underset{c}{argmin} \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c)$

（d）更新强学习器

$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$（I为指示函数）

- （3）得到最终的学习器

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



- A的预测值：1.2 - 0.1 = 1.1
- B的预测值：1.2 + 0.1 = 1.3
- C的预测值：1.8 - 0.1 = 1.7
- D的预测值：1.8 + 0.1 = 1.9

4.4 课堂提问

提升树和GBDT的主要区别在哪?

4.5 GBDT（分类）

从思想上看，GBDT分类算法和回归算法没有区别，但是对于分类来说我们的样本输出不是连续的值，而是离散的类别，这也导致我们无法直接从输出类别去拟合类别输出的误差。

GBDT二分类算法

我们使用类似于逻辑回归的对数似然函数时，损失函数为：

$L(y, f(x)) = log(1 + exp(-yf(x)))$

$y \in \{-1, +1\}$

则此时的负梯度误差为：

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

对于生成的决策树，我们各个叶子结点的最佳负梯度拟合值为：

$$c_{mj} = \underset{c}{argmin} \sum_{x_i \in R_{mj}} log(1 + exp(-y_i(f_{m-1}(x_i) + c)))$$

由于上式难以优化，我们一般使用近似值代替：

$$c_{mj} = \frac{\sum_{x_i \in R_{mj}} r_{mi}}{\sum_{x_i \in R_{mj}} |r_{mi}|(1-|r_{mi}|)}$$

除了负梯度计算和叶子结点的最佳负梯度拟合的线性搜索，GBDT二元分类和GBDT回归算法过程相同。

GBDT多分类算法

假设类别为K，则此时我们的对数似然损失函数为：

$$L(y, f(x)) = -\sum_{k=1}^{k} y_k log(p_k(x))$$

其中如果样本输出类别为k，则$y_k = 1$。第k类的概率$p_k(x)$的表达式为：

$$p_k(x) = \frac{exp(f_k(x))}{\sum_{l=1}^{k} exp(f_l(x))}$$

集合上两式，我们可以计算出第m轮的第i个样本对应类别l的负梯度误差为：

$$r_{mil} = -[\frac{\partial L(y,f(x_i))}{\partial f(x_i)}]_{f(x)=f_{l,m-1}}(x) = y_{il} - p_{l,m-1}(x_i)$$

根据上式我们就能够知道，这里得到的负梯度的误差其实就是真实概率和m-1轮预测概率的差值。

对于生成的决策树，我们各个叶子结点的最佳负梯度拟合值为：

$$c_{mjl} = \underset{c_{jl}}{argmin} \sum_{i=0}^{m} \sum_{k=1}^{k} L(y_k, f_{m-1,l}(x) + \sum_{j=1}^{J} c_{jl} I(x_i \in R_{mjl}))$$

同理我们使用近似值代替：

$$c_{mjl} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{mjl}} r_{mil}}{\sum_{x_i \in R_{mjl}} |r_{mil}|(1-|r_{mil}|)}$$

除了负梯度计算和叶子节点的最佳负梯度拟合的线性搜索，GBDT多分类和GBDT二分类以及GBDT回归算法过程相同。

4.6 GBDT的过拟合

为了防止过拟合问题的产生，我们也需要对GBDT采取一些措施，主要的方式有以下的三种：

- 剪枝

剪枝是树模型中常用的防止过拟合的方法，在GBDT中同样适用。

- 定义步长

我们把步长定义为v，对于前面的弱学习器的迭代：

$$f_k(x) = f_{k-1}(x) + h_k(x)$$

如果我们加上了正则化项，则有：

$$f_k(x) = f_{k-1}(x) + vh_k(x)$$

v的取值范围为(0,1]，对于同样的训练集学习效果，较小的v意味着我们需要更多的弱学习器的迭代次数，通常我们用步长和迭代最大次数一起来决定算法的拟合效果。

- 子采样比例

我们还可以通过子采样比例的方式来防止过拟合，取值为(0,1]，这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但是会增加样本拟合的偏差，因此取值不能太低。推荐在[0.5, 0.8]之间。

使用了子采样的GBDT有时也称作随机梯度提升树(Stochastic Gradient Boosting Tree, SGBT)。由于使用了子采样，程序可以通过采样分发到不同的任务去做boosting的迭代过程，最后形成新树，从而减少弱学习器难以并行学习的弱点。

4.7 GBDT的优缺点

优点

- 预测精度高；
- 适合低维数据；
- 能处理非线性数据；
- 可以灵活处理各种类型的数据，包括连续值和离散值；
- 在相对少的调参时间情况下，预测的准备率也可以比较高。

缺点

- 由于弱学习器之间存在依赖关系，难以并行训练数据。不过可以通过自采样的SGBT来达到部分并行；
- 如果数据维度较高时会加大算法的计算复杂度。

4.8 GBDT算法

GBDT回归算法

```
from sklearn.datasets import make_regression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

X, y = make_regression(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
reg = GradientBoostingRegressor(random_state=0)
reg.fit(X_train, y_train)
reg.predict(X_test[1:2])
reg.score(X_test, y_test)
```

```
array([-61.05212593])
```

```
x
```

```
array([[-1.5415874 ,  0.22739278, -1.35338886, ...,  0.86727663,
         0.40520408, -0.06964158],
       [-1.02125401,  0.39232256,  0.42971434, ..., -0.75693055,
         1.5038265 , -0.036413  ],
       [ 0.65880214, -0.65276145, -1.18488744, ..., -0.91798431,
        -0.04648116, -1.21868383],
       ...,
       [-0.09834549, -1.2550679 , -0.04217711, ..., -0.41783444,
         0.31709817,  0.60367669],
       [ 0.32099947, -1.12658943, -0.33945432, ...,  1.60286363,
         0.27623985, -0.67832984],
       [ 1.83708069, -0.00296475,  1.80144921, ...,  0.11230769,
         0.33109242,  1.51848293]])
```

```
y
```

```
array([ 5.89981499e+01, -1.51472301e+02,  3.92774675e+00,  1.30275835e+02,
        2.04728060e+00,  1.01587138e+02, -1.81389163e+02, -1.99827729e+02,
        2.36839440e+02,  3.00206479e+02,  2.71801441e+01,  2.76530071e+01,
        4.44317863e+01, -2.97416896e+01, -1.65646056e+02,  1.06302533e+02,
       -2.38325493e+02, -1.27217684e+02, -1.49696870e+02,  3.91961166e+01,
       -3.38549645e+00,  1.44280015e+02, -6.77946699e+01, -8.78067763e+01,
        2.59043702e+02, -1.98514664e+02,  1.24378523e+01, -7.03056179e+01,
        1.53948181e+02, -1.12712395e+02,  6.01916320e+01, -1.02364589e+02,
        8.02760399e+01, -2.54989466e+02, -2.65026472e+01,  8.45554208e+01,
       -7.86065692e+01,  6.59383309e+00, -8.40267190e+01, -1.88440481e+02,
       -1.05322661e+02, -1.48609151e+02,  3.19325058e+02, -1.03811985e+01,
       -1.66106829e+02, -1.92127360e+00, -1.66459184e+02,  2.22568255e+02,
       -1.90733117e+02, -4.47361111e+00, -1.40862199e+02, -1.04998874e+02,
       -1.55406951e+02,  8.84001312e+01,  1.10668159e+01,  2.17822668e+02,
       -2.30191323e+02, -9.53599988e+01,  1.43696372e+02, -1.51545011e+01,
       -6.01995736e+01, -7.87414903e+01,  5.49872561e+01, -9.60694579e+01,
        1.02073292e+02, -1.82433407e+02,  1.43967921e+01,  2.57273352e+01,
       -3.11997026e+01, -6.92314397e+01, -1.55678062e+02, -2.42752747e+01,
       -1.92480844e+01,  2.01408422e+02, -1.61883070e+02,  2.53833013e-01,
        1.66969921e+02,  7.10931137e+01, -4.85818195e+01, -3.12184318e+01,
       -2.73168840e+02,  3.00308517e+00, -1.33474939e+01,  1.21349806e+02,
       -7.92412566e+01,  4.61451813e+01, -2.39554177e+02, -4.81780542e+00,
```

```
        4.82434320e+01,  5.70661767e+01,  5.59382489e+01,  1.77274799e+02,
       -5.81475360e+01, -1.15194499e+02,  1.17344962e+02,  5.81688314e+01,
       -7.85896400e+01,  1.59876618e+01, -2.50470392e+02, -2.19074129e+02])
```

GradientBoostingRegressor的主要参数

- n_estimators : int (default=100)

梯度提升的迭代次数，也是弱分类器的个数

- learning_rate : float, optional (default=0.1)

SGB（随机梯度提升）的步长，也叫学习速度，一般情况下learning_rate越低，n_estimators越大；经验表明learning_rate越小，测试误差越小；

- max_depth : integer, optional (default=3)

决策树桩（Decision Stump）的最大深度,预剪枝操作（这里的树深度不包括树根）

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import ensemble
from sklearn import datasets
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# 加载数据
boston = datasets.load_boston()
X = boston.data
y = boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=13)

# 设置参数
# 1、迭代次数 2、最大深度 3、最小样本数 4、学习率 5、损失函数(最小二乘)
params = {
    'n_estimators': 500,
    'max_depth': 4,
    'min_samples_split': 0.5,
    'learning_rate': 0.01
}
clf = ensemble.GradientBoostingRegressor(**params)

clf.fit(X_train, y_train)
mse = mean_squared_error(y_test, clf.predict(X_test))
print("MSE: %.4f" % mse)

# 训练误差
```

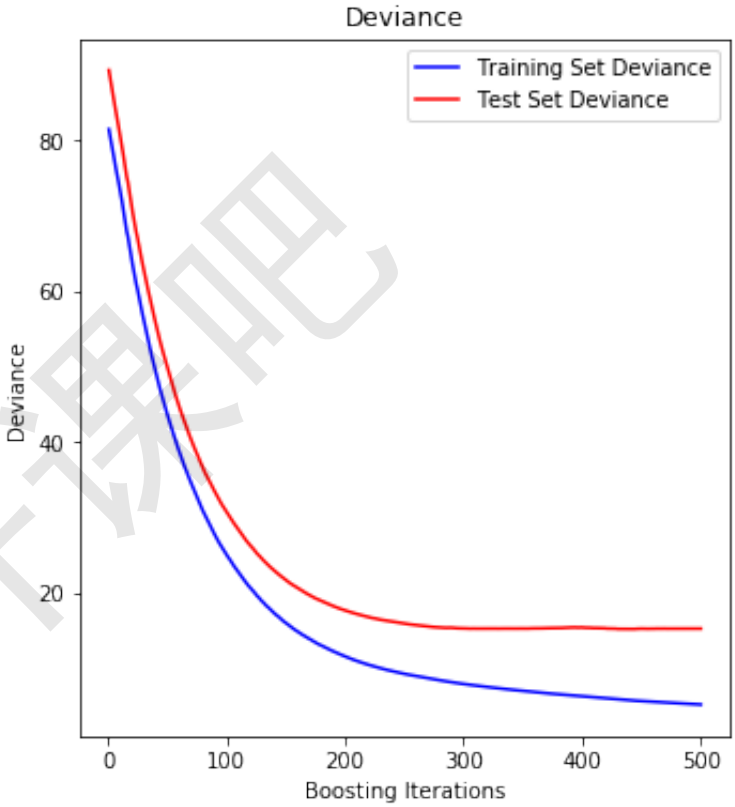
```
# 测试误差
test_score = np.zeros((params['n_estimators'], ), dtype=np.float64)

# staged_predict():预测每一轮迭代后输入样本的预测值
for i, y_pred in enumerate(clf.staged_predict(X_test)):
    test_score[i] = clf.loss_(y_test, y_pred)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('Deviance')
plt.plot(np.arange(params['n_estimators']) + 1,
         clf.train_score_,
         'b-',
         label='Training Set Deviance')
plt.plot(np.arange(params['n_estimators']) + 1,
         test_score,
         'r-',
         label='Test Set Deviance')
plt.legend(loc='upper right')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
```

```
MSE: 15.3039
```

```
Text(0, 0.5, 'Deviance')
```



GBDT分类算法

```
from sklearn.datasets import make_hastie_10_2
from sklearn.ensemble import GradientBoostingClassifier

X, y = make_hastie_10_2(random_state=0)
X_train, X_test = X[:2000], X[2000:]
y_train, y_test = y[:2000], y[2000:]

clf = GradientBoostingClassifier(n_estimators=100,
                                 learning_rate=1.0,
                                 max_depth=1,
                                 random_state=0).fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.913
```

GradientBoostingClassifier的主要参数

- n_estimators : int (default=100)

梯度提升的迭代次数，也是弱分类器的个数

- learning_rate : float, optional (default=0.1)

SGB（随机梯度提升）的步长，也叫学习速度，一般情况下learning_rate越低，n_estimators越大；经验表明learning_rate越小，测试误差越小；

- max_depth : integer, optional (default=3)

决策树桩（Decision Stump）的最大深度,预剪枝操作（这里的树深度不包括树根）

```
import gzip
import pickle as pkl
from sklearn.model_selection import train_test_split


def load_data(path):
    f = gzip.open(path, 'rb')
    train_set, valid_set, test_set = pkl.load(f, encoding='latin1')
    f.close()
    return (train_set, valid_set, test_set)


path = 'mnist.pkl.gz'
train_set, valid_set, test_set = load_data(path)

Xtrain, _, ytrain, _ = train_test_split(train_set[0],
                                        train_set[1],
                                        test_size=0.9)
Xtest, _, ytest, _ = train_test_split(test_set[0], test_set[1], test_size=0.9)
print(Xtrain.shape, ytrain.shape, Xtest.shape, ytest.shape)
```

```
(5000, 784) (5000,) (1000, 784) (1000,)
```

```
ytrain
```

```
array([4, 1, 6, ..., 1, 2, 8])
```

```
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
import time

clf = GradientBoostingClassifier(n_estimators=10,
                                 learning_rate=0.1,
                                 max_depth=3)
```

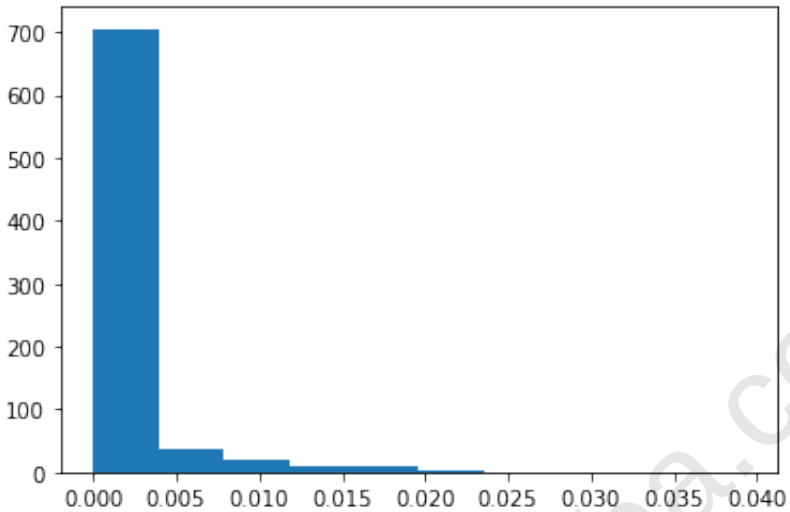
```
start_time = time.time()
clf.fit(Xtrain, ytrain)
end_time = time.time()
print('The training time = {}'.format(end_time - start_time))

#prediction and evaluation
pred = clf.predict(Xtest)
accuracy = np.sum(pred == ytest) / pred.shape[0]
print('Test accuracy = {}'.format(accuracy))
```

```
The training time = 19.139846801757812
Test accuracy = 0.837
```

```
# 查看特征重要性
%matplotlib inline
import matplotlib.pyplot as plt
plt.hist(clf.feature_importances_)
print(max(clf.feature_importances_), min(clf.feature_importances_))
```

```
0.03926870021407323 0.0
```



4.9 Shrinkage(衰减) 与 Step(步长)

- Shrinkage（衰减）是 GBDT 算法的一个重要演进分支，目前大部分的源码都是基于这个版本的。

  核心思想在于：Shrinkage 认为每次走一小步来逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易防止过拟合。

  也就是说，它不信任每次学习到的残差，它认为每棵树只学习到了真理的一小部分，累加的时候只累加一小部分，通过多学习几棵树来弥补不足。

具体的做法就是：仍然以残差作为学习目标，但是对于残差学习出来的结果，只累加一小部分（step* 残差）逐步逼近目标，

step 一般都比较小 0.01-0.001, 导致各个树的残差是渐变而不是陡变的。本质上，Shrinkage 为每一颗树设置了一个 weight，

累加时要乘以这个 weight，但和 Gradient 没有关系。

五、总结

- 关于DT: GDBT 中的树全部都是回归树，核心就是累加所有树的结果作为最终结果。

  只有回归树的结果累加起来才是有意义的，分类的结果加是没有意义的。GDBT 调整之后

  可以用于分类问题，但是内部还是回归树。
- 关于GB:首先 Boosting 是一种集成方法。通过对弱分类器的组合得到强分类器，他是串行的，

  几个弱分类器之间是依次训练的。GBDT 的核心就在于，每一颗树学习的是之前所有树结论和的残差。
- 结果逼近:核心思想在于：每次走一小步来逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易防止过拟合。也就是说，它不信任每次学习到的残差，它认为每棵树只学习到了真理的一小部分，累加的时候只累加一小部分，通过多学习几棵树来弥补不足。

六、作业

- 自己推导提升树和GBDT的实现过程
- 预习XGBoost算法