2018绿色计算大赛-大数据题解决方案

By: Chuanyu Xue

之前参加过很多的大数据比赛,大多是在天池和Kaggle和DataCastle打的,这次参加绿色计算大赛有几个地方感觉还是比较有意思的,主要有这么几个方面.

---

1. 仅能使用numpy和pandas算法包,这就需要对算法底层有所了解,这样的坏处是很多骚操作缺少第三方包是搞不起来的,但是好处是一大批调包侠都被劝退了,所以说平时各种模型还是不能光看,也要自己动手实现一下.(Do machine learning as an engineer instead of an algorithm expert)
2. 必须在线上运行代码,而且线上存在算法时间限制.这也就要求对代码能力有一定的要求,如果写的代码太差,在线上运行可能会跑不完.
3. 数据量非常的小,但是数据非常的规整,分布非常的平稳.这一点就导致线下线上比较一致,模型的效果很容易评估.
4. 只有一天的时间,非常的紧凑.平时干一个月的比赛,一天就要干完,这里还是要感谢我们孙老师的后勤照顾,给我们买了小零食,帮我们订外卖,大家做起来都非常开心.

模拟赛

模拟赛链接: https://www.educoder.net/shixuns/ir768pfq/challenges (https://www.educoder.net/shixuns/ir768pfq/challenges)

> 利用已有的同现标签数据以及给出的20位开发者技能标签数据，推荐兴趣标签给这20位开发者。
>
> 其中，同现标签指的是：“共同出现的标签”。比如，某技术问答帖子的标签为<java>,<spring>,<mybatis>，这三个标签在该帖子的标签集合中共同出现，则称其为同现标签。
>
> 推荐标签指的是：“根据同现标签的数据和开发者自己的技能标签来推荐给该开发者的标签”，比如，某位开发者的技能标签为<java>,<spring>，根据同现标签数据，可以推荐/<mybatis/>给开发者。

具体意思就是训练集给了我们很多标签的组合,例如 java,spring,mybatis是一个组合, cocox2d,cpp,game是一个组合, 这些组合代表了某个帖子里面共同出现的标签,这就是这些标签是具有一定关联的,我们要挖掘这些关联关系. 然后测试集会给我们每个程序员的标签, 以及这些标签的权重, 我们需要根据这些标签和权重, 选出应该给这些程序员推荐的内容

第一反应是这是不是推荐系统, 因为之前在研究CF, Contend-based之类的推荐系统, 但是想想和广义的推荐系统还是不一样的, 推荐系统往往考虑的是两个对象或者两类节点的问题, 而这个训练集只提供了一种对象, 所以就有点蒙. 想起来谷歌大神Martin Zinkevich说的机器学习四十七条原则的第一条, Don’t be afraid to launch a product without machine learning. 那就试试搞个规则吧.

最开始的思路是, 通过训练集建立标签与标签的关联, 建立关联之后, 通过用户某个标签的某个权重, 去选择TOP-N个关联标签做推荐. 但是想想这样好像有点不太合理, 如果一个用户关注了Python, 并不能给它推荐BigData, 但是

如果一个用户同时关注了Python和Probability, 那么给它推荐BigData好像就比较准了, 可是如何将两个标签关联起来呢? 我们采取的方法是选取Python关联的Tag和Probability关联的Tag取交集的方式, 然后在交集中选取TOP-N个元素. 这里还存在一个选取多少个元素合适, 这个选择方式可以根据线上的反馈来选取. 由于不排名, 我们很难知道最后结果在什么位置, 大概200分+.

In [1]:

```
import pandas as pd
import numpy as np

test = pd.read_csv('user_tag.csv')
data = pd.read_csv('tag_cooccurrence.csv')
data['tags'] = data['tags'].str.split(',')

#统计每个tag关联次数最多的
asso_hash = {}
for index, row in data.iterrows():
    line = row['tags']
    for i in range(len(line)):
        for k in range(i+1, len(line)):
            (i_,k_) = (line[i], line[k])
            if i_ not in asso_hash:
                asso_hash[i_] = {}
            if k_ not in asso_hash:
                asso_hash[k_] = {}
            if k_ not in asso_hash[i_]:
                asso_hash[i_][k_] = 0
            if i not in asso_hash[k_]:
                asso_hash[k_][i_] = 0
            asso_hash[i_][k_] += 1
            asso_hash[k_][i_] += 1
for i in asso_hash:
    asso_hash[i] = sorted(asso_hash[i].items(), key=lambda v: v[1], reverse=True)

#根据共同关联度推荐
results = []
for index,row in test.iterrows():
    line = row['origin_tags'].split(',')
    weights = list(map(float,row['tag_value'].split('[')[1].split(']')[0].split(','
    summ = sum(weights)
    for i in range(len(weights)):
        weights[i] /= summ
    reserve = []
    for i in range(5):
        label_this = line[i]
        weight_this = weights[i]
        listt = asso_hash[label_this]
        reserve.append(listt[:int(len(listt) * weight_this)])
    count_hash = {}
    for i in reserve:
        for k in i:
            label = k[0]
            if label not in count_hash:
                count_hash[label] = 0
            count_hash[label]+=1
    count_hash = sorted(count_hash.items(), key=lambda v: v[1], reverse=True)
    results.append(list(set([x[0] for x in count_hash[:10]])))

resultt = []
for i in results:
    listt = ''
    for k in i:
        listt+= k + ','
    resultt.append(listt[:-1])
```

```
upload = pd.DataFrame()
upload['id'] = list(range(1,21))
upload['recommand_tags'] = resultt
```

预赛

预赛链接: https://www.educoder.net/tasks/mp76jai92oct (https://www.educoder.net/tasks/mp76jai92oct)

到了预赛感觉问题变得更普遍了一些, 任务是典型的预测任务, 预测任务是: 现在有一家公司拥有很多关于客户购买公司产品基本评价的数据，需要你根据这些数据来预测客户对于产品的满意率. 但事实上这里是没有任何产品和用户标识的, 也就是说我们实际上是需要通过这些产品的评价来对产品做一个数值上的刻画. 最后的结果是AUC指标, 在广义上讲AUC指标应该=正确预测数/预测总数, 但是要求上传的却是概率, 实际上处理的时候我们采用的是sigmod损失函数.

EDA & Feature Enginnering

数据决定了模型的上限, 模型逼近这个上限, 做好数据分析工作往往是最重要的. 这个题提供了很多的匿名特征, 在不知道特征意义的情况下, 处理这些特征成了一个比较棘手的问题.

In [2]:

```
data = pd.read_csv("train_final.csv")
```

首先我们发现这些数值都是一些比较正常的数值,这样我们就没必要再考虑对偏离过度的数据做处理. 有一些特征有些问题, 比如像ps_ind_10_bin 这个特征, 它的方差是0, 对于这类的特征我们选择对他们进行剔除.

In [3]:

```
data.loc[:, data.std() == 0].columns
```

Out[3]:

```
Index(['ps_ind_10_bin', 'ps_ind_13_bin'], dtype='object')
```

还有问题是特征的区间并不一致, 有些特征偏离远点很多, 但是有些特征却集中在0-1之间, 由于我们最后使用的是线性模型, 数值较大的特征将会对模型产生较大的影响, 因此我们采用了ZScore归一化对数据了进行处理.

In [4]:

```
print(sum(data.max() > 10), sum(data.min() < 1))
feature = [x for x in data.columns if x not in ['id', 'target']]
data[feature] -= np.mean(data[feature], axis=0)
data[feature] /= np.std(data[feature], axis=0)
```

```
10 53
```

In [5]:

```
feature = data[[x for x in data.columns if x not in['TARGET','ID']]]
feature -= np.mean(feature, axis=0)
feature /= np.std(feature, axis=0)
data[[x for x in data.columns if x not in['TARGET','ID']]] = feature
```

很多特征是存在缺失值的, 我们的线性模型无法自动处理缺失值, 有一种方法是删除带有缺失值的样本, 但是本身样本空间比较小, 这时候我们采取了使用平均值填补的保守策略

In [6]:

```
data = data.fillna(data.mean()).fillna(np.random.normal(loc=0.0, scale=1e-5))
```

虽然是匿名特征, 但是特征名字中都含有一些前缀, 考虑是否会有什么特殊意义呢. 后来经过分析我们发现了一个规律, 所有带ind的特征都是离散的特征, 而对于离散特征有很多处理的方法, 这里我们提取了特征对应Label的平均值 与 特征出现的次数作为原始离散特征的替代. 至于为什么不使用OneHot或者Dummy Coding等较为普遍的离散数值处理方法, 是因为离散特征过多, 如果都做ONEHOT的话特征数量会暴增, 线性模型下很可能出现复共线性的问题.

In [7]:

```
#获取所有ind特征
ind_features = []
for i in data.columns:
    if i[:3] == 'ind':
        ind_features.append(i)
for i in ind_features:
    data[[i+'a', i+'b']] = data.groupby(i, as_index=False).agg({'TARGET':['count','
```

其次选用线性模型的话, 不具有线性相关的特征实际上是对模型不会起到作用的, 因此我们剔除了一些非线性相关的特征, 具体的方式是计算了它们的皮尔逊相关系数.

In [8]:

```
featureloc = [index for index,value in enumerate(list(pd.DataFrame(np.corrcoef(data
```

Model

我们选择的是简单的以Sigmod作为损失函数的逻辑回归作为模型, 之所以选择这个模型一方面是其他模型短时间内难以实现, 其次是我们相信在特征足够好的时候精力不必放在模型上太多.

但是这里也有很多要注意的坑, 其中一点是Sigmoid的底数中由于是指数函数, 非常容易发生精度溢出以及数值溢出, 这里要格外的小心

In [9]:

```
#逻辑回归放在这里
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

def gradAscent(dataArray, labelArray, alpha, maxCycles):
    dataMat=np.mat(dataArray)
    labelMat=np.mat(labelArray)
    labelMat = labelMat.reshape((-1,1))
    m,n=dataMat.shape
    weigh=np.ones((n,1))
    for i in range(maxCycles):
        h=sigmoid(dataMat * weigh)
        error=labelMat-h
        weigh=weigh + alpha * dataMat.transpose() * error
        if np.sum(np.square(alpha * dataMat.transpose() * error)) < 1e-5:
            break
    return weigh

def classfy(dataArray, weigh):
    dataMat=np.mat(dataArray)
    h=sigmoid(dataMat*weigh)
    return h
```

具体的建模方式, 我们没有做线下的validation, 因为线上是可以提交无限次的, 因此我们可以直接通过提交选择最佳参数

In [10]:

```
train = data[~np.isnan(data['target'])]
train_X = train.drop(['id','target'],axis=1)
train_y = train['target']

test = data[np.isnan(data['target'])]

test_X = test.drop(['id','target'],axis=1)
weight = gradAscent(train_X, train_y, 1e-5, 10000)

h = classfy(test_X, weight)
```

Trick

1. 其次我们还使用了一些小技巧, 首先是我们的模型最后得出的结果不是无偏的, 因此如果乘以一个系数可能会使最后的结果更好, 可以乘个0.9或者1.1什么的, 经过线上测试找到最好的系数.
2. 另一个技巧是, 由于我们的估计是不精确的, 因此一定存在着一个常数c属于0to1的区间, 使用c预测总可以比任意小于c的预测值预测效果要好, 同样也存在一个值d使得总可以比用大于d的值预测效果要好, 经过线上测试我们也可以尝试出最好的c与d

决赛

决赛链接: https://www.educoder.net/tasks/znbifyqc3mow (https://www.educoder.net/tasks/znbifyqc3mow)

实际上决赛的赛题与预赛是很相似的, 都是最普通的回归问题, 同样的是AUC指标与匿名特征, 本来觉得会是比较复杂的时间序列问题或者推荐问题呢. 不过也存在一些很有技巧性的东西需要我们去做. 这次的评分方式也有了很大的区别, 刻意让得分低的队伍最后的总分也低哈哈, 这样挺公平的, 毕竟最后一个阶段也是最耗费精力的最容易拉开差距的.

任务是去过用户是否购买保险的预测, 提供的数据要比预赛多一些.

EDA & Feature Engineering

In [11]:

```
data = pd.read_csv("train.csv")
```

一拿到数据, 发现有很多数据都特别不正常, 很多数据的特征值在0到25之间的离散值, 有些特征最大值是67145, 严重偏离了其他特征. 看来数据方面要下很大的功夫了.

In [12]:

```
print(np.sum(data.max() == 25), np.sum(data.max() > 10000))
```

```
33 2
```

首先做的事情还是把只有单一类别的特征删掉, 这些特征是totally useless的

In [13]:

```
data[['GeographicField10B','PersonalField39']].std()
```

Out[13]:

```
GeographicField10B    0.0
PersonalField39       0.0
dtype: float64
```

我们又分析了这些最大值在0到25的特征, 但是我们对这些特征并没有进行变换, 虽然直觉上做变换是正确的, 但是我们分析了这些特征和target具有比较强的相关性, 所以我们怀疑这些特征是数值上有意义的, 例如通过中位数或者其他分箱方法把连续值划分到了连续空间.

In [14]:

```
sum(data.std() < 0.1)
```

Out[14]:

7

在决赛中, 为了处理各个变量度量方式不同的问题, 我们对数据同样进行了归一化, 但是这次没有使用和预赛相同的ZScore变换, 而是采取了Min-Max变换, 考虑的是很多变量的方差实在是太小了, 做zscore一方面会损失值的信息, 不利于上面的那些分箱变换特征, 另一方面方差太小反而会让值变得很大.

In [15]:

```
feature = [x for x in data.columns if x not in ['ID','TARGET']]
data[feature] = (data[feature] - np.min(data[feature], axis=0)) / (np.max(data[feat
```

给神经网络模型定制的特殊特征

这次决赛我们搬出来了大杀器--手写MLP神经网络, 估计像我们一样手写好几百行神经网络放到网页上跑的队伍应该不多吧哈哈, 大家都知道神经网络的特点是可以处理特征空间很大的数据集, 不用手动挑特征, 解释性差但是效果非常好. 原有的数据给神经网络岂不是太浪费了, 因此我们对神经网络的数据进行了深度的定制, 最后做出来大概是1000多维的数据吧.

首先是我们为那些离散的特征进行了预赛我们做过的均值与次数特征的重塑, 因为不再考虑特征的数量, 我们为每一个离散特征都进行了OneHot处理!

In [16]:

```
big_categories = [x for x in data.columns if x not in ['ID','TARGET','SalesField8']]
for i in big_categories:
    temp =  data.groupby(i, as_index=False).agg({'TARGET':'mean'})
    temp.columns = [i,i+'_mean']
    data = pd.merge(data, temp, on=i, how='left')

for i in big_categories:
    temp = pd.DataFrame(data[i].value_counts()).reset_index()
    temp.columns = [i,i+'_count']
    data = pd.merge(data, temp, on=i, how='left')

splits = []
for i in big_categories:
    floor = pd.get_dummies(data[i])
    floor.columns = [(i + '_' + str(x)) for x in list(floor.columns)]
    for k in floor.columns:
        splits.append(k)
    data = pd.concat([data, floor], axis=1)
```

做完之后数据集的特征数量变为959维

In [17]:

```
data.shape
```

Out[17]:

```
(1824, 959)
```

后面我们还对特征进行了组合处理, 具体的方案是选取原始特征两两之间做点积的方案, 可以从原始数据中挖掘更多的信息, 但是最后由于一定的原因, 并没有提交到线上, 我认为是比较有创意的工作, 有些可惜了.

Model

这次我们在模型上下了很大的功夫, 一方面我们沿用了在预赛中表现不错的逻辑回归模型, 另一方面我们又搞了手写的神经网络与梯度提升决策树这两个机器学习大杀气, 但是做到最后却让我们有一点点失望.

神经网络代码:

In [18]:

```
from abc import ABCMeta, abstractmethod
class loss(metaclass=ABCMeta):
    def obj(self, pred, true):
        pass

    def gradient(self, pred, true):
        pass


class mse(loss):
    def obj(self, pred, true):
        return np.square(pred - true).mean() / 2

    def gradient(self, pred, true):
        return pred - true


class log_loss(loss):
    def obj(self, pred, true):
        return (-np.multiply(true, np.log(pred)) - np.multiply(1 - y, np.log(1 - pr

    def gradient(self, pred, true):
        return -np.multiply(true, 1 / pred) + np.multiply(1 - true, 1 / (1 - pred))


class act(metaclass=ABCMeta):

    def forward(self, matrix):
        pass

    def backward(self, matrix):
        pass


class linear(act):

    def forward(self, matrix):
        return matrix

    def backward(self, matrix):
        return np.ones_like(matrix)


class relu(act):

    def forward(self, matrix):
        return np.multiply(matrix > 0, matrix)

    def backward(self, matrix):
        return 1 * (matrix > 0)


class logistic(act):

    def forward(self, matrix):
        return 1 / (1 + np.exp(-matrix) + 0.000001)
```

```
    def backward(self, matrix):
        return np.multiply(self.forward(matrix), 1 - self.forward(matrix))


class MLP(object):

    def __init__(self,
                 n_hidden_units=100,
                 batch_size=200,
                 n_epochs=200,
                 learning_rate=0.01,
                 momentum=0.9,
                 weight_decay=0.0001,
                 activation='relu',
                 loss='mse'):

        self.n_hidden_units = n_hidden_units
        self.batch_size = batch_size
        self.n_epochs = n_epochs
        self.learning_rate = learning_rate
        self.momentum = momentum
        self.weight_decay = weight_decay

        if activation == 'relu':
            self.act1 = relu()
        elif activation == 'logistic':
            self.act1 = logistic()
        elif activation == 'linear':
            self.act1 = linear()
        else:
            self.act1 = activation
        if loss == 'mse':
            self.loss = mse()
            self.act2 = linear()
        elif loss == 'log_loss':
            self.loss = log_loss()
            self.act2 = logistic()
        else:
            self.loss = loss[0]
            self.act2 = loss[1]

    def forward(self):
        self.layer1 = self.W1 * self.X + self.b1
        self.layer1act = self.act1.forward(self.layer1)
        self.score = self.W2 * self.layer1act + self.b2
        self.pred = self.act2.forward(self.score)

    def backward(self):
        self.dpred = self.loss.gradient(self.pred, self.true)
        self.dscore = np.multiply(self.dpred, self.act2.backward(self.score))
        self.dlayer1act = self.W2.T * self.dscore
        self.dlayer1 = np.multiply(self.dlayer1act, self.act1.backward(self.layer1))

        self.dW1 = (self.dlayer1 * self.X.T - self.weight_decay * self.W1) / self.b
        self.db1 = np.sum(self.dlayer1, axis=1) / self.batch_size
        self.dW2 = (self.dscore * self.layer1act.T - self.weight_decay * self.W2) /
        self.db2 = np.sum(self.dscore, axis=1) / self.batch_size

    def update_weights(self):
        self.tW1 = self.momentum * self.tW1 + (1 - self.momentum) * self.dW1
        self.tb1 = self.momentum * self.tb1 + (1 - self.momentum) * self.db1
```

```
        self.tW2 = self.momentum * self.tW2 + (1 - self.momentum) * self.dW2
        self.tb2 = self.momentum * self.tb2 + (1 - self.momentum) * self.db2

        self.W1 -= self.tW1 * self.learning_rate
        self.b1 -= self.tb1 * self.learning_rate
        self.W2 -= self.tW2 * self.learning_rate
        self.b2 -= self.tb2 * self.learning_rate

    def fit(self, train, target):
        train = np.matrix(train).T
        target = np.matrix(target.values.reshape(-1, 1)).T

        n_features = train.shape[0]
        n_obs = train.shape[1]

        s1 = np.sqrt(6 / (n_features + self.n_hidden_units))
        s2 = np.sqrt(6 / (1 + self.n_hidden_units))
        self.W1 = np.matrix(np.random.uniform(-s1, s1, [self.n_hidden_units, n_feat
        self.b1 = np.matrix(np.random.uniform(-s1, s1, [self.n_hidden_units, 1]))
        self.W2 = np.matrix(np.random.uniform(-s2, s2, [1, self.n_hidden_units]))
        self.b2 = np.matrix(np.random.uniform(-s2, s2, [1, 1]))

        self.tW1 = self.W1 * 0
        self.tb1 = self.b1 * 0
        self.tW2 = self.W2 * 0
        self.tb2 = self.b2 * 0

        for i in range(self.n_epochs):
            for j in range(n_obs // self.batch_size):
                self.X = train[:, j * self.batch_size:(j + 1) * self.batch_size]
                self.true = target[:, j * self.batch_size:(j + 1) * self.batch_size
                self.forward()
                self.backward()
                self.update_weights()

    def predict(self, test):
        self.X = np.matrix(test).T
        self.forward()
        return np.squeeze(np.asarray(self.pred))
```

GBDT代码

In [19]:

```
def leaf_score(g,h,reg_lambda):
    return -np.sum(g)/(np.sum(h)+reg_lambda)

def leaf_loss(g,h,reg_lambda):
    return -0.5*np.square(np.sum(g))/(np.sum(h)+reg_lambda)

def calculate_gain(original_loss,feature,g,h,threshold,reg_lambda):
    left_g=0
    left_h=0
    right_g=0
    right_h=0
    for i in range(len(feature)):
        if feature[i]<threshold:
            left_g+=g[i]
            left_h+=h[i]
        else:
            right_g+=g[i]
            right_h+=h[i]
    left_loss=-0.5*np.square(left_g)/(left_h+reg_lambda)
    right_loss=-0.5*np.square(right_g)/(right_h+reg_lambda)
    return original_loss-left_loss-right_loss

def find_threshold(g,h,train,reg_lambda):
    loss=leaf_loss(g,h,reg_lambda)
    threshold=None
    best_gain=0
    unq=np.unique(train)
    for i in range(1,len(unq)):
        this_threshold=(unq[i-1]+unq[i])/2
        this_gain=calculate_gain(loss,train,g,h,this_threshold,reg_lambda)
        if this_gain>best_gain:
            threshold=this_threshold
            best_gain=this_gain
    return threshold,best_gain

def find_best_split(train,g,h,reg_lambda):
    train=train.T
    feature=0
    threshold=None
    best_gain=0
    for i in range(len(train)):
        this_threshold,this_gain=find_threshold(g,h,train[i],reg_lambda)
        if this_gain>best_gain:
            feature=i
            threshold=this_threshold
            best_gain=this_gain
    return feature,threshold,best_gain

class loss(metaclass=ABCMeta):
    @abstractmethod
    def link(self,score):
        pass

    @abstractmethod
    def g(self,true,score):
        pass
```

```
    @abstractmethod
    def h(self,true,score):
        pass

class mse(loss):
    '''Loss class for mse. As for mse, link function is pred=score.'''
    def link(self,score):
        return score

    def g(self,true,score):
        return score-true

    def h(self,true,score):
        return np.ones_like(score)

class log(loss):
    '''Loss class for log loss. As for log loss, link function is logistic transfor
    def link(self,score):
        return 1/(1+np.exp(-score))

    def g(self,true,score):
        pred=self.link(score)
        return pred-true

    def h(self,true,score):
        pred=self.link(score)
        return pred*(1-pred)


class GBDT(object):
    def __init__(self,
        loss='mse',
        max_depth=3,min_sample_split=10,reg_lambda=1,gamma=0,
        learning_rate=0.1,n_estimators=100):
        self.loss=loss
        self.max_depth=max_depth
        self.min_sample_split=min_sample_split
        self.reg_lambda=reg_lambda
        self.gamma=gamma
        self.learning_rate=learning_rate
        self.n_estimators=n_estimators

    def fit(self,train,target):
        self.estimators=[]
        if self.loss=='mse':
            self.loss=mse()
        if self.loss=='log':
            self.loss=log()
        self.score_start=target.mean()
        score=np.ones(len(train))*self.score_start
        for i in range(self.n_estimators):
            estimator=Tree(
                max_depth=self.max_depth,min_sample_split=self.min_sample_split,reg
            estimator.fit(train,g=self.loss.g(target,score),h=self.loss.h(target,sc
            self.estimators.append(estimator)
            score+=self.learning_rate*estimator.predict(train)
        return self

    def predict(self,test):

        score=np.ones(len(test))*self.score_start
```

```
        for i in range(self.n_estimators):
            score+=self.learning_rate*self.estimators[i].predict(test)
        return self.loss.link(score)


class TreeNode(object):
    def __init__(self,
        is_leaf=False,score=None,
        split_feature=None,split_threshold=None,left_child=None,right_child=None):
        self.is_leaf=is_leaf
        self.score=score
        self.split_feature=split_feature
        self.split_threshold=split_threshold
        self.left_child=left_child
        self.right_child=right_child

class Tree(object):

    def __init__(self,max_depth=3,min_sample_split=10,reg_lambda=1,gamma=0):
        self.max_depth=max_depth
        self.min_sample_split=min_sample_split
        self.reg_lambda=reg_lambda
        self.gamma=gamma

    def fit(self,train,g,h):
        self.estimator=self.construct_tree(train,g,h,self.max_depth)
        return self

    def predict(self,test):
        result=np.zeros(len(test))
        for i in range(len(test)):
            result[i]=self.predict_single(self.estimator,test[i])
        return result

    def predict_single(self,treenode,test):
        if treenode.is_leaf:
            return treenode.score
        else:
            if test[treenode.split_feature]<treenode.split_threshold:
                return self.predict_single(treenode.left_child,test)
            else:
                return self.predict_single(treenode.right_child,test)

    def construct_tree(self,train,g,h,max_depth):

        if max_depth==0 or len(train)<self.min_sample_split:
            return TreeNode(is_leaf=True,score=leaf_score(g,h,self.reg_lambda))

        feature,threshold,gain=find_best_split(train,g,h,self.reg_lambda)

        if gain<=self.gamma:
            return TreeNode(is_leaf=True,score=leaf_score(g,h,self.reg_lambda))

        index=train[:,feature]<threshold
        left_child=self.construct_tree(train[index],g[index],h[index],max_depth-1)
        right_child=self.construct_tree(train[~index],g[~index],h[~index],max_depth
        return TreeNode(split_feature=feature,split_threshold=threshold,left_child=
```

最后我们使用这三个模型调好参数融合后效果其实是不错的, 但是比较失望的一个地方是主办方在网页上设置了

300秒超时. 要知道即使是用LightGBM或者Xgboost或者Keras这样众多开发者一起努力写好的优化好的包也是不止300秒的. 所以我们在线上根本没有办法提交GBDT和MLP的最好参数, 本来想着做融合的计划也是泡汤了, 这方面说实话还是很遗憾的. 因为时间的限制, 我为NN做的定制数据由于数据量太大也没有办法使用, 希望主办方下次还是考虑一下这方面的问题, 让更多有创造性的工作可以实现.

最后我们选择了GBDT * 0.5 + LogicRegression * 0.5的简单Bagging方案，获得了决赛第二名的成绩.