XGBoost 实验报告

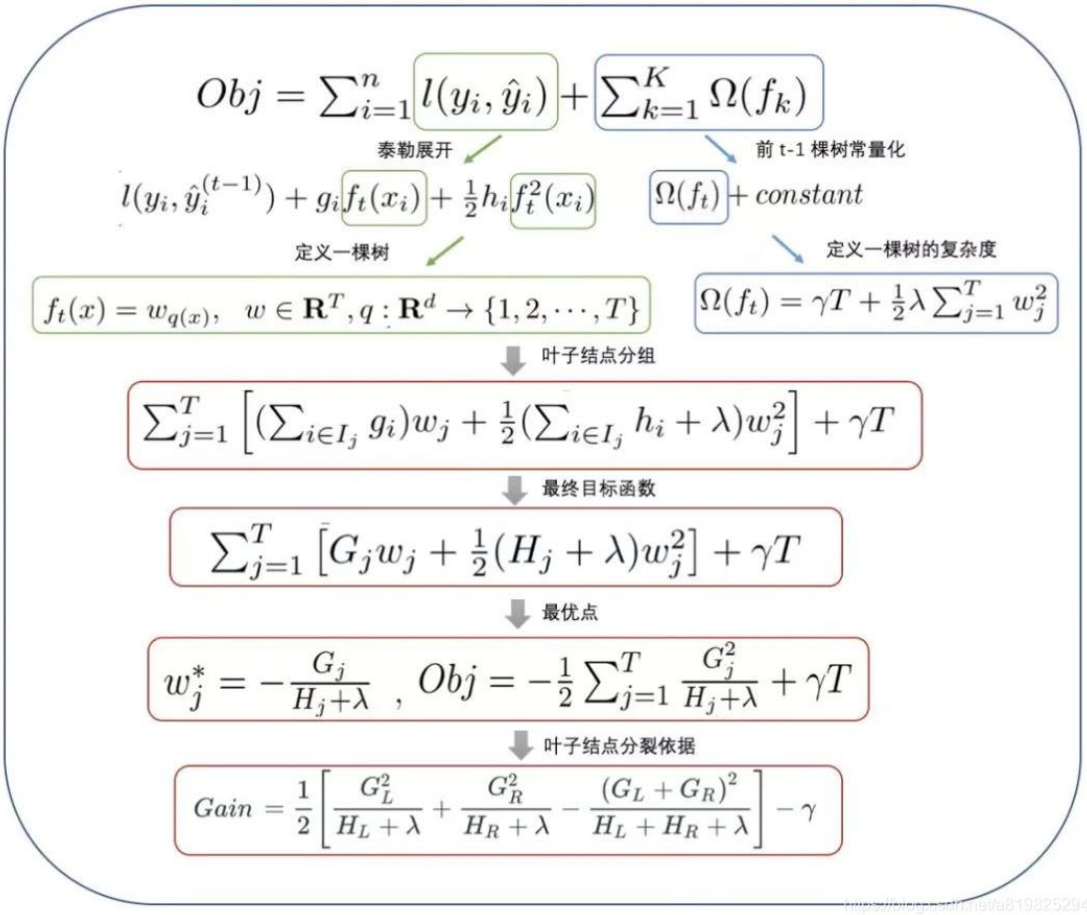
一、 任务描述

熟悉 XGBoost 的输入、输出和评价，并使用它完成一个分类 回归任务。

二、 XGBoost 算法概述

XGBoost 是 boosting 算法的其中一种。Boosting 算法的思想是将许多弱分类器集成在一起形成一个强分类器。因为 XGBoost 是一种提升树模型，所以它是将许多树模型集成在一起，形成一个很强的分类器。而所用到的树模型则是 CART 回归树模型。

该算法思想就是不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数，去拟合上次预测的残差。当我们训练完成得到 k 棵树，我们要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，每个叶子节点就对应一个分数，最后只需要将每棵树对应的分数加起来就是该样本的预测值。其大致推导示意图如下：



三、 实验方法

实验运用 python 在 jupyter notebook 环境下进行，在此之前需配置安装好 XGBoost 的包。

实验数据分为训练集、验证集和测试集，分别为同目录下的 train.csv、validation.csv 和 test.csv 文件。每个文件包含若干样本，每个样本包含 162 维的特征和样本标签。

通过对数据进行读取处理，将训练集数据放入 XGBoost 模型中进行训练。取在验证集上表现最好时的模型参数：100 棵树、学习率为 0.1。此时在训练集上的准确率达到 0.96124，在测试集上的各项指标分别能达到 Precesion: 0.6647，Recall: 0.4883，F1-score: 0.5630，Accuracy: 0.7814，AUC: 0.8246。最后选出了对分类器影响较大的一些特征。

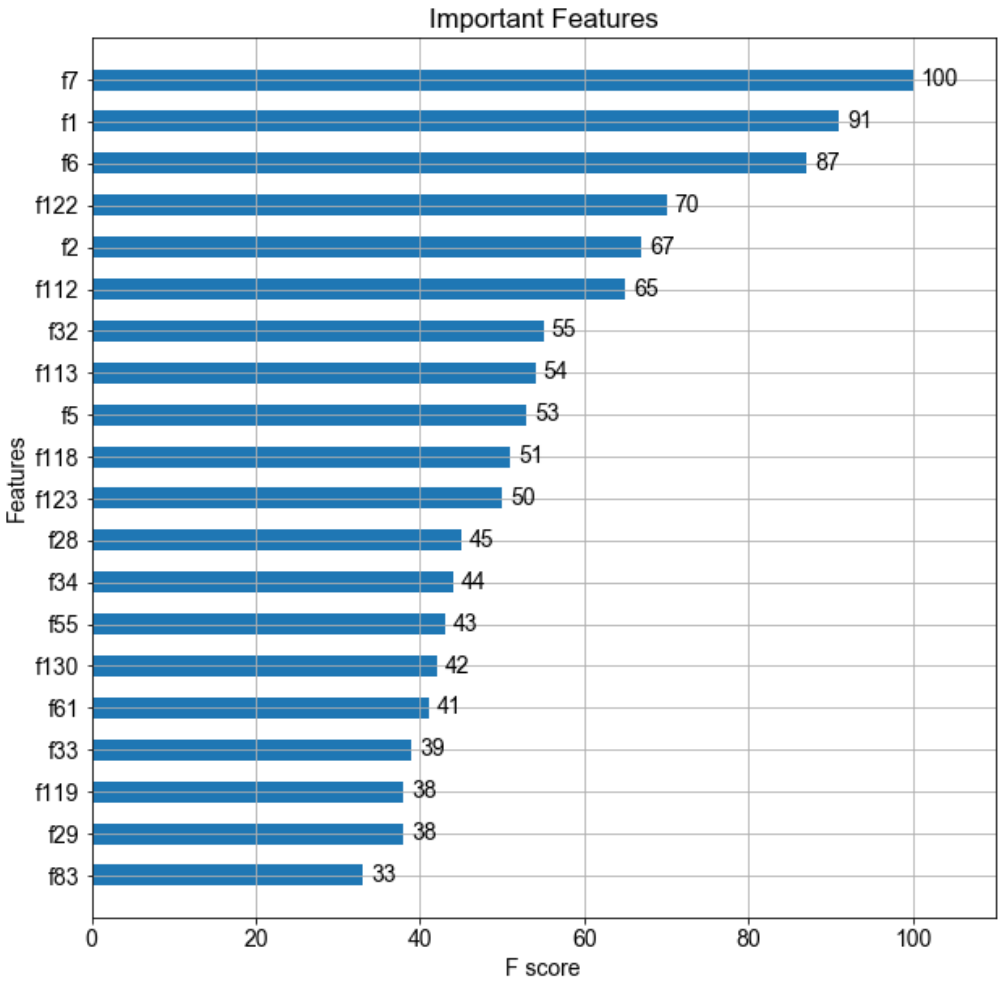
四、 结果展示

实验结果如下：

```
[71]    train-auc:0.94571
[72]    train-auc:0.94602
[73]    train-auc:0.94714
[74]    train-auc:0.94796
[75]    train-auc:0.94886
[76]    train-auc:0.94947
[77]    train-auc:0.95004
[78]    train-auc:0.95066
[79]    train-auc:0.95143
[80]    train-auc:0.95212
[81]    train-auc:0.95250
[82]    train-auc:0.95299
[83]    train-auc:0.95369
[84]    train-auc:0.95394
[85]    train-auc:0.95467
[86]    train-auc:0.95506
[87]    train-auc:0.95576
[88]    train-auc:0.95621
[89]    train-auc:0.95674
[90]    train-auc:0.95701
[91]    train-auc:0.95784
[92]    train-auc:0.95823
[93]    train-auc:0.95861
[94]    train-auc:0.95908
[95]    train-auc:0.95948
[96]    train-auc:0.95978
[97]    train-auc:0.96007
[98]    train-auc:0.96068
[99]    train-auc:0.96124
```

```
----------for validation data----------
Precesion: 0.7211
Recall: 0.6299
F1-score: 0.6724
Accuracy: 0.8195
AUC: 0.8793
验证集每个样本的得分
 [0.24728023 0.05158438 0.12027677 ... 0.02691913 0.10059986 0.1480652 ]
验证集每棵树所属的节点数
 [[62 44 48 ... 35 43 31]
 [62 44 48 ... 35 44 32]
 [62 44 48 ... 35 31 31]
 ...
 [62 44 48 ... 36 39 32]
 [62 59 48 ... 34 39 44]
 [62 44 48 ... 24 39 28]]
特征的重要性
 [[ 4.9120304e-03 -2.0864698e-01 -2.9899746e-02 ... -1.9843459e-02
   3.1086013e-02 -8.7671363e-01]
 [-5.3086676e-02 -3.7335449e-01 -3.7518807e-02 ... -1.2389141e-02
   4.4614300e-02 -8.7671363e-01]
 [-3.8523763e-02 -3.4690037e-01 -7.7052927e-04 ... -8.4215356e-03
   5.9061825e-02 -8.7671363e-01]
 ...
 [-3.8322303e-02 -3.0549768e-01 -5.6034375e-02 ... -2.5911083e-02
   5.9252385e-02 -8.7671363e-01]
 [-4.9462125e-02 -6.3565046e-02  8.1148155e-02 ... -3.5441007e-02
   5.2808352e-02 -8.7671363e-01]
 [-3.8710088e-02  2.2398938e-01  1.2208397e-02 ... -2.7093867e-02
   6.0223047e-02 -8.7671363e-01]]

----------for test data----------
Precesion: 0.6647
Recall: 0.4883
F1-score: 0.5630
Accuracy: 0.7814
AUC: 0.8246
测试集每个样本的得分
 [0.15013078 0.12705094 0.24904178 ... 0.04131015 0.01378498 0.17598328]
测试集每棵树所属的节点数
 [[31 31 31 ... 35 47 31]
 [46 40 49 ... 35 48 39]
 [59 39 49 ... 35 48 26]
 ...
 [62 44 48 ... 38 28 32]
 [62 44 48 ... 21 46 38]
 [62 44 48 ... 35 44 32]]
特征的重要性
 [[ 0.00650658 -0.01686223  0.01750353 ...  0.04526495  0.04253626
  -0.87671363]
 [ 0.02561823 -0.23103392 -0.0753253  ...  0.04120616  0.08915053
  -0.87671363]
 [ 0.00883511 -0.14391811 -0.10248443 ...  0.04065876  0.0551112
  -0.87671363]
 ...
 [ 0.04139393 -0.28045234 -0.05307652 ... -0.01351984 -0.00323758
  -0.87671363]
 [-0.02023524 -0.3328192  -0.09465612 ... -0.02059004 -0.01608757
  -0.87671363]
 [ 0.04860631 -0.3361546   0.00627191 ... -0.00696666  0.06233648
  -0.87671363]]
```



五、 附录（实验代码）

```
from numpy import loadtxt
from sklearn import metrics
from sklearn.model_selection import train_test_split
import xgboost as xgb
import matplotlib.pyplot as plt
# load data
train_data = loadtxt('train.csv', delimiter=',', skiprows=1)
train_X = train_data[1:,0:162]
train_Y = train_data[1:,162]
validation_data = loadtxt('validation.csv', delimiter=',', skiprows=1)
validation_X = validation_data[1:,0:162]
validation_Y = validation_data[1:,162]
test_data = loadtxt('test.csv', delimiter=',', skiprows=1)
test_X = test_data[1:,0:162]
test_Y = test_data[1:,162]

```

```
# xgboost 模型初始化设置
dtrain = xgb.DMatrix(train_X, label=train_Y)
dvalid = xgb.DMatrix(validation_X)
dtest = xgb.DMatrix(test_X)
watchlist = [(dtrain, 'train')]

# booster:
params = {    'booster': 'gbtree',
                    'objective': 'binary:logistic',
                    'eval_metric': 'auc',
                    'max_depth': 5,
                    'lambda': 10,
                    'subsample': 0.75,
                    'colsample_bytree': 0.75,
                    'min_child_weight': 2,
                    'eta': 0.025,
                    'seed': 0,
                    'nthread': 8,
                    'gamma': 0.15,
                    'learning_rate' : 0.1    }

#  建模与预测：100 棵树
bst = xgb.train(params, dtrain, num_boost_round=100, evals=watchlist)

print("----------for validation data----------")
ypred = bst.predict(dvalid)
#  设置阈值、评价指标
y_pred = (ypred >= 0.5)*1
print ('Precesion: %.4f' %metrics.precision_score(validation_Y,y_pred))
print ('Recall: %.4f' % metrics.recall_score(validation_Y,y_pred))
print ('F1-score: %.4f' %metrics.f1_score(validation_Y,y_pred))
print ('Accuracy: %.4f' % metrics.accuracy_score(validation_Y,y_pred))
print ('AUC: %.4f' % metrics.roc_auc_score(validation_Y,ypred))

ypred = bst.predict(dvalid)
print("验证集每个样本的得分\n",ypred)
ypred_leaf = bst.predict(dvalid, pred_leaf=True)
print("验证集每棵树所属的节点数\n",ypred_leaf)
ypred_contribs = bst.predict(dvalid, pred_contribs=True)
print("特征的重要性\n",ypred_contribs )


print("----------for test data----------")
ypred = bst.predict(dtest)
```

```
# 设置阈值、评价指标
y_pred = (ypred >= 0.5)*1
print ('Precesion: %.4f' %metrics.precision_score(test_Y,y_pred))
print ('Recall: %.4f' % metrics.recall_score(test_Y,y_pred))
print ('F1-score: %.4f' %metrics.f1_score(test_Y,y_pred))
print ('Accuracy: %.4f' % metrics.accuracy_score(test_Y,y_pred))
print ('AUC: %.4f' % metrics.roc_auc_score(test_Y,ypred))

ypred = bst.predict(dtest)
print("测试集每个样本的得分\n",ypred)
ypred_leaf = bst.predict(dtest, pred_leaf=True)
print("测试集每棵树所属的节点数\n",ypred_leaf)
ypred_contribs = bst.predict(dtest, pred_contribs=True)
print("特征的重要性\n",ypred_contribs )

# 参数 importance_type(string, default “gain”), 其他取值有“weight”, “cover”, “total_gain” or “total_cover”.
# weight:该特征被选为分裂特征的次数;
# gain: 该特征在它所有分裂使用中带来的平均增益;
# total_gain: 该特征在它所有分裂使用中带来的增益和;
# cover: 该特征在它所有分裂使用中的平均覆盖率(覆盖率:受分裂影响的样本数);
# total_cover: 该特征在它所有分裂使用中的总覆盖率.

fig,ax = plt.subplots(figsize=(10,10))
xgb.plot_importance(bst, ax=ax, height=0.5, max_num_features=20, title='Important Features', ylabel='Features')
plt.rc('font', family='Arial Unicode MS', size=14)
plt.show()
```