**学科实践二**

**实验报告**

**题 目：**

**驱动赛道A题：MarTech Challenge 点击反欺诈预测&&基于ResNet101的体育运动100类分类**

**专 业： 计算机科学与技术**

**姓 名：邓语苏 汤慧婷**

**学 号1： 22920212204066**

**学 号2： 20420212201858**

**日 期**：**2023年6月21日**

# （一）驱动赛道A题：点击反欺诈预测

1. **实验目的**

在本题目中提供了训练集train.csv文件以及测试集test1.csv文件，数据集中提供了会话sid以及基于会话的各维度的特征值。基于训练集得出的模型进行预测，判断该会话sid是否为作弊行为。

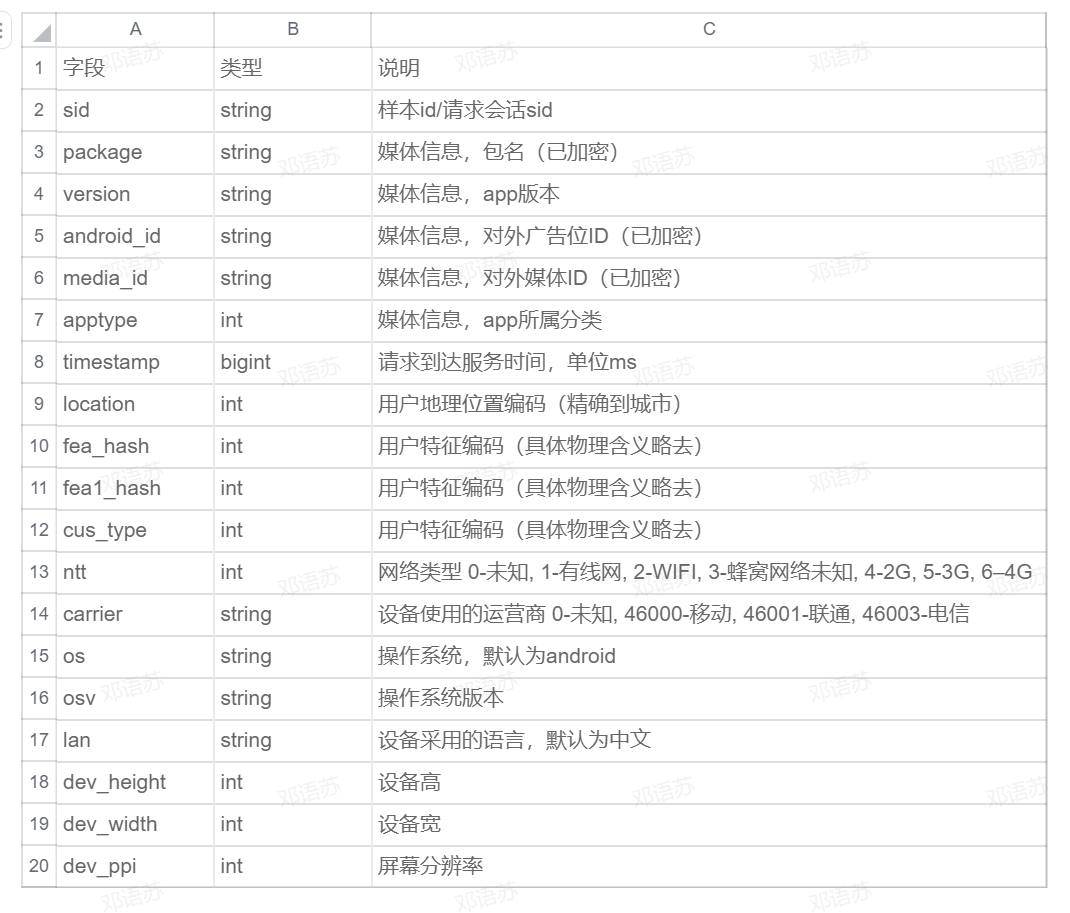
1. **实验内容**

分析原始数据集，选择合适的模型，按照模型要求进行特征工程（包括数据清洗、数据增强）。将处理后的训练集放入基模型做Stacking训练，将Stacking得到的新列加入训练集，将新训练集放入元模型训练。将三个元模型(catboost、xgboost、lightgbm)得到的结果使用硬投票的模型融合结果。

**三．实验步骤以及结果**

1. **数据描述：**

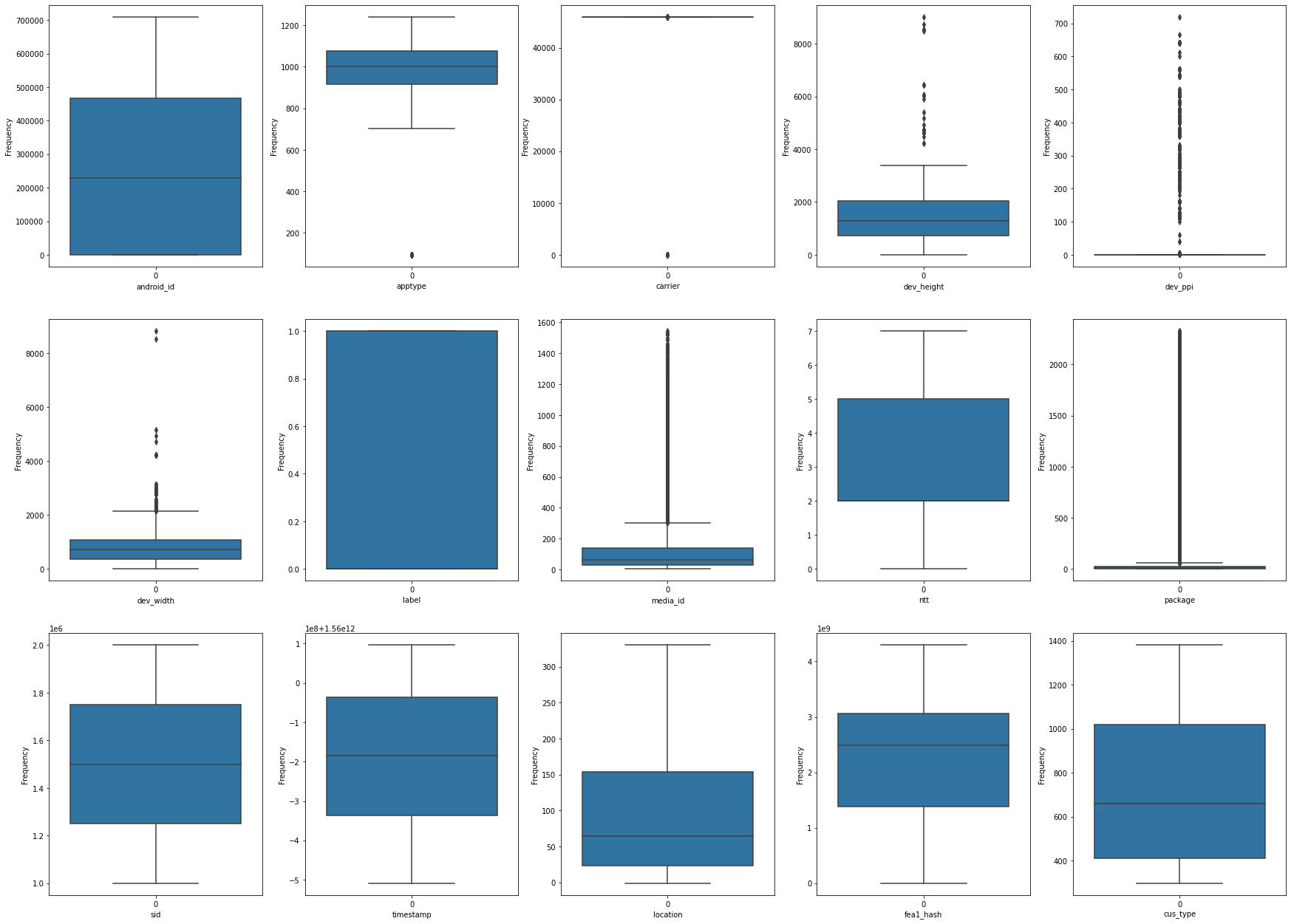
训练集数据为50万条，测试集数据为15万条，数据较大数据量的数据集。



我们可以利用python自带的库函数来进行一些初步的数据分析，分析结果如下

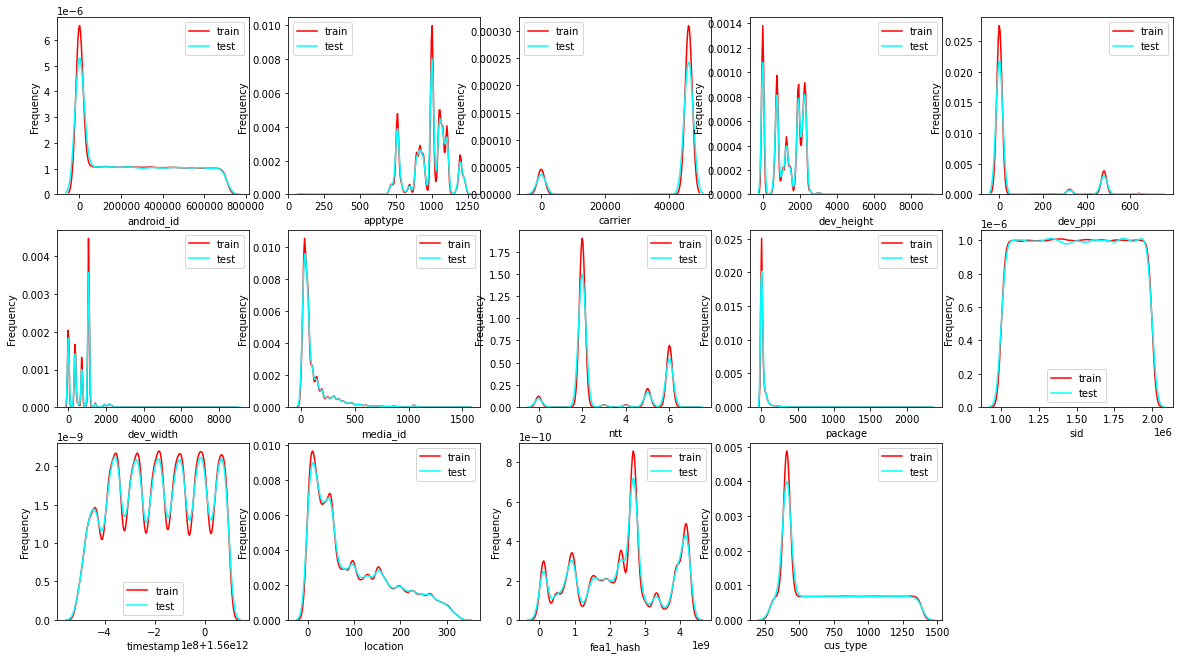
#### 对数值型变量绘制箱线图

观察图表可以看出carrier的取值比较集中，而dev\_ppi的取值偏于离散，若对label值影响不大可以考虑删除。dev\_height和dev\_width中包含了一定的异常值，需要进行一定的数据预处理。

****

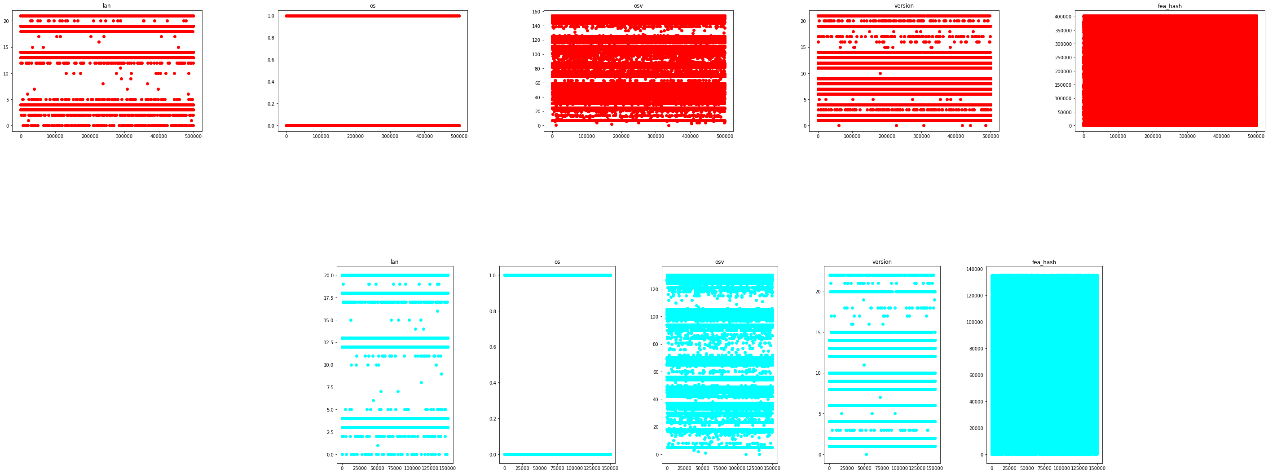
#### 查看训练集与测试集数值变量分布

可以看出训练集与测试集数值变量分布大致一致

****

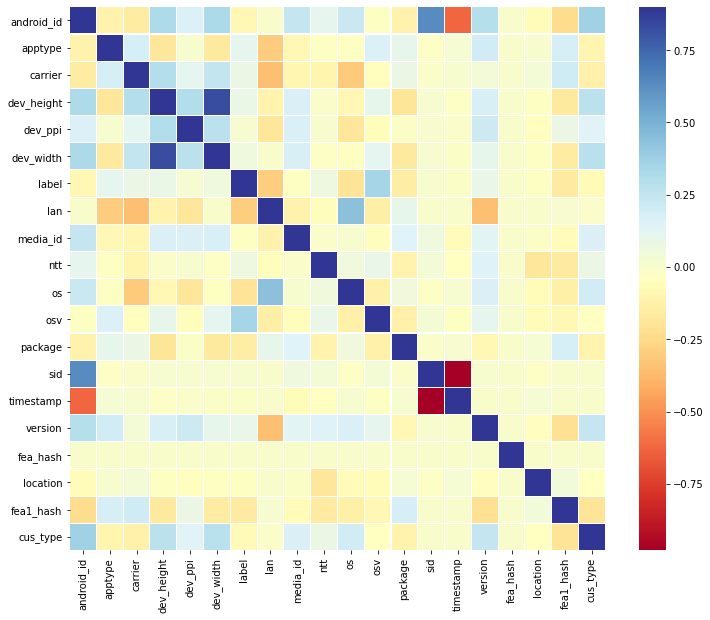
#### 查看分类变量的分布

可以看出osv取值单一，对label值的预测没有贡献，可以删除。fea\_hash多为唯一值。

****

#### 绘制绘制协方差相关矩阵热图

观察协方差相关矩阵热图，可作出初步判断

****

1. **特征工程**

* 新构造特征列

1. # 长度特征
2. all\_df['fea1\_hash\_len'] = all\_df['fea1\_hash'].map(**lambda** x:len(str(x)))
3. all\_df['fea\_hash\_len'] = all\_df['fea\_hash'].map(**lambda** x:len(str(x)))
5. # 设别高宽比、设备面积
6. all\_df['dev\_rate'] = all\_df['dev\_height']/all\_df['dev\_width']
7. all\_df['dev\_area'] = all\_df['dev\_height']\*all\_df['dev\_width']
9. # 面积与分辨率比
10. all\_df['dev\_area\_ppi'] = all\_df['dev\_ppi']/all\_df['dev\_area']
11. all\_df['dev\_area\_ppi'][np.isinf(all\_df['dev\_area\_ppi'])] = 0

* 构造数量特征(unique)

1. # 利用数量特征构造。
2. **def** unique(x):
3. result = pd.value\_counts(x)
4. x= [result[each] **for** each **in** x]
5. **return** x
7. **for** f **in** ['dev\_height','dev\_width','media\_id','package','apptype',
8. 'android\_id','fea1\_hash','fea\_hash']:
9. all\_df[f+'\_value\_count']=unique(all\_df[f])

* 对类别变量编码

1. le = LabelEncoder()
2. all\_df['lan']= all\_df['lan'].astype('str')
3. all\_df['lan'] = le.fit\_transform(all\_df['lan'])

* 处理时间戳

1. # 时间类特征
2. **from** datetime **import** datetime
3. train['timestamp'] = train['timestamp'].apply(**lambda** x : datetime.fromtimestamp(x/1000))
4. test['timestamp'] = test['timestamp'].apply(**lambda** x : datetime.fromtimestamp(x/1000))
5. # train和test都是从2019-06-03凌晨到2019-06-10凌晨
7. # 分解时间
8. train['day'] = train['timestamp'].dt.day
9. train['weekday'] = train['timestamp'].dt.weekday
10. train['hour'] = train['timestamp'].dt.hour
11. train['minute'] = train['timestamp'].dt.minute
13. test['day'] = test['timestamp'].dt.day
14. test['weekday'] = test['timestamp'].dt.weekday
15. test['hour'] = test['timestamp'].dt.hour
16. test['minute'] = test['timestamp'].dt.minute
18. start\_time = train['timestamp'].min()
19. train['timestamp\_diff'] = train['timestamp']-start\_time
20. train['timestamp\_diff'] = train['timestamp\_diff'].dt.days\*24 + train['timestamp\_diff'].dt.seconds/3600 # 按小时来计算
21. test['timestamp\_diff'] = test['timestamp']-start\_time
22. test['timestamp\_diff'] = test['timestamp\_diff'].dt.days\*24 + test['timestamp\_diff'].dt.seconds/3600 # 按小时来计算
24. # 对时间差做分桶
25. test['timestamp\_diff'] = test['timestamp\_diff'].apply(**lambda** x: int(x//13))
26. train['timestamp\_diff'] = train['timestamp\_diff'].apply(**lambda** x: int(x//13))
28. train.drop('timestamp', axis=1,inplace=True)
29. test.drop('timestamp', axis=1,inplace=True)

* 特征清洗

1. #特征清洗version
2. **def** rep(x):
3. **if** str(x).isdigit():
4. **return** int(x)
5. **elif** str(x)[0] == "v" **or** "V":
6. **if** str(x)[1:].isdigit():
7. **return** int(str(x)[1:])
8. **else**:
9. **return** 0
10. **else**:
11. **return** 0
13. train['version'] = train['version'].apply(rep)
14. test['version'] = test['version'].apply(rep)
15. **算法结构**

* **特征工程**

1. 数据清洗、数据增强

* **模型建立**

将5个基模型作为**Stacking**第一层，将Stacking得到的新列加入train集和test1集，把更新后的数据集放入3个元模型进行训练，并将得到的预测值进行**投票表决**

1. 基于五折交叉验证的基分类器的训练
2. 元分类器的训练
3. 元分类器预测结果的投票表决
4. **模型建立与训练**

### （一）基于五折交叉验证的基分类器训练

1. # 构建基分类器
2. **from** sklearn.ensemble **import** RandomForestClassifier
3. **from** sklearn.ensemble **import** ExtraTreesClassifier
4. **from** sklearn.ensemble **import** AdaBoostClassifier
5. **from** sklearn.ensemble **import** GradientBoostingClassifier
6. **from** sklearn.neural\_network **import** MLPClassifier
7. **from** sklearn.model\_selection **import** train\_test\_split
8. **from** sklearn.model\_selection **import** StratifiedKFold
10. #定义基分类器
11. clf = [RandomForestClassifier(),
12. ExtraTreesClassifier(),
13. AdaBoostClassifier(),
14. GradientBoostingClassifier(),
15. MLPClassifier(hidden\_layer\_sizes=(15))]
16. name = ['randomforest','extratrees','gradientboost','adaboost','mlp']
18. X\_train=pd.DataFrame(X)
19. label=pd.DataFrame(y)
21. # 五折交叉验证，不随机选取
22. skf = StratifiedKFold(n\_splits=5,shuffle=False)
24. # 基分类器训练和预测
25. **for** i,j **in** zip(clf,name):
27. yhat = pd.DataFrame(columns=['label','sid']) # 当前基分类器的train集完整的预测结果
29. #五折交叉验证
30. **for** index,(train\_index,test\_index) **in** enumerate(skf.split(X\_train, label)):
32. pred =pd.DataFrame(columns=['label','sid'])  # 当前基分类器第index折的test1集预测结果
34. train\_x,test\_x,train\_y,test\_y=X\_train.iloc[train\_index],X\_train.iloc[test\_index],label.iloc[train\_index],label.iloc[test\_index]
35. i.fit(train\_x,train\_y)
37. new\_row = pd.DataFrame({'label': i.predict(test\_x)})# 当前基分类器第index折的test\_x集的预测结果
38. yhat=yhat.append(new\_row)#拼接上已有的预测结果
39. **print**(len(yhat['label']))
41. pred['label'] = pred['label'].append(pd.Series(i.predict(x\_pred)), ignore\_index=True)
42. pred['sid']=test['sid']
43. pred.to\_csv("test\_"+str(index)+"\_"+j+".csv", index=False)
45. yhat['sid']=train['sid']
46. yhat.to\_csv("train\_"+j+".csv",index=False)#当前基分类器对train集完整的预测结果

文件处理：将已有的文件处理为可用于stacking的形式

1. **import** pandas as pd
2. **import** numpy as np
3. **import** os
4. name = ['randomforest','extratrees','gradientboost','adaboost','mlp','label']
5. train=pd.read\_csv("train.csv")
6. train\_new=pd.DataFrame(columns=['label'])
7. train\_new['label']=train['label']
9. root="/home/aistudio/"
10. listdir=os.listdir(root)
11. **for** file **in** listdir:
12. # print(file)
13. **for** n **in** name:
14. **if** "train\_"+n **in** file:
15. file\_root=os.path.join(root,file)
16. **print**(file\_root)
17. df=pd.read\_csv(file\_root)
18. **print**(train\_new)
19. train\_new[n]=df['label']
21. train\_new.to\_csv("train\_new.csv")
23. **for** n **in** name:
24. test\_new = pd.DataFrame()
25. **for** file **in** listdir:
26. #取平均
27. **for** i **in** range(5):
28. **if** str(i) + "\_" + n **in** file:
29. file\_root = os.path.join(root, file)
30. **print**(file)
31. df = pd.read\_csv(file\_root)
32. test\_new[str(i)] = df['label']
33. # print(test\_new)
34. test\_new[n+"\_average"] = test\_new.mean(axis=1)
35. test\_new.to\_csv("test\_" + n + ".csv", index=False,encoding="gbk")
37. listdir=os.listdir(root)
38. **for** file **in** listdir:
39. # print(file)
40. **for** n **in** name:
41. **if** "test\_"+n **in** file:
42. file\_root=os.path.join(root,file)
43. **print**(file\_root)
44. df=pd.read\_csv(file\_root)
45. **print**(test\_new)
46. test\_new[n]=df[n+'\_average']
47. test\_new[n] = test\_new[n].apply(**lambda** x: 0 **if** x < 0.5 **else** 1)
48. test\_new.to\_csv("test\_new.csv")

### （二）元分类器的训练

1. xgboost&&lightgbm

1. **import** pandas as pd
2. name = ['randomforest','extratrees','gradientboost','adaboost','mlp']
3. test\_new=pd.read\_csv("test\_new.csv")
4. train\_new=pd.read\_csv("train\_new.csv")
5. X=pd.DataFrame(X)
6. X\_test=pd.DataFrame(X\_test)
8. #将test\_new、train\_new拼接到原数据集上
9. X=pd.concat([X,train\_new[name]],axis=1)
10. X\_test=pd.concat([X\_test,test\_new[name]],axis=1)
12. # LGB XGB 5折交叉验证
13. **from** lightgbm **import** LGBMClassifier
14. **from** xgboost **import** XGBClassifier
15. **def** lgb\_xgb\_train(KF,X,y,X\_test):
16. seed = 2023
17. lgbcls = LGBMClassifier(num\_leaves=1024,
18. max\_depth=12,
19. learning\_rate=0.005,
20. n\_estimators=5000,
21. subsample=0.8,
22. feature\_fraction=0.8,
23. reg\_alpha=0.5,
24. reg\_lambda=0.5,
25. random\_state=seed,
26. metric='auc',
27. boosting\_type='gbdt',
28. subsample\_freq=1,
29. bagging\_fraction=0.8,verbose=-1)
31. xgbcls=XGBClassifier(
32. max\_depth=13, learning\_rate=0.005, n\_estimators=2400,
33. objective='binary:logistic',
34. subsample=0.95, colsample\_bytree=0.4,n\_jobs=-1,random\_state=seed,
35. min\_child\_samples=3, eval\_metric='auc', reg\_lambda=0.5,verbosity=0)
37. xgb\_prob = np.zeros(X\_test.shape[0])
38. lgb\_prob = np.zeros(X\_test.shape[0])
39. skf = StratifiedKFold(n\_splits=KF, shuffle=True, random\_state=seed)
41. **for** k,(train\_index, test\_index) **in** enumerate(skf.split(X, y)):
42. X\_train, X\_val = X.iloc[train\_index], X.iloc[test\_index]
43. y\_train, y\_val = y.iloc[train\_index], y.iloc[test\_index]
44. **print**(f'第{k+1}次训练')
46. # 训练
47. xgbs = xgbcls.fit(X\_train,y\_train,
48. eval\_set=[(X\_val,y\_val)],
49. verbose=10)
50. lgbs = lgbcls.fit(X\_train,y\_train,
51. eval\_set=[(X\_val, y\_val)],
52. verbose = 10)
54. # 正式预测
55. xgb\_prob += xgbs.predict\_proba(X\_test)[:,1]/KF
56. **print**(f'Xgboost:第{k+1}次训练的Accuracy{accuracy\_score(y\_val,xgbs.predict(X\_val))}')
57. lgb\_prob += lgbs.predict\_proba(X\_test)[:,1]/KF
58. **print**(f'LightGBM:第{k+1}次训练的Accuracy{accuracy\_score(y\_val,lgbs.predict(X\_val))}')
60. **return** xgb\_prob,lgb\_prob

2. catboost

1. #catboost建模
2. cate\_ = [x **for** x **in** cate\_cols **if** x **in** f\_cols]
3. train[cate\_] = train[cate\_].astype(str)
4. **print**('变量数',len(f\_cols),'类别变量数',len(cate\_))
5. gc.collect()
6. **print**(train.shape)
7. **from** sklearn.model\_selection **import** RandomizedSearchCV
8. **from** sklearn.metrics **import** accuracy\_score
9. **from** sklearn.metrics **import** make\_scorer
10. params = {
11. #  'depth': [4, 6, 10],
12. 'learning\_rate' : [0.07,0.09,0.1],
13. 'l2\_leaf\_reg':[1,4,9],
14. 'iterations':[2000,3000,5000],
15. 'task\_type':['GPU'],
16. 'max\_depth':[4,6,8],
17. 'loss\_function':['Logloss'],
18. 'gpu\_ram\_part':[0.9],
19. }
20. estimator=cb.CatBoostClassifier(verbose=30,eval\_metric='Accuracy',use\_best\_model=True)
21. #进行随机搜索最优参数
22. cbt\_model=RandomizedSearchCV(estimator, params, scoring="accuracy", cv =5)
24. cbt\_model.fit(train[f\_cols], train['label'],
25. eval\_set=(train[f\_cols], train['label']),
26. cat\_features=cate\_)
27. gc.collect()
29. #test文件预测
30. test\_cat = pd.read\_hdf('./data/test\_hdf.h5');
31. gc.collect()
33. test\_cat[cate\_] = test\_cat[cate\_].astype(str)
34. test\_cat['label'] = cbt\_model.predict(test\_cat[f\_cols])
36. sub = pd.read\_csv('test1.csv')
37. sub = pd.merge(sub,test\_cat[['sid','label']],on='sid')
38. sub\_ = sub[['sid','label']]
39. sub\_.to\_csv('catboost.csv',index=False)

**四．实验结果与分析**

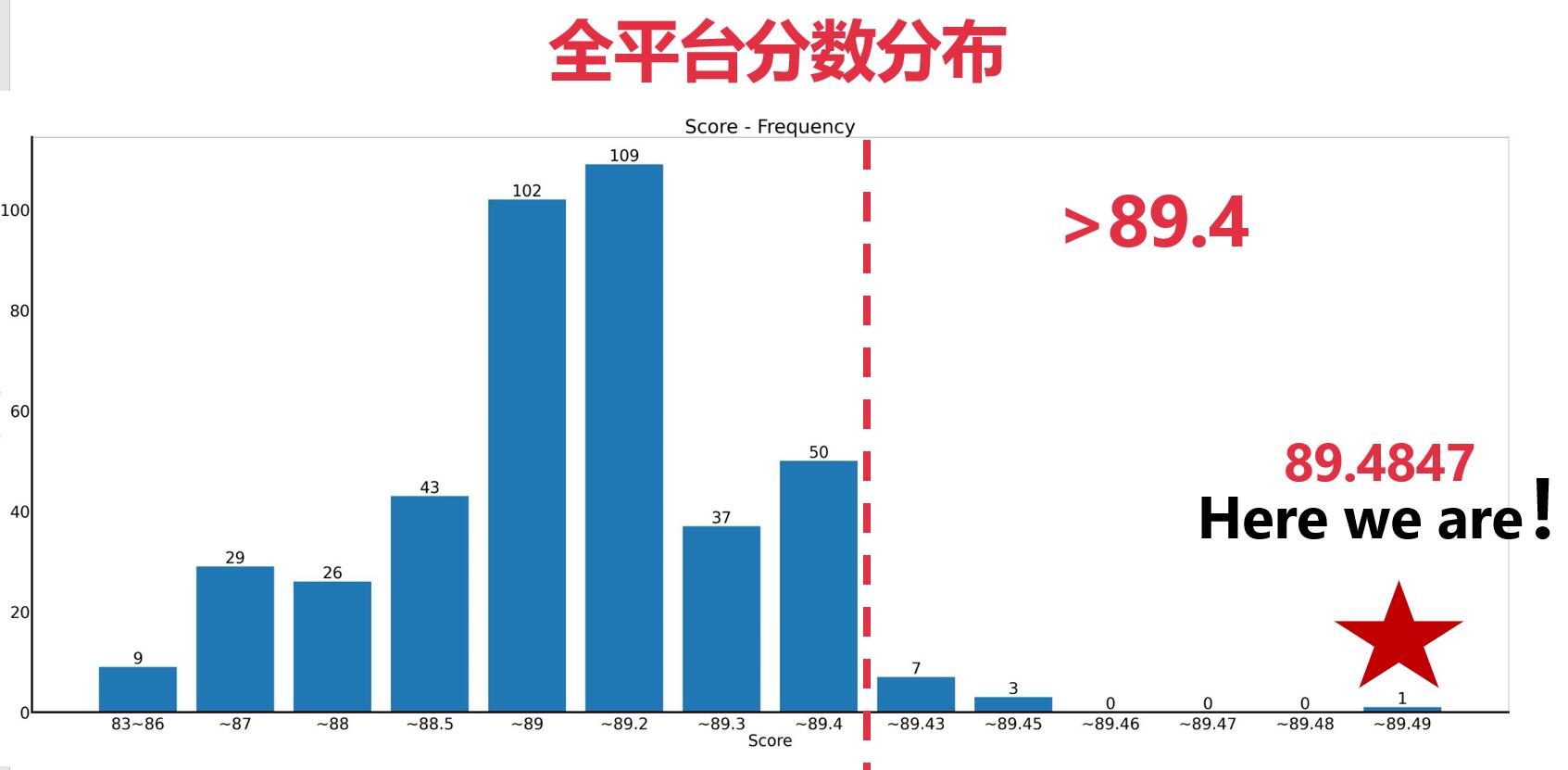
### 1. ROC曲线

### 2. 准确率

**基于Stacking改进Voting算法在本比赛实现结果为：89.4847**

 \* 当前结果为厦门大学第二届人工智能创新驱动赛**-驱动赛A题最高分**

 \* 超过所有同名比赛的最高准确率，现为**全平台最高分**

****

**五．实验总结**

#### 优点：

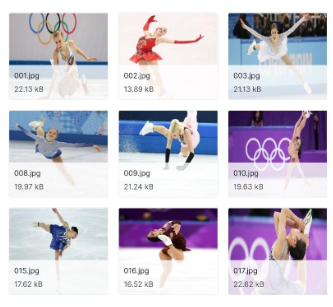
1. 所选用的三种模型CatBoost、XGBoost和LightGBM都是基于梯度提升决策树（GBDT）算法的框架，GBDT 具有训练效果好、不易过拟合等优点。GBDT在实际应用中被广泛使用，特别是在分类和回归问题中。它在处理结构化数据和非线性关系方面表现优秀，并且对特征工程的要求相对较低。
2. 选用三个框架都是GBDT算法的优秀实现，它们在模型性能、速度和功能方面都有不同的优势和特点。其中CatBoost有很好的处理大量类别特征的能力，避免了用独热编码可能造成的维度灾难和大量计算资源的消耗，更适用于本题。
3. 使用了Stacking集成模型，通过将多个独立模型的预测结果进行结合，可以融合它们的优势，抵消个别模型的弱点，增加模型的鲁棒性，提高了泛化能力，从而提高整体模型的性能。

#### 缺点：

1. Stacking训练了5个基础模型和3个元模型，增加了计算复杂性和训练时间。
2. 模型可解释性较差
   1. **基于ResNet101的体育运动100类分类**
3. **实验目的**

随着我国体育产业的蓬勃发展，体育图像数据量呈指数增长，对体育图像进行有效的分类可以方便用户快速检索和访问、加速体育产业的智能化发展、对观看体育赛事的大众人员进行体育知识的相关普及。

1. **数据集介绍**

****本项目使用palldle平台上名为“100类体育运动分类数据集”的数据集。共包含100个类，以paddle平台的一个数据集为基准，在设置的100个运动分类中进行相应数量的数据扩充与数据增强。训练集数据为13572张，测试集数据为500张。

1. **实验内容**

将图片进行数据增强后放入resnet101\_vd\_ssld预训练模型进行训练

1. **实验步骤以及结果**

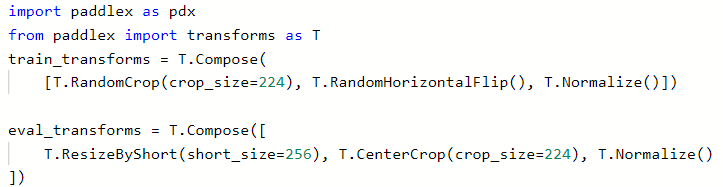
* 数据增强

**利用transforms模块进行数据增强和预处理**

1.进行随机裁剪操作，将图像随机裁剪为指定的尺寸

2.随机水平翻转，以一定的概率对图像进行水平翻转

3.图像归一化



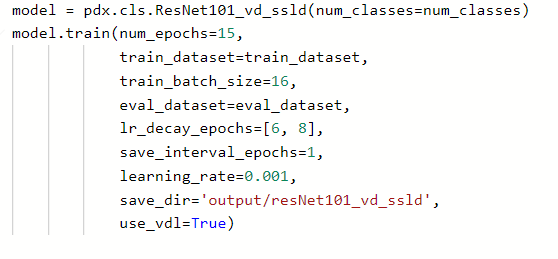
* 模型选择

选择PaddleX框架下的ResNet101模型。 ResNet101是由微软研究院于2015年提出的卷积神经网络架构。它是ResNet（残差网络）模型的一种变体，以其包含101个层的深度而得名。ResNet101在图像分类、目标检测和语义分割等计算机视觉任务中取得了很好的性能，并成为深度学习领域的经典模型之一。

ResNet101的具体结构如下：

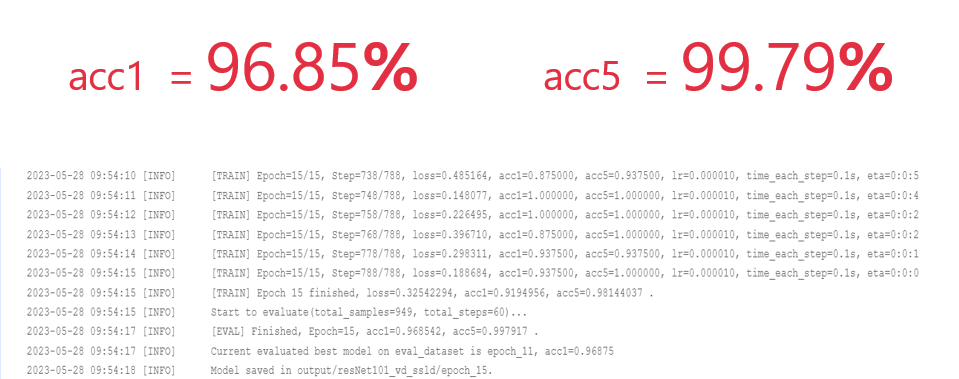
1. 输入层：接受输入图像或特征图。
2. 卷积层和池化层：通过一系列的卷积层和池化层进行特征提取和下采样，逐渐减小特征图的尺寸。
3. 残差块：由多个残差块组成，每个残差块都包含两个或更多卷积层。
4. 全局平均池化层：对最后一个残差块的输出特征图进行全局平均池化，将特征图的尺寸减小到1x1。
5. 全连接层和分类器：使用一个全连接层和softmax分类器将特征映射到不同的类别，并生成最终的预测结果。

* 模型训练



1. **实验结果与分析**

训练20轮，在第15轮达到最优，Acc1=96.85%



1. **实验总结**

#### 优点：

1. 解决梯度消失问题：引入残差连接允许梯度直接传播到浅层，使得可以训练非常深的网络，避免了传统深层网络中的梯度消失问题。
2. 提高特征表达能力：残差块的设计允许网络更好地学习到输入和输出之间的残差，从而提高特征的表达能力，使得模型更易于学习和捕捉复杂的特征。
3. 减少参数量：相比于传统的深层网络，ResNet101使用了残差连接，可以减少模型的参数量，降低了过拟合的风险，同时减少了模型的计算复杂性。
4. 较好的模型泛化能力：ResNet101在许多计算机视觉任务中取得了优秀的性能，具有较好的模型泛化能力，能够对不同类型的数据集进行有效的学习和预测。

#### 缺点：

1. 需要更多的计算资源：由于ResNet101是一个非常深的网络，相对于浅层网络，它需要更多的计算资源和较长的训练时间，特别是在较低的硬件配置下。
2. 容易过拟合：虽然残差连接可以减少过拟合的风险，但当网络过于深时，仍然可能面临过拟合问题。