案例名称：国际足球比赛结果数据分析及预测

摘要

随着我国大数据时代的来临，数据的存在有着十分重要的意义。在当今这个信息大爆炸的时代，对数据进行收集和分析进而获得对我们有益的数据变得尤为重要。故此次研究的目的就是结合实际案例，对所学习的数据分析课程进行实践，通过对大量数据进行统计、分析、解释和挖掘，进而推动现实问题的决策和价值的实现。

首先对所收集的大量数据进行统计性的分析，得到其重要的特征属性等信息。进而，对数据进行整理、加工、分析和转化，从而获得更为有效的数据内容。接下来对具体的问题进行分析并选用适合的模型对数据进行处理，从而得到我们需要的结果。最后，对整个模型进行评估并优化，并将此类模型进行推广应用。在此案例中，我们对一给定的数据集进行分析和预测，其中数据集包括从 1972 年的第一场正式比赛开始到 2018 年的 39,669 个国际男子足球比赛结果。比赛从 FIFA 世界杯到 FIFI Wild Cup 到常规友谊赛。比赛严格来说是男子全部国际比赛，数据不包括奥运会或其中至少有一支球队是国家队的 B 队，U-23 或联盟精选球队的比赛。

其主要应用的模型有：K-means 聚类模型、决策树模型、xgboost 模型等。研究的问题包括哪些球队热衷于友谊比赛，并讨论能否对比赛成绩进行预测，并最终给出分析的可视化结果。

第一章 案例分析

1.1 分析步骤与流程

（1）读取所给数据
（2）对数据进行数据探索分析与预处理，包括转换数据类型、查看数据样本是否均衡、相关度检测、数据标准化等操作
（3）使用 K-Means 算法进行聚类分析，得出结论：1 类球队最热衷参加友谊赛
（4）使用决策树模型进行预测，得到混淆矩阵和各类评价指标
（5）使用 xgboost 模型进行预测，得到混淆矩阵和各类评价指标
（6）综合两个预测模型，得出结论：可以进行比赛成绩预测

1.2 数据探索分析

足球比赛结果的数据属性说明：

序号	字段名	数据类型	字段描述
1	date	String	日期
2	home_team	String	主队队名
3	away_team	String	客队队名
4	home_score	Integer	主队分数
5	away_score	Integer	客队分数
6	tournament	String	赛事名
7	city	String	城市
8	country	String	国家
9	neutral	String	是否在中立场地打

1.2.1 描述性统计分析

统计 home_team 和 away_team 中每个队伍参赛的次数，并分别以表格形式输出，再将两个表格合并。

代码：数据探索

```
# 读取数据
inputfile = '../dataa/shuju.csv' # 输入的数据文件
data = pd.read_csv(inputfile) # 读取数据

################ 哪些球队最热衷于参加友谊赛(聚类分析) ##########################

# 统计 home_team 每个队伍参加的次数
mid_1 = data["home_team"].value_counts() #对重复元素进行统计
print('home_team 的队伍及参赛次数\n\n',mid_1,'\n\n') #输出队伍和进行比赛的次数

# 统计 away_team 每个队伍参加的次数
mid_2 = data["away_team"].value_counts()
print('away_team 的队伍及参赛次数\n\n',mid_2,'\n\n')

# 两个表格合并
indexs = [] #定义一个数组
for i in mid_2.index.tolist(): #遍历列表（.index.tolist()将索引转换为列表）中所有元素
    if i in mid_1.index.tolist():
        indexs.append(i) #将遍历到的数据存入数组中

data_team = pd.DataFrame([mid_1[indexs].tolist(),mid_2[indexs].tolist()],index = ["home_team","away_team"]).T
data_team.index=mid_1[indexs].index #聚类索引为 mid_1 这一列
print('将两个表格合并\n\n',data_team,'\n\n') #将合并的表输出
```

数据探索分析结果

home_team 的队伍及参赛次数

```
 Brazil                    552
Argentina                  535
Germany                    495
Mexico                     494
England                    483
Sweden                     478
France                     471
Korea Republic             449
Hungary                    445
Italy                      434
                           ...
Artsakh                      1
Crimea                       1
Kabylia                      1
Cascadia                     1
Sark                         1
St. Pierre & Miquelon        1
Romani people                1
Name: home_team, Length: 291, dtype: int64
```

away_team 的队伍及参赛次数

```
 Uruguay                   526
Sweden                     520
England                    499
Hungary                    467
Paraguay                   441
Germany                    438
Argentina                  422
                           ...
Darfur                       3
Western Sahara               2
Romani people                2
Somaliland                   2
Chagos Islands               1
Manchukuo                    1
Name: away_team, Length: 289, dtype: int64
```

将两个表格合并

```
                          home_team  away_team
Uruguay                       339        526
Sweden                        478        520
England                       483        499
```

Hungary	445	467
Paraguay	259	441
...	...	...
Darfur	4	3
Western Sahara	3	2
Romani people	1	2
Somaliland	2	2
Chagos Islands	3	1
Manchukuo	2	1

[286 rows x 2 columns]

1.3 数据预处理

1.3.1 标准化

（1）特点：通过对原始数据进行变换把数据变换到均值为 0 方差为 1 的范围内

（2）公式：

$$X' = \frac{x - mean}{\sigma}$$

注：作用于每一列，mean 为平均值，σ为标准差（考察数据的稳定性）

std 称为方差，

$$\text{std} = \frac{(\text{x1} - \text{mean})^2 + (\text{x2} - \text{mean})^2 + \dots}{\text{n(每个特征的样本数)}}，\ \sigma = \sqrt{\text{std}}$$

（3）对于归一化来说：如果出现异常点，影响了最大值和最小值，那么结果显然会发生改变；对于标准化来说：如果出现异常点，由于具有一定数据量，少量的异常点对于平均值的影响并不大，从而方差改变较小。

代码：标准化数据

```
# 标准化数据
from sklearn.preprocessing import StandardScaler #sklearn 数据特征预处理（标准化）
scaler = StandardScaler()
scaler.fit(data_team) #用于计算训练数据的均值和方差
X_scaled = scaler.transform(data_team) #用得到的均值和方差来转换数据，使其标准化

# 标准化表格存储
Scaled = pd.DataFrame(X_scaled,columns=data_team.columns,index=data_team.index)
# 空值用 0 填充
Scaled = Scaled.fillna(0)
# 查看标准化结果
print('标准化后的数据\n\n',Scaled,'\n\n')
```

标准化结果

标准化后的数据

	home_team	away_team
Uruguay	1.524929	3.125106
Sweden	2.582867	3.076708
England	2.620923	2.907314
Hungary	2.331702	2.649189
Paraguay	0.916043	2.439463
Germany	2.712256	2.415264
Argentina	3.016698	2.286201
Poland	1.798927	2.270069
Finland	1.387929	2.149073
...	...	...
Western Sahara	-1.032391	-1.101683
Romani people	-1.047613	-1.101683
Somaliland	-1.040002	-1.101683
Chagos Islands	-1.032391	-1.109749
Manchukuo	-1.040002	-1.109749

[286 rows x 2 columns]

1.3.2 转换数据类型

代码：查看数据类型并转换数据类型

```
# 将时间设置为索引
data = data.set_index("date")

# 制作特征，主队是否获胜，获胜为1，否则为0
result = []
for i in range(len(data)):
    if data.iloc[i,2]>data.iloc[i,3]: #iloc 函数是通过行号获取行数据
        result.append(1)
    else:
        result.append(0)
# 加入数据
data["result"] = result

# 将 neutral 类型转换为 str
data["neutral"] = data["neutral"].astype("str")

# 查看数据类型
print(' \n\n',data.info(),'\n\n')

# 将类型为 object 类型的标签编码为数值类型
```

```
from sklearn.preprocessing import LabelEncoder
data.iloc[:,[0,1,4,5,6,7]]                                                    =
data.iloc[:,[0,1,4,5,6,7]].apply(LabelEncoder().fit_transform)
# 将数据类型都转换为 float 类型用于建模
data = data.apply(lambda x:x.astype(float))

# 查看转换之后的数据类型看是否有异常
print('\n\n',data.info(),'\n\n')
```

数据类型

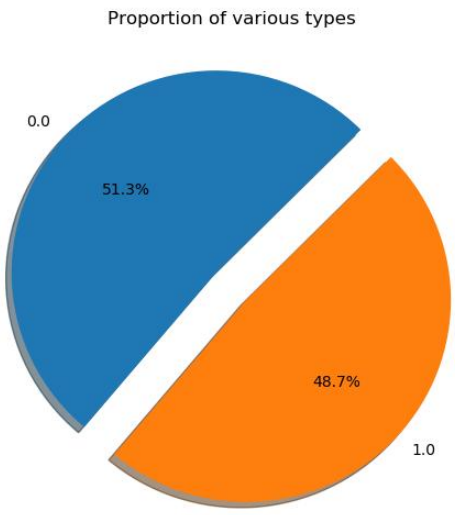
```
<class 'pandas.core.frame.DataFrame'>
Index: 39669 entries, 1872-11-30 to 2018-07-10
Data columns (total 9 columns):
home_team     39669 non-null object
away_team     39669 non-null object
home_score    39669 non-null int64
away_score    39669 non-null int64
tournament    39669 non-null object
city          39669 non-null object
country       39669 non-null object
neutral       39669 non-null object
result        39669 non-null int64
dtypes: int64(3), object(6)
memory usage: 4.3+ MB
 None

<class 'pandas.core.frame.DataFrame'>
Index: 39669 entries, 1872-11-30 to 2018-07-10
Data columns (total 9 columns):
home_team     39669 non-null float64
away_team     39669 non-null float64
home_score    39669 non-null float64
away_score    39669 non-null float64
tournament    39669 non-null float64
city          39669 non-null float64
country       39669 non-null float64
neutral       39669 non-null float64
result        39669 non-null float64
dtypes: float64(9)
memory usage: 4.3+ MB
 None
```

1.3.3 查看数据正负样本是否均衡

代码：查看数据正负样本是否均衡

```
# 查看数据正负样本是否均衡
import matplotlib.pyplot as plt
# 调试图例
plt.figure(figsize=(6,6))
# 设置标签
labels = data["result"].value_counts().index
# 设置扇形间隔
explode=(0.1,0.1)
# 绘制饼图
plt.pie(data["result"].value_counts().tolist(),explode=explode,labels=labels,
autopct='%3.1f%%',#文本标签对应的数值百分比样式
startangle=45,#x 轴起始位置,第一个饼片逆时针旋转的角度
shadow=True)
# 设置标题
plt.title("Proportion of various types")
plt.show()
# 特征，删除标签列和分数
X = data.drop(["away_score","home_score","result"],axis=1)
# 标签
y = data["result"]
```

可视化结果



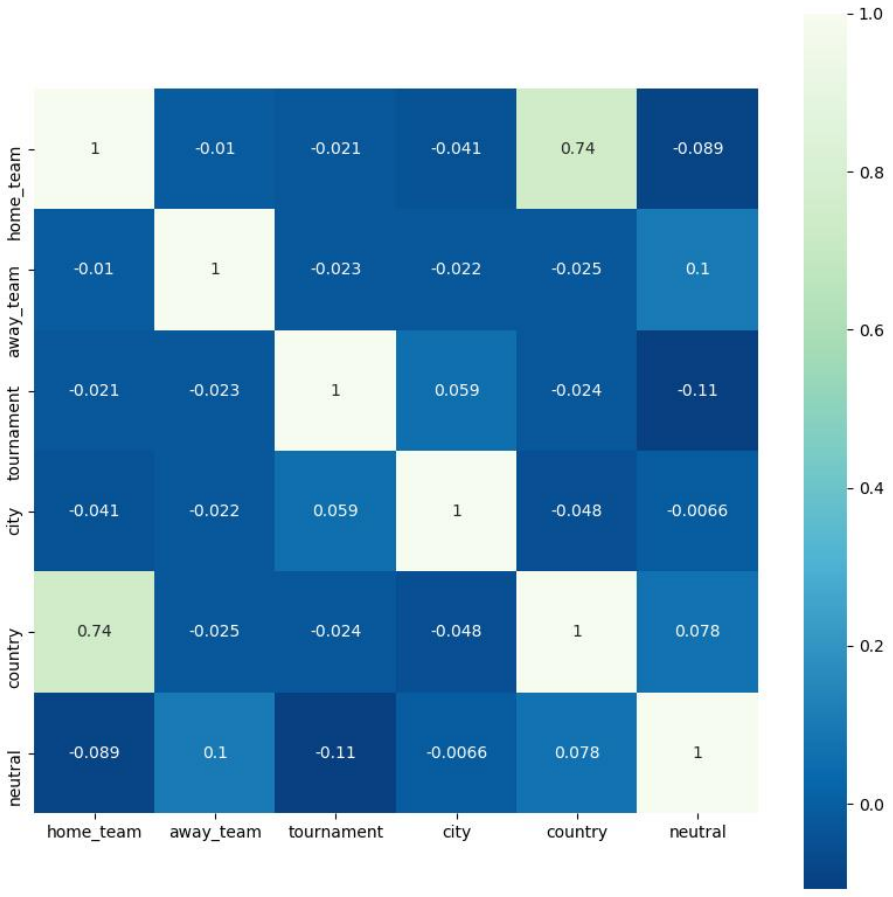
注：从饼图中可以看出，数据的正负样本较为均衡，进而为后面的建模提供了有利基础。

1.3.4 相关性分析

代码：相关性分析

```
# 相关度（特征之间没有较大的相关性不需要 pca 降维）
corr = X.corr()
plt.figure(figsize=(10,10))
# 相关度热力图
sns.heatmap(corr, cmap='GnBu_r', square=True, annot=True)
plt.show()
```

热力图



注：从图中可以看出，特征之间没有较大的相关性不需要 pca 降维。

1.4 模型构建

1.4.1 聚类分析（哪些球队最热衷于友谊赛）

采用 K-Means 聚类算法对各个球队参加友谊赛的数据进行分群，聚成 4 类。

使用 scikit-learn 库下的聚类子库（sklearn-cluster）可以实现 K-Means 聚类算法。使用标准化后的数据进行聚类。

代码：K-Means 聚类

```
# kmeans 聚类训练
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# 下面是用 K-Means 聚类方法做的聚类
mod = KMeans(n_clusters=4,random_state=255) #调用聚类模型 聚类为 4 类
print('构建的 KM 模型为\n\n',mod,'\n\n')
y_pre = mod.fit_predict(Scaled)
print('构建的 KM 模型的结果为\n\n',y_pre,'\n\n')
# 将结果存于表格
data_team["kmeans"] = y_pre

# 可视化查看聚类结果(1 类球队最热衷于参加友谊赛)
plt.figure(figsize=(10,6)) #设定空白画布，并制定大小
# sns.scatterplot()画散点图
# x,y 为数据中变量的名称 作用是生成具有不同颜色的元素的变量进行分组
# hue 根据设置的类别，产生颜色不同的点的散点图 根据 kmeans 进行的分类
sns.scatterplot(x="home_team",y='away_team',data=data_team,hue='kmeans')
# plt.scatter()画散点图
# x,y 是数组，即即将绘制散点图的数据点
# s 是数组的大小，即散点的点的大小
# c 表示颜色    marker 表示标记的样式
plt.scatter(data_team.groupby("kmeans").mean().iloc[:,0],data_team.groupby("kme
ans").mean().iloc[:,1],c="k",marker="*",s=150)
plt.show()

# 这些队伍最爱参加友谊赛
print("这些队伍最爱参加友谊赛：\n")
print(' ',data_team[data_team["kmeans"]==1].index,'  ')
```

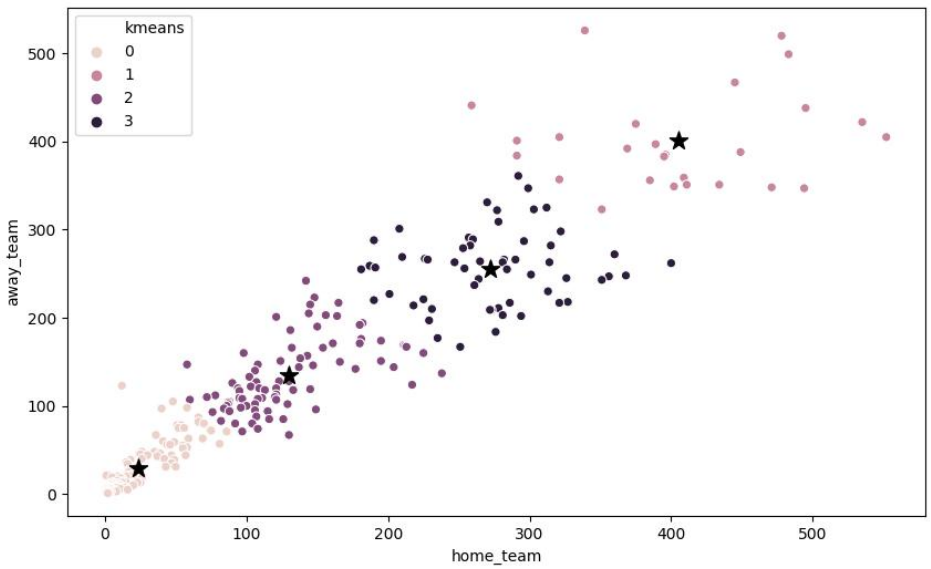
可视化聚类结果

构建的 KM 模型为

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=255, tol=0.0001, verbose=0)
```

构建的 KM 模型的结果为

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 3 1 3 3 3 1 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3 3 2
 3 2 3 3 3 3 3 2 3 2 2 3 2 3 2 2 2 2 3 3 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2
 2 2 2 2 2 0 0 2 2 2 2 2 2 2 0 2 2 2 0 0 2 2 0 0 0 0 2 0 0 0 0 2 2 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```



这些队伍最爱参加友谊赛：

```
Index(['Uruguay', 'Sweden', 'England', 'Hungary', 'Paraguay', 'Germany',
       'Argentina', 'Poland', 'Finland', 'Brazil', 'Zambia', 'Norway',
       'Scotland', 'Korea Republic', 'Switzerland', 'Russia', 'Denmark',
       'Netherlands', 'Romania', 'Chile', 'Austria', 'Italy', 'Belgium',
       'France', 'Mexico', 'Spain'],
      dtype='object')
```

1.4.2. 分类模型（是否可以进行比赛成绩预测）

（1）采用决策树作为评价分类模型。

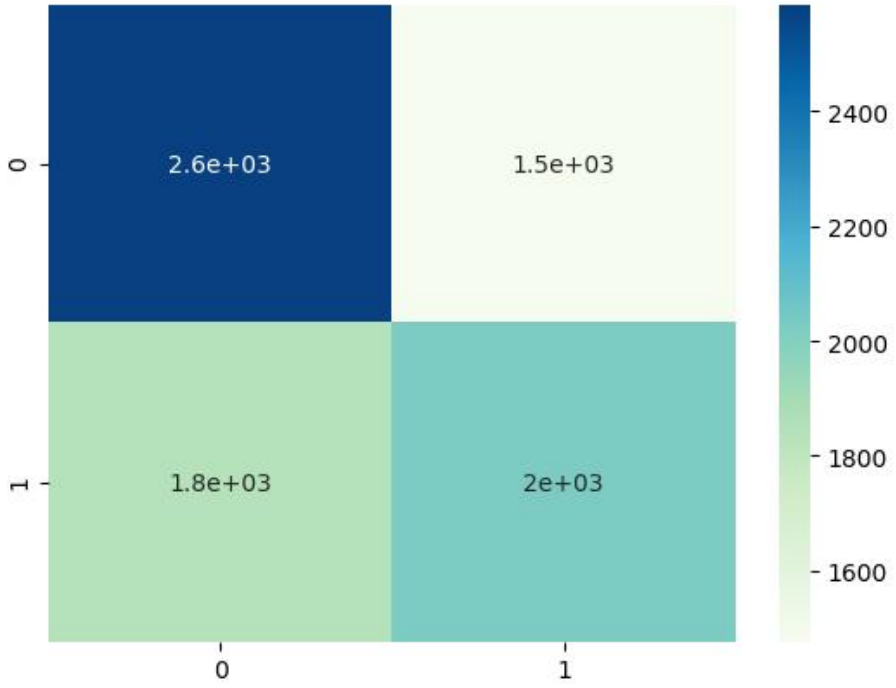
代码：决策树模型

```
# train_test_split 返回划分的训练集 train/测试集 test
from sklearn.model_selection import train_test_split
# 切分数据，0.2 做测试集，0.8 做训练集
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.2,
random_state=255)

from sklearn.model_selection import GridSearchCV #网格搜索
from sklearn.tree import DecisionTreeClassifier
# DecisionTreeClassifier 模型
tree_modle = DecisionTreeClassifier(random_state=255)
# 参数列表
tree_param = {"splitter":["best","random"],
              "max_depth":[5,10,15,20,30,50,80],
              "min_samples_leaf":[30,50,100]}
# 五折网格搜索法
clf = GridSearchCV(tree_modle,tree_param,cv = 5) #cv 交叉验证参数
clf.fit(X_train, y_train) #运行网格搜索
# 打印参数
print("五折网格搜索法\n",clf.best_estimator_,"\n\n") #通过搜索选择的估计器，即在
左侧数据上给出最高分数（或指定的最小损失）的估计器,估计器括号里包括选中的参数。
如果 refit = False，则不可用。

# 决策树模型
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=255) #调用决策树模型
# 训练模型
dt_model.fit(X_train, y_train) #用决策树拟合构造的数据集
# 预测
y_pre = dt_model.predict(X_test) #在训练集和测试集上分别用训练好的模型进行预测

from sklearn import metrics
# 混淆矩阵
# 查看混淆矩阵（预测值和真实值的各类情况统计矩阵）
cm = metrics.confusion_matrix(y_test,y_pre,labels=[0,1])
print('hhhhh\n',cm,'\n\n')
# 利用热力图对于结果进行可视化
sns.heatmap(cm,annot=True,cmap="GnBu")
plt.show()

# 各类评价指标
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pre),'\n\n')
```

混淆矩阵及各类评价指标

五折网格搜索法

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=30,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=30, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=255,
splitter='best')



	precision	recall	f1-score	support
0.0	0.58	0.64	0.61	4061
1.0	0.58	0.52	0.55	3873
avg / total	0.58	0.58	0.58	7934

注：使用 scikit-learn 内置的 GridSearchCV 网格搜索获得最优参数

当使用决策树算法的时候，为了能够更好地拟合和预测，需要调整它的参数。在决策树算法中，通常选择的参数是决策树的最大深度。于是将给出一系列的最大深度的值，使其尽可能包含最优最大深度。

GridSearchCV 是将交叉验证和网格搜索封装在一起的方法，在 GridSearchCV 中有一个属性称为 best_score_，这个属性会存储模型在交叉验证中所得的最高分。

（2）采用 xgboost 模型作为评价分类模型。

*Xgboost算法：*全名为极端梯度提升，应用机器学习领域一个强有力的工具，其基本思想为：一棵树一棵树逐渐地往模型里加，每加一棵 CRAT 决策树时，要使得整体的效果有所提升。使用多棵决策树构成组合分类器，并且给每个叶子节点赋与一定的权值。

*CRAT回归树：*它包括分类树和回归树两种；两者的不同之处是，分类树的样本输出（即响应值）是类的形式；而回归树的样本输出是数值的形式，这时没法用信息增益、信息增益率、基尼系数来判定树的节点分裂了，我们会采用新的方式：预测误差，常用的有均方误差、对数误差等。节点的确定有时用节点内样本均值，有时是最优化算出来的，比如 Xgboost。

简单原理介绍：

模型：模型由 k 棵 CART 树组成，表示为：

$$\widehat{y_i} = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathrm{F}$$

这里的 K 就是树的棵数，F 表示所有可能的 CART 树，f 表示一棵具体的 CART 树

目标函数：

$$\mathrm{obj}(\theta) = \sum_{i}^{n} l(y_i, \widehat{y_i}) + \sum_{k=1}^{K} \Omega(f_k)$$

等式右边第一部分是损失函数，第二部分是正则项，这里的正则化项由 K 棵树的正则化项相加而来

训练 xgboost：

训练目的：最小化目标函数

如果 K 棵树的结构都已经确定，那么整个模型剩下的就是所有 K 棵树的叶子节点的值，模型的正则化项也可以设为各个叶子节点的值的平方和。此时，整个目标函数其实就是一个 K 棵树的所有叶子节点的值的函数，我们就可以使用梯度下降或者随机梯度下降来优化目标函数。

加法训练：

分步骤优化目标函数，首先优化第一棵树，完了之后再优化第二棵树，直至优化完 K 棵树。

模型正则化项

首先对 CART 树作另一番定义：

$$f_t(x) = \omega_{q(x)}, \omega \in R^T, q: R^d \to \{1,2,\ldots,T\}$$

定义解释如下：一棵树有 T 个叶子节点，这 T 个叶子节点的值组成了一个 T 维向量ω，q(x)是一个映射，用来将样本映射成 1 到 T 的某个值，也就是把它分到某个叶子节点，q(x)其实就代表了 CART 树的结构。$\omega_q(x)$就是这棵树对样本 x 的预测值了。

于是 xgboost 使用了如下的正则化项

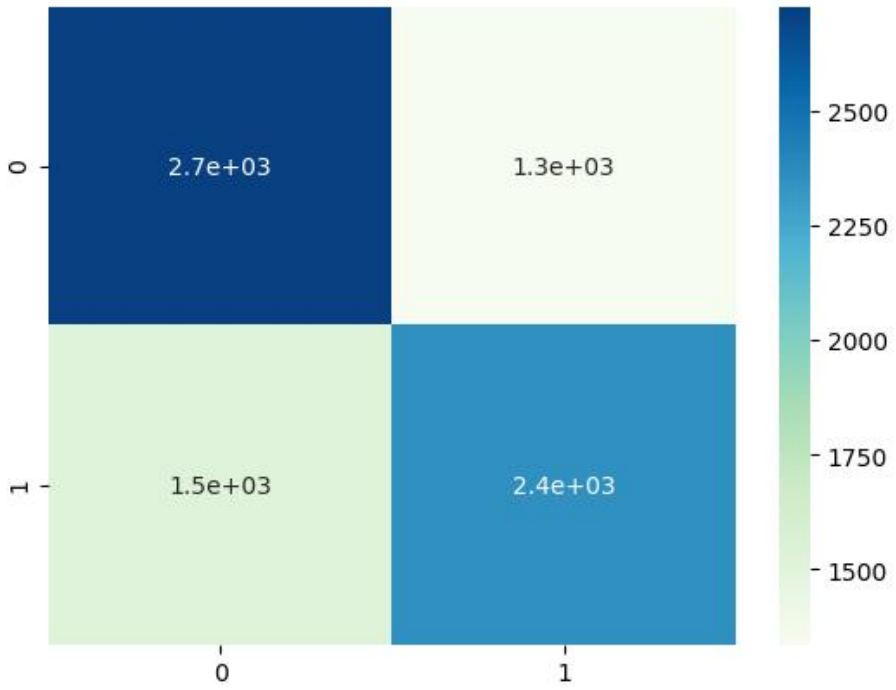
$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$$

说明：xgboost 定义了γ和λ，可以自定义值，γ越大，对较多叶子节点的树的惩罚越大，即表示希望获得结构简单的树。λ越大也是越希望获得结构简单的树。

代码：xgboost 模型

```
# XGBClassifier 模型
from xgboost import XGBClassifier
xgb_model = XGBClassifier(random_state=255)
# 训练模型
xgb_model.fit(X_train, y_train)
# 预测
y_pre = xgb_model.predict(X_test)
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
# 混淆矩阵
cm = metrics.confusion_matrix(y_test,y_pre,labels=[0,1])
sns.heatmap(cm,annot=True,cmap="GnBu")
plt.show()
# 各类评价指标
from sklearn.metrics import classification_report
print('mmmmm\n',classification_report(y_test,y_pre),'\n\n')
```

混淆矩阵及各类评价指标



	precision	recall	f1-score	support
0.0	0.64	0.67	0.66	4061
1.0	0.64	0.61	0.62	3873
avg / total	0.64	0.64	0.64	7934

注：

（1）混淆矩阵的指标

预测性分类模型，肯定是希望越准越好。那么，对应到混淆矩阵中，那肯定是希望 TP 与 TN 的数量大，而 FP 与 FN 的数量小。所以当我们得到了模型的混淆矩阵后，就需要去看有多少观测值在第二、四象限对应的位置，这里的数值越多越好；反之，在第一、三四象限对应位置出现的观测值肯定是越少越好。

（2）评价指标

精确率：TP/(TP+FP)

召回率：TP/(TP+FN)

F1-Score：F1-Score 又称为平衡 F 分数（balanced F Score），他被定义为精准率和召回率的调和平均数。

$$\mathrm{F1-Score} = \frac{2}{\frac{1}{\mathrm{precision}} + \frac{1}{\mathrm{recall}}} = 2 \times \frac{\mathrm{precision} \times \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

F1-Score 指标综合了 Precision 与 Recall 的产出的结果。F1-Score 的取值范围从 0 到 1 的，1 代表模型的输出最好，0 代表模型的输出结果最差。
更一般的，我们定义 Fβ 分数为:

$$F_\beta = (1+\beta^2) \times \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

除了 F1 分数之外，F2 分数和 F0.5 分数在统计学中也得到大量的应用。其中，F2 分数中，召回率的权重高于精准率，而 F0.5 分数中，精准率的权重高于召回率.

F1-Score 的范围在 0-1 之间，其指标越大模型好，指标越低模型越差。

（3）XGBoost 是一种基于决策树的集成机器学习算法，梯度增强为框架。在涉及非结构化数据（图像，文本等）的预测问题中，人工神经网络往往优于所有其他算法或框架。然而，当涉及到中小型结构化或表格数据时，基于决策树的算法被认为是同类中最佳的。

1.5 总结

首先：对所给数据进行探索，再做相应的数据处理，为了应用于之后的建模；

进而：分别建立聚类模型和预测模型；

最后：由 K-Means 聚类的可视化结果可以看出：1 类球队最热衷于参加友谊赛；

由决策树模型得到的各类评价指标，准确率为 0.58；而 xgboost 的准确率为 0.64，均高于 0.5，因此，可以进行比赛成绩的预测。