**非参数统计期中大作业**

——利用决策树进行NBA预测分析

**１算法介绍**

决策树是运用非参数统计理论的基本分类算法，通过对树的生成，对树枝的修剪，再利用训练得到的最佳回归参数辅助选择分类的一种分类模型，由于课本并未涉及到决策树（清华大学版有涉及），因此本次只涉及单棵决策树的模型建立的实现，多颗和随机森林改进算法只进行代码的实现而删去原理和算法的部分，该算法的精髓就是利用常见的非参数统计抽样算法：

1. Booststraping：称为自助法，一种有放回的抽样方法，他是非参数统计中一种重要的估计统计量方差进而进行区间估计的统计方法其核心思想和基本步骤如下：

（1）采用重抽样技术从原始样本中抽取一定数量（自己给定）的样本，此过程允许重复抽样。

（2）根据抽出的样本计算统计量T。

（3）重复上述N次（一般大于1000），得到统计量T。

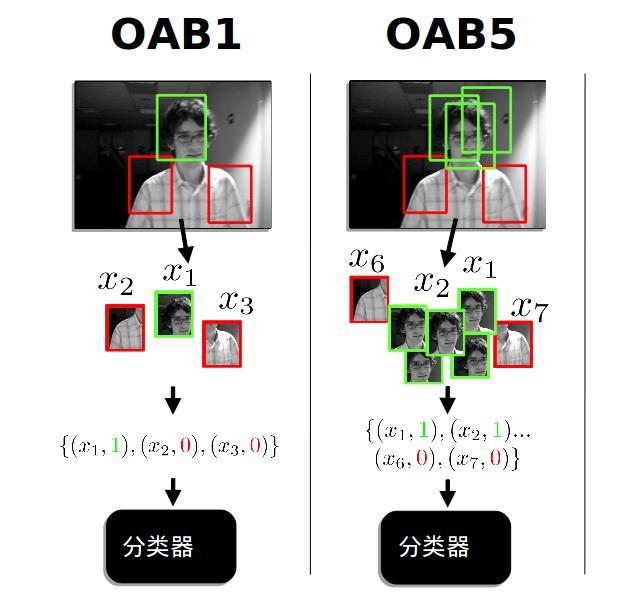
（4）计算上述N个统计量T的样本方差，得到统计量的方差。

应该说是Bootstrap是现代统计学较为流行的方法，小样本效果好，通过方差的估计可以构造置信区间等。

2 bagging：bootstrap aggregating的缩写。让该学习算法训练多轮，每轮的训练集由从初始的训练集中随机取出n个训练样本组成，某个训练样本在某训练集中可能出现多次或者不出现，训练之后可得到一个预测函数序列h\_1,h\_n，也类似于一个森林。最终的预测函数H对分类问题采用投票方式，对回归问题（加权平均好点，但是没）采用简单平均方式判别。训练R个分类器f\_i,分类器之间其他相同就是参数不相同。其中f\_i是通过从训练集和中随机取N次样本训练得到的。对于新样本，用这个R个分类器去分类，得到最多的那个类别就是这个样本的最终类别。

3 boost：其中主要的是adaboost（adaptiveboosting）。初始化时对每一个训练赋予相同的权重1/n，然后用该学算法对训练集训练t轮，每次训练后，对训练失败的训练列赋予较大的权重，也就是让学习算法在后续的学习中集中对比较难的训练列进行训练（就是把训练分类错了的样本，再次拿出来训练，看它以后还敢出错不），从而得到一个预测函数序列h\_1,h\_m,其中h\_i也有一定的权重，预测效果好的预测函数权重大，反之小。最终的预测函数H对分类问题采用有权重的投票方式，对回归问题采用加权平均的方式对新样本判别。类似bagging方法，但是训练是串行的，第K个分类器训练时，关注对前k-1分类器中错误，不是随机取样本，而是加大取这些分错的样本的权重。

最后通过一个案例图片展示分类过程：



NBA球队获胜预测是一个典型的随机问题，并且和任何分布都不相关（我并不同意课堂上很多同学说的大样本都趋向于正态分布的观点，在社会活动中我们更加倾向于泊松分布，但是样本足够大的情况下，就什么分布都不是了），我们只能通过大量的可能参数去给予权重，和回归不同的是，决策树更像一个分流程的选择（模型内部会返回多个结果），我们不会从第一步就给出答案：

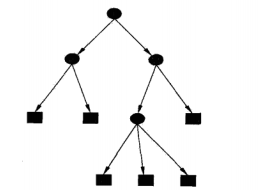
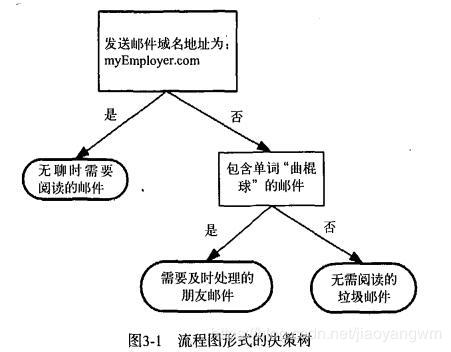


图２.1

而是通过一个流程来进行选择，就像民主选举一样，普通人是间接参与投票，最后由人大代表参与最后的投票，我可以用一个更加形象的比喻来说明模型：



他不会在第一步就返回是否需要及时处理，而是拥有一个分支：是否包含单词曲棍球。

下面给出一个伪代码为展示决策树模型的算法构造（python解释）：

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

If so return 类标签：

Else

寻找划分数据集的最好特征

划分数据集

创建分支节点

for 每个划分的子集

调用函数createBranch()并增加返回结果到分支节点中

return 分支节点

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

２选题背景

NBA球队预测通常是难以预测的，有时候并不是强队一定战胜弱队，每个队伍都有自己的特点，很可能刚好克制某一个综合能力更强的队（比如太阳队），如果我们使用逻辑斯蒂回归，就很难运用这些信息去计算，如果强行加入很可能也影响了整个模型的准确度，能否能对不同队使用不同的模型和计分方法呢，我们可以使用决策树来解决此类问题：

1. 该队是否在去年排名很高
2. 该队是否有带伤的球员
3. 该队是否连续客场
4. 该队是否处于背对背比赛（back to back）
5. 两队进攻效率和防守效率比较

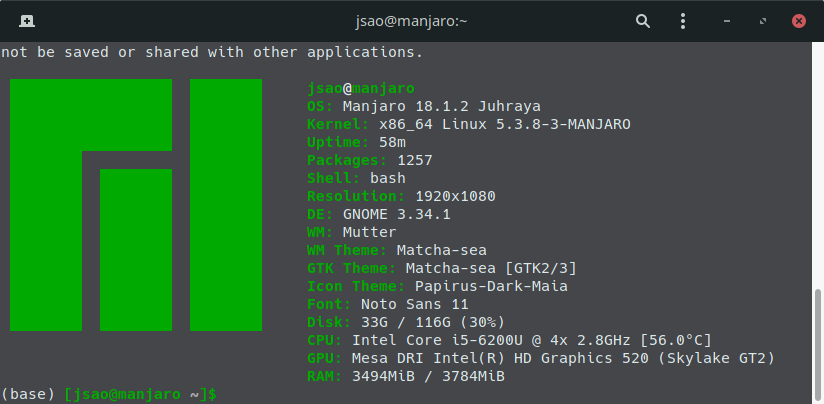
传统球迷经常使用上面的六个信息进行判断，目前在网络上并没有对此类信息完整纳入预测的开源模型，而是使用一些外行人认为的可能因素进行判断，因此我将进行一个更加完善的决策树模型建立。

３模型建立和计算

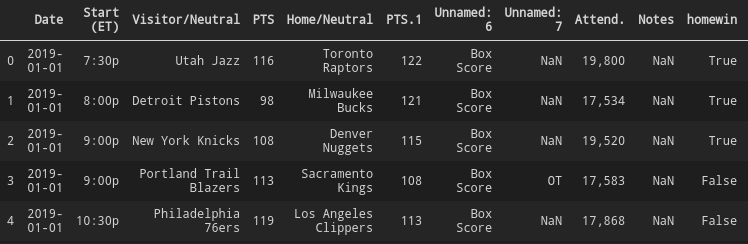
1. １数据收集

我们使用<http://www.basketball-reference.com/> 上的球赛数据进行预测，清洗后的已放入我的github: <http://www.github.com/caork/statistic>

使用python 进行数据清洗，平台：linux-manjaro , IDE: vscode+anaconda环境



从basketball-reference下载原dataset 如下：



显然对我们有用的数据有：时间，客队，主队，主队赢/输

首先导入所需的库：

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import cross\_val\_score

from collections import defaultdict

import pandas as pd

import numpy as np

from backup import \*

其中：

１．sklearn 为机器学习库，负责调用决策树核心函数；

２．Collections　为字典

３．Pandas 为数据分析库

４．Numpy 为向量运算数学库

５．Backup 为自建函数，负责简化画图流程，代码可见 <http://www.github.com/caork/statistic>

以下为数据属性的查看：

# len(df) #查看球赛数量

# out： 1312 #1312场球赛

# df.head() #查看数据前五行

# out：

'''

Date Start (ET) Visitor/Neutral ... Unnamed: 7 Attend. Notes

0 2019-01-01 7:30p Utah Jazz ... NaN 19,800 NaN

1 2019-01-01 8:00p Detroit Pistons ... NaN 17,534 NaN

2 2019-01-01 9:00p New York Knicks ... NaN 19,520 NaN

3 2019-01-01 9:00p Portland Trail Blazers ... OT 17,583 NaN

4 2019-01-01 10:30p Philadelphia 76ers ... NaN 17,868 NaN

[5 rows x 10 columns]

'''

３．２特征值提取

除了数据标签：日期，球队外，我们可参考数据特征只有胜负，所以我们首先对数据进行特征值提取：

location = r'/home/jsao/Documents/pycode/dataset.csv'

df = pd.read\_csv(location, parse\_dates=["Date"])

# 特征值提取

df['homewin'] = df['PTS'] < df['PTS.1']

y\_value = df['homewin'].values # 转化特征值供scikit-learn提取

# 创建字典储存球队比赛结果

'''

提取新特征：

1. 上一场主场是否胜利

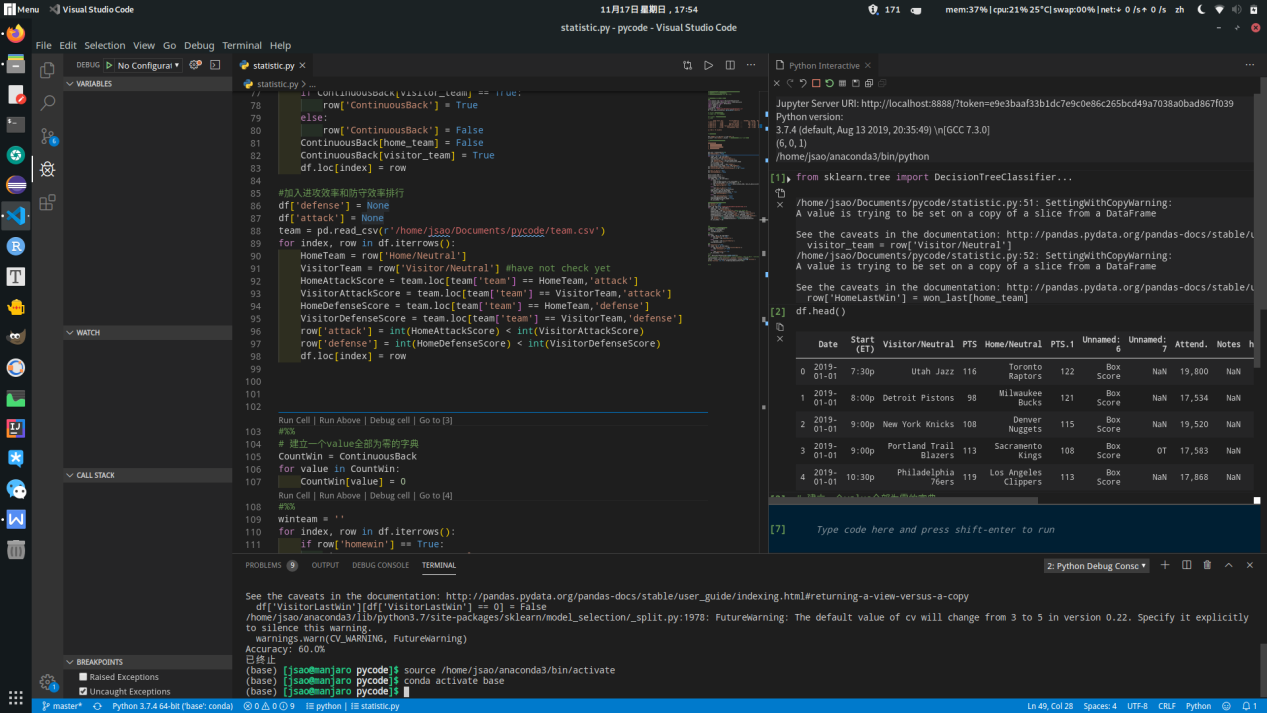
2. 上一场客场是否胜利

3. 主场是否是背靠背比赛

4. 是否是连续客场

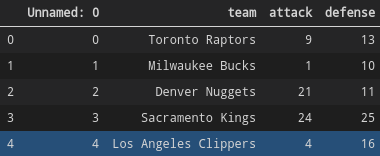
'''

调用sklearn 进行模型训练：



准确率为：56.4%

为了提高准确率，增加进攻效率和防守效率：



准确率为60.0%

可能是参数过多导致的，暂时引入随机森林模型：

rfc = RandomForestClassifier()

结果保持不变(60.0%)

附录

决策树（python）：

# 基于非参数统计-决策树对NBA篮球赛的获胜球队预测

# 采用NBA2018-2019年赛季数据

# 整合后数据已上传至github ——> http:

# 首先使用pandas读取csv数据

#%%

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import cross\_val\_score

from collections import defaultdict

import pandas as pd

import numpy as np

from backup import \*

location = r'/home/jsao/Documents/pycode/dataset.csv'

df = pd.read\_csv(location, parse\_dates=["Date"])

# len(df) #查看球赛数量

# out： 1312 #1312场球赛

# df.head() #查看数据前五行

# out：

'''

Date Start (ET) Visitor/Neutral ... Unnamed: 7 Attend. Notes

0 2019-01-01 7:30p Utah Jazz ... NaN 19,800 NaN

1 2019-01-01 8:00p Detroit Pistons ... NaN 17,534 NaN

2 2019-01-01 9:00p New York Knicks ... NaN 19,520 NaN

3 2019-01-01 9:00p Portland Trail Blazers ... OT 17,583 NaN

4 2019-01-01 10:30p Philadelphia 76ers ... NaN 17,868 NaN

[5 rows x 10 columns]

'''

# 特征值提取

df['homewin'] = df['PTS'] < df['PTS.1']

y\_value = df['homewin'].values # 转化特征值供scikit-learn提取

# 创建字典储存球队比赛结果

'''

提取新特征：

1. 上一场主场是否胜利

2. 上一场客场是否胜利

3. 主场是否是背靠背比赛

4. 是否是连续客场

'''

won\_last = defaultdict(int)

df['HomeLastWin'] = None

df['VisitorLastWin'] = None

for index, row in df.iterrows():

home\_team = row['Home/Neutral']

visitor\_team = row['Visitor/Neutral']

row['HomeLastWin'] = won\_last[home\_team]

row['VisitorLastWin'] = won\_last[visitor\_team]

df.loc[index] = row

won\_last[home\_team] = row['homewin']

won\_last[visitor\_team] = not row['homewin']

df['HomeLastWin'][df['HomeLastWin'] == 0] = False

df['VisitorLastWin'][df['VisitorLastWin'] == 0] = False

df['Back\_to\_Back'] = None

df['ContinuousBack'] = None

back\_to\_back\_home = ''

back\_to\_back\_visitor = ''

ContinuousBack = won\_last

for index, row in df.iterrows():

if index != 0:

back\_to\_back\_visitor = df.iloc[index-1, 2]

back\_to\_back\_home = df.iloc[index-1, 4]

if row['Visitor/Neutral'] not in [back\_to\_back\_home, back\_to\_back\_visitor] or row['Home/Neutral'] not in [back\_to\_back\_home, back\_to\_back\_visitor]:

row['Back\_to\_Back'] = True

else:

row['Back\_to\_Back'] = False

visitor\_team = back\_to\_back\_visitor

home\_team = back\_to\_back\_home

if ContinuousBack[visitor\_team] == True:

row['ContinuousBack'] = True

else:

row['ContinuousBack'] = False

ContinuousBack[home\_team] = False

ContinuousBack[visitor\_team] = True

df.loc[index] = row

#加入进攻效率和防守效率排行

df['defense'] = None

df['attack'] = None

team = pd.read\_csv(r'/home/jsao/Documents/pycode/team.csv')

for index, row in df.iterrows():

HomeTeam = row['Home/Neutral']

VisitorTeam = row['Visitor/Neutral'] #have not check yet

HomeAttackScore = team.loc[team['team'] == HomeTeam,'attack']

VisitorAttackScore = team.loc[team['team'] == VisitorTeam,'attack']

HomeDefenseScore = team.loc[team['team'] == HomeTeam,'defense']

VisitorDefenseScore = team.loc[team['team'] == VisitorTeam,'defense']

row['attack'] = int(HomeAttackScore) < int(VisitorAttackScore)

row['defense'] = int(HomeDefenseScore) < int(VisitorDefenseScore)

df.loc[index] = row

#%%

# 建立一个value全部为零的字典

CountWin = ContinuousBack

for value in CountWin:

CountWin[value] = 0

#%%

winteam = ''

for index, row in df.iterrows():

if row['homewin'] == True:

winteam = row['Home/Neutral']

else:

winteam = row['Visitor/Neutral']

CountWin[winteam] += 1

#%%

df['totalscore'] = None

for index, row in df.iterrows():

if row['Home/Neutral'] < row['Visitor/Neutral']:

row['totalscore'] = False

else:

row['totalscore'] = True

df.loc[index] = row

# %%

clf = DecisionTreeClassifier(random\_state=14)

#x\_previouswins = df[['HomeLastWin','VisitorLastWin','Back\_to\_Back','ContinuousBack','defense','attack','totalscore']].values

x\_previouswins = df[['totalscore','defense','attack']].values

scores = cross\_val\_score(clf,x\_previouswins, y\_value,scoring='accuracy')

print('Accuracy: {0:.1f}%'.format(np.mean(scores)\*100))

Backup.py

# %%

# 绘图

import matplotlib.pyplot as plt

import numpy as np

from pandas import DataFrame

# %%

# format transforma

def transfor(data):

tube = []

for col in data.columns:

try:

cols = data[col].to\_list()

tube.append(cols)

finally:

a = 1

tube = tuple(tube)

return tube

# make the length similar

def flat(\*data):

if len(data) == 2:

a = np.array(data[0])

b = np.array(data[1])

diff = len(a) - len(b)

if diff < 0:

b = b[0:diff]

else:

a = a[0:len(b)]

return a, b

elif len(data) == 3:

a = np.array(data[0])

b = np.array(data[1])

c = np.array(data[2])

theminimum = min(len(a), len(b), len(c))

a = a[0:theminimum]

b = b[0:theminimum]

c = c[0:theminimum]

return a, b, c

def plot(\*data, d2='line', d3='scatter'):

# 2d

if len(data) != 3:

if len(data) == 1:

a = np.arange(1, len(data[0])+1)

if isinstance(data[0], DataFrame):

b = transfor(data[0])

else:

b = data[0]

if len(data) == 2:

a = np.array(data[0])

b = np.array(data[1])

if len(data[0]) != len(data[1]):

diff = len(a) - len(b)

if diff < 0:

b = b[0:diff]

else:

a = a[0:len(b)]

print('数组有切尾长度：', abs(diff))

if isinstance(b, tuple):

plt.rcParams['figure.figsize'] = (8.0, 4.0)

plt.rcParams['figure.dpi'] = 300

for col in b:

plt.plot(a, col)

else:

plt.plot(a, b)

# 3d

if len(data) == 3:

from mpl\_toolkits.mplot3d import Axes3D

a, b, c = data

a = np.array(a)

b = np.array(b)

c = np.array(c)

fig = plt.figure()

axes3d = Axes3D(fig)

if d3 == 'scatter':

axes3d.scatter(a, b, c)

elif d3 == 'line':

axes3d.plot(a, b, c)

elif d3 == 'surface':

axes3d.plot\_surface(a, b, c)