|  |  |
| --- | --- |
| 姓名 |  |
| 评分 |  |

## 实验报告

《数学建模》实验报告

课题名称： 模糊数学在足球队排名的运用

专 业： 信息与计算科学

姓 名： 袁永润

班 级： 123212

完成日期： 2024年4月23日

### 实验报告

## 问题分析

足球队排名次问题要求我们建立一个客观的评估方法，只依据过去一段时间(几个赛季或几年)内每个球队的战绩给出各个球队的名次，具有很强的实际背景．通过分析题中12支足球队在联赛中的成绩，不难发现表中的数据残缺不全，队与队之间的比赛场数相差很大，直接根据比赛成绩来排名次比较困难。

　　下面我们利用PageRank算法的随机冲浪模型来求解．类比PageRank算法，我们可以综合考虑各队的比赛成绩为每支球队计算相应的等级分(Rank)，然后根据各队的等级分高低来确定名次，直观上看，给定球队的等级分应该由它所战胜和战平的球队的数量以及被战胜或战平的球队的实力共同决定．具体来说，确定球队Z的等级分的依据应为：一是看它战胜和战平了多少支球队；二要看它所战胜或战平球队的等级分的高低．这两条就是我们确定排名的基本原理．在实际中，若出现等级分相同的情况，可以进一步根据净胜球的多少来确定排名．由于表中包含的数据量庞大，我们先在不计平局，只考虑获胜局的情形下计算出各队的等级分，以说明算法原理。然后我们综合考虑获胜局和平局，加权后得到各队的等级分，并据此进行排名。考虑到竞技比赛的结果的不确定性，我们最后建立了等级分的随机冲浪模型，分析表明等级分排名结果具有良好的参数稳定性。

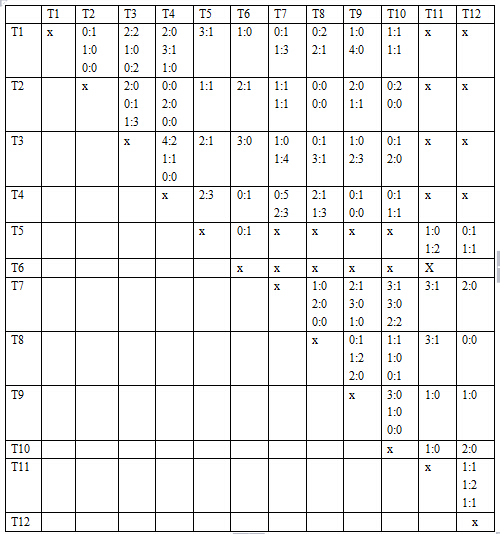
## 题目内容

下表给出了我国 12 支足球队在 1988 ~ 1989 年足球甲级队联赛中的成绩，要求

1）设计一个依据这些成绩排出诸队名次的算法，并给出用该算法排名次的结果。

2）把算法推广到任意 N 个队的情况。

3）讨论：数据应具备什么样的条件，用你的方法才能够排出诸队的名次。



## 模型建立

**获取转移概率矩阵**

首先利用有向赋权图的权重矩阵来表达出各队之间的胜负关系．用图的顶点表示相应球队，用连接两个顶点的有向边表示两队的比赛结果。同时给边赋权重，表明占胜的次数。所以，可以得到数据表中给出的12支球队所对应的权重矩阵，这个矩阵描述了各队之间的胜负关系及胜负次数。接下来，我们可以根据这个矩阵计算出各队的等级分。等级分的计算可以基于各队战胜或战平球队的等级分，同时考虑净胜球的数量。通过这种方法，我们可以初步确定各队的排名。然而，由于比赛结果的不确定性，我们还需要建立一个随机冲浪模型来进一步分析等级分的稳定性。这个模型可以帮助我们更好地理解排名结果，并确保其具有良好的参数稳定性。最终，我们可以根据这个模型给出的等级分来确定各队的最终排名。



例如，表中和比赛了两场，各胜了一场，故，。其余同理，表示曾三次战胜了。

而被战胜球队的等级应该平均分配给各个获胜球队，将权重矩阵的每个列向量进行归一化，得到的转移概率矩阵为



现在设每个队的等级分,这些等级分应有战胜的那些队的等级分确定,即：

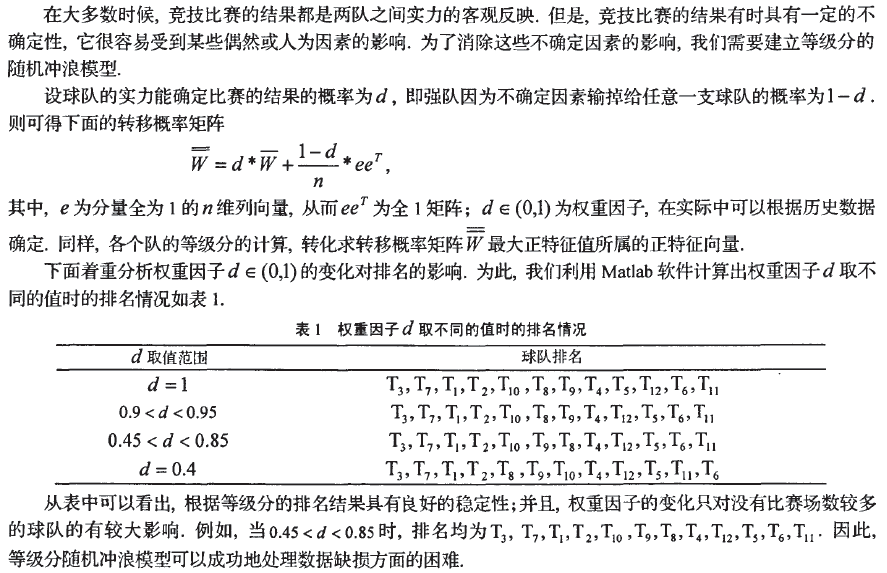
，其中为比例系数。令，则由矩阵乘法，可改写为。

即各个队的等级分的计算，转换为这个转移概率矩阵的最大特征值所属的正向特征向量，直接利用Matlab软件计算出=1,相应的等级分为(0.2371,0.2085,0.7144,0.0302,0.0026,0.003,0.456,0.2416,0.2503,0.2042,0.0005,0.0006),由此可以确定一下的排名为



上述权重不够科学，在论文中，作者提出了加权等级分，就是考虑平局的影响，对2个矩阵进行加权得到权重矩阵，从而得到转移概率矩阵。

**随机冲浪模型**



C#实现过程

**计算权重矩阵**

权重矩阵要根据测试数据，球队和每2个球队直接的比分来获取，所以我们使用一个字典来存储原始数据，将每个节点，2个队伍的比赛结果比分都写成数组的形式，来根据胜平负的场次计算积分，得到边的权重。

/// <summary>根据比赛成绩，直接根据积分来构造权重矩阵，根据i,对j比赛获取的分数</summary>

/// <param name="data">key为2个对的边名称，value是比分列表，分别为主客进球数</param>

/// <param name="teamInfo">球队的编号列表</param>

/// <returns>权重矩阵</returns>

public static double[,] CalcLevelTotalScore(Dictionary<String, Int32[][]> data, List<Int32> teamInfo)

{

Int32 N = teamInfo.Count;

double[,] result = new double[N, N];

#region 利用对称性，只计算一半

for (int i = 1; i < N; i++)

{

for (int j = i + 1; j <= N; j++)

{

#region 循环计算

String key = String.Format("{0}-{1}", teamInfo[i - 1], teamInfo[j - 1]);

//不存在比赛成绩

if (!data.ContainsKey(key))

{

result[i - 1, j - 1] = result[j - 1, i - 1] = 0;

continue;

}

//计算i,j直接的互胜场次

var scores = data[key];//i,j直接的比分列表

var Si3 = scores.Where(n => n[0] > n[1]).ToList();//i胜场次

var S1 = scores.Where(n => n[0] == n[1]).ToList();//i平场次

var Si0 = scores.Where(n => n[0] < n[1]).ToList();//i负场次

result[i - 1, j - 1] = Si3.Count\*3 + S1.Count ;

result[j - 1, i - 1] = Si0.Count \*3 + S1.Count ;

#endregion

}

}

#endregion

//按照列向量进行归一化

return GetNormalizedByColumn(result);

}

上面最后返回调用了归一化的函数，比较简单，直接代码贴出来，折叠一下：

/// <summary>按照列向量进行归一化</summary>

/// <param name="data"></param>

/// <returns></returns>

public static double[,] GetNormalizedByColumn(double[,] data)

{

int N = data.GetLength(0);

double[,] result = new double[N, N];

#region 各个列向量归一化

for (int i = 0; i < N; i++) //列

{

double sum = 0;

//行

for (int j = 0; j < N; j++) sum += data[j, i];

for (int j = 0; j < N; j++)

{

if (sum != 0) result[j, i] = data[j, i] / (double)sum;//归一化,每列除以和值

else result[j, i] = data[j, i];

}

}

#endregion

return result;

}

3.2计算最大特征值

/// <summary>求最大特征值下的特征向量</summary>

/// <param name="data"></param>

/// <returns></returns>

public static double[] GetEigenVectors(double[,] data)

{

var formatProvider = (CultureInfo)CultureInfo.InvariantCulture.Clone();

formatProvider.TextInfo.ListSeparator = " ";

int N = data.GetLength(0);

Matrix<double> A = DenseMatrix.OfArray(data);

var evd = A.Evd();

var vector = evd.EigenVectors;//特征向量

var ev = evd.EigenValues;//特征值，复数形式发

if (ev[0].Imaginary > 0) throw new Exception("第一个特征值为复数");

//取 vector 第一列为最大特征向量

var result = new double[N];

for (int i = 0; i < N; i++)

{

result[i] =Math.Abs(vector[i, 0]);//第一列，取绝对值

}

return result;

}

3.3 随机冲浪模型的实现

/// <summary>获取随机冲浪模型的 转移矩阵:

/// 作用很明显，结果有明显的改善

/// </summary>

/// <returns></returns>

public static double[,] GetRandomModeVector(double[,] data ,double d = 0.35)

{

int N = data.GetLength(0);

double k = (1.0 - d) / (double)N;

double[,] result = new double[N, N];

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++) result[i, j] = data[i, j] \* d + k;

}

return result;

}

4.算法测试

/// <summary>

/// 获取测试的数据集，key=对1-对2,value = int[,] 为比分

/// </summary>

public static Dictionary<String, Int32[][]> GetTestData()

{

Dictionary<String, Int32[][]> data = new Dictionary<string, int[][]>();

#region 依次添加数据

#region T1

data.Add("1-2", new Int32[][]{ new Int32[] { 0, 1 }, new Int32[] { 1, 0 }, new Int32[] { 0, 0 } });

data.Add("1-3", new Int32[][] { new Int32[] { 2, 2 }, new Int32[] { 1, 0 }, new Int32[] { 0, 2 } });

data.Add("1-4", new Int32[][] { new Int32[] { 2, 0 }, new Int32[] { 3, 1 }, new Int32[] { 1, 0 } });

data.Add("1-5", new Int32[][] { new Int32[] { 3, 1 } });

data.Add("1-6", new Int32[][] { new Int32[] { 1, 0 } });

data.Add("1-7", new Int32[][] { new Int32[] { 0, 1 }, new Int32[] { 1, 3 } });

data.Add("1-8", new Int32[][] { new Int32[] { 0, 2 }, new Int32[] { 2, 1 } });

data.Add("1-9", new Int32[][]{ new Int32[] { 1, 0 }, new Int32[] { 4, 0 } });

data.Add("1-10", new Int32[][]{ new Int32[] { 1, 1 }, new Int32[] { 1, 1 } });

#endregion

#region T2

data.Add("2-3", new Int32[][] { new Int32[] { 2, 0 }, new Int32[] { 0, 1 }, new Int32[] { 1, 3 } });

data.Add("2-4", new Int32[][] { new Int32[] { 0, 0 }, new Int32[] { 2, 0 }, new Int32[] { 0, 0 } });

data.Add("2-5", new Int32[][] { new Int32[] { 1, 1 } });

data.Add("2-6", new Int32[][] { new Int32[] { 2, 1 } });

data.Add("2-7", new Int32[][] { new Int32[] { 1, 1 }, new Int32[] { 1, 1 } });

data.Add("2-8", new Int32[][] { new Int32[] { 0, 0 }, new Int32[] { 0, 0 } });

data.Add("2-9", new Int32[][] { new Int32[] { 2, 0 }, new Int32[] { 1, 1 } });

data.Add("2-10", new Int32[][] { new Int32[] { 0, 2 }, new Int32[] { 0, 0 } });

#endregion

#region T3

data.Add("3-4", new Int32[][] { new Int32[] { 4, 2 }, new Int32[] { 1, 1 }, new Int32[] { 0, 0 } });

data.Add("3-5", new Int32[][] { new Int32[] { 2, 1 } });

data.Add("3-6", new Int32[][] { new Int32[] { 3, 0 } });

data.Add("3-7", new Int32[][] { new Int32[] { 1, 0 }, new Int32[] { 1, 4 } });

data.Add("3-8", new Int32[][] { new Int32[] { 0, 1 }, new Int32[] { 3, 1 } });

data.Add("3-9", new Int32[][] { new Int32[] { 1, 0 }, new Int32[] { 2, 3 } });

data.Add("3-10", new Int32[][] { new Int32[] { 0, 1 }, new Int32[] { 2, 0 } });

#endregion

#region T4

data.Add("4-5", new Int32[][] { new Int32[] { 2, 3 } });

data.Add("4-6", new Int32[][] { new Int32[] { 0, 1 } });

data.Add("4-7", new Int32[][] { new Int32[] { 0, 5 }, new Int32[] { 2, 3 } });

data.Add("4-8", new Int32[][] { new Int32[] { 2, 1 }, new Int32[] { 1, 3 } });

data.Add("4-9", new Int32[][] { new Int32[] { 0, 1 }, new Int32[] { 0, 0 } });

data.Add("4-10", new Int32[][] { new Int32[] { 0, 1 }, new Int32[] { 1, 1 } });

#endregion

#region T5

data.Add("5-6", new Int32[][] { new Int32[] { 0, 1 } });

data.Add("5-11", new Int32[][] { new Int32[] { 1, 0 }, new Int32[] { 1, 2 } });

data.Add("5-12", new Int32[][] { new Int32[] { 0, 1 }, new Int32[] { 1, 1 } });

#endregion

#region T7

data.Add("7-8", new Int32[][] { new Int32[] { 1, 0 }, new Int32[] { 2, 0 }, new Int32[] { 0, 0 } });

data.Add("7-9", new Int32[][] { new Int32[] { 2, 1 }, new Int32[] { 3, 0 }, new Int32[] { 1, 0 } });

data.Add("7-10", new Int32[][] { new Int32[] { 3, 1 }, new Int32[] { 3, 0 }, new Int32[] { 2, 2 } });

data.Add("7-11", new Int32[][] { new Int32[] { 3, 1 } });

data.Add("7-12", new Int32[][] { new Int32[] { 2, 0 } });

#endregion

#region T8

data.Add("8-9", new Int32[][] { new Int32[] { 0, 1 }, new Int32[] { 1, 2 }, new Int32[] { 2, 0 } });

data.Add("8-10", new Int32[][] { new Int32[] { 1, 1 }, new Int32[] { 1, 0 }, new Int32[] { 0, 1 } });

data.Add("8-11", new Int32[][] { new Int32[] { 3, 1 } });

data.Add("8-12", new Int32[][] { new Int32[] { 0, 0 } });

#endregion

#region T9

data.Add("9-10", new Int32[][] { new Int32[] { 3, 0 }, new Int32[] { 1, 0 }, new Int32[] { 0, 0 } });

data.Add("9-11", new Int32[][] { new Int32[] { 1, 0 } });

data.Add("9-12", new Int32[][] { new Int32[] { 1, 0 } });

#endregion

#region T10

data.Add("10-11", new Int32[][] { new Int32[] { 1, 0 } });

data.Add("10-12", new Int32[][] { new Int32[] { 2, 0 } });

#endregion

#region T11

data.Add("11-12", new Int32[][] { new Int32[] { 1, 1 }, new Int32[] { 1, 2 }, new Int32[] { 1, 1 } });

#endregion

#endregion

return data;

}

var team = new List<Int32>(){1,2,3,4,5,6,7,8,9,10,11,12};

var data = GetTestData();

var k3 = CalcLevelScore3(data,team);

var w3 = GetEigenVectors(k3);

var teamOrder = TeamOrder(w3,team);

Console.WriteLine(teamOrder.ArrayToString());

排序结果如下：

7,3,1,9,8,2,10,4,6,5,12,11