



# 人工智能——自动驾驶

武迪、张思

# 本次课程内容

- 任务一：NBA球员成绩预测
- 任务二：K-means聚类算法
- 学生表演、网页互动
- 任务三：示例——NBA球员控球后卫聚类
- 任务四：课堂作业+essay

# 作业回顾：NBA球员分数预测

- 问题一、谁是与Lebron James最像的球员
- 问题分析：
- 1、读取数据

```
import pandas
import math
with open("nba_2013.csv", 'r') as csvfile:
    nba = pandas.read_csv(csvfile)

# The names of all the columns in the data.
print(nba.columns.values)
```

- 2、数据处理：将表格中所有NA用0代替

```
# Fill in NA values with 0
nba.fillna(0, inplace=True)
```

# 作业回顾：NBA球员分数预测

- 问题一、谁是与Lebron James最像的球员
- 问题分析：
- 3、归一化（规范化）Normalizing——避免某一项影响过大
  - ✓ 选择具有数据用来做欧氏距离的列，并对每一列进行归一化，公式为：
  - ✓  $(\text{当前值} - \text{均值}) / \text{均方差}$

```
# Choose only the numeric columns (we'll use these to compute euclidean distance)
distance_columns = ['age', 'g', 'gs', 'mp', 'fg', 'fga', 'fg.', 'x3p', 'x3pa', 'x3p.',
                    'x2p', 'x2pa', 'x2p.', 'efg.', 'ft', 'fta', 'ft.', 'orb', 'drb', 'trb', 'ast', 'stl',
                    'blk', 'tov', 'pf', 'pts']
```

```
# Select only the numeric columns from the NBA dataset
nba_numeric = nba[distance_columns]
```

```
# Normalize all of the numeric columns
nba_normalized = (nba_numeric - nba_numeric.mean()) / nba_numeric.std()
print(nba_normalized)
```

# 作业回顾：NBA球员分数预测

- 问题一、谁是与Lebron James最像的球员
- 问题分析：
- 4、计算欧氏距离

```
# Select Lebron James from our dataset
selected_player = nba_normalized[nba["player"] == "LeBron James"].iloc[0]
print(selected_player)
def euclidean_distance(row):
    """
    A simple euclidean distance function
    """
    inner_value = 0
    for k in distance_columns:
        inner_value += (row[k] - selected_player[k]) ** 2
    return math.sqrt(inner_value)

# Find the distance from each player in the dataset to lebron.
lebron_distance = nba_normalized.apply(euclidean_distance, axis=1)
print(lebron_distance)
```

# 作业回顾：NBA球员分数预测

- 问题一、谁是与Lebron James最像的球员
- 问题分析：
- 5、升序排序
  - ✓ 新生成一个dataframe，内容是上一步计算好的欧氏距离
  - ✓ 对该dataframe进行升序排列

```
# Create a new dataframe with distances.  
distance_frame = pandas.DataFrame(data={"dist": lebron_distance, "idx": lebron_distance.index})  
distance_frame.sort_values("dist", inplace=True)
```

# 作业回顾：NBA球员分数预测

- 问题一、谁是与Lebron James最像的球员
- 问题分析：
- 6、选择相似性度量最近的球员
  - ✓ 注意第一个值应该为James本人，距离为0

```
# Find the most similar player to lebron (the lowest distance to lebron is lebron, the second smallest is the most similar non-lebron player)  
second_smallest = distance_frame.iloc[1]["idx"]  
most_similar_to_lebron = nba.loc[int(second_smallest)]["player"]  
print(most_similar_to_lebron)
```

# 作业回顾：NBA球员分数预测

- 问题二、预测球员分数
- 问题分析：
  - 1、找到某一个成员的k个最近邻，
  - 2、对其进行求平均值，即为某一个成员的成绩进行预测
- 思考：如何确定你的预测是否合理呢？如何证明你的算法效果好呢？
- 方法：
  - 测试集&训练集：在所给数据中，随机抽取一部分数据作为训练集，对你的算法进行训练，使用测试集为实际值，以及你根据算法求出的预测值做均方差，得到结果即为对你算法的误差评价。



# 作业回顾：NBA球员分数预测

➡ 读取数据，随机选取测试集和训练集

```
import pandas
with open("nba_2013.csv", 'r') as csvfile:
    nba = pandas.read_csv(csvfile)

# The names of all the columns in the data.
print(nba.columns.values)

# Fill in NA values in nba_normalized
nba.fillna(0, inplace=True)

import random
import math
from numpy.random import permutation

# Randomly shuffle the index of nba.
random_indices = permutation(nba.index)
# Set a cutoff for how many items we want in the test set (in this case 1/3 of the items)
test_cutoff = math.floor(len(nba)/3)
# Generate the test set by taking the first 1/3 of the randomly shuffled indices.
test = nba.loc[random_indices[1:test_cutoff]]
# Generate the train set with the rest of the data.
train = nba.loc[random_indices[test_cutoff:]]
```

# 作业回顾：NBA球员分数预测

<https://www.cnblogs.com/pinard/p/6065607.html>

- KNN求预测值，使用sklearn.neighbors中的KNeighborsRegressor函数
- fit（）函数

```
# The columns that we will be making predictions with.
x_columns = ['age', 'g', 'gs', 'mp', 'fg', 'fga', 'fg.', 'x3p', 'x3pa', 'x3p.', 'x2p', 'x2pa', 'x2p.', 'efg.', 'ft',
'fta', 'ft.', 'orb', 'drb', 'trb', 'ast', 'stl', 'blk', 'tov', 'pf']
# The column that we want to predict.
y_column = ["pts"]
```

```
from sklearn.neighbors import KNeighborsRegressor
# Create the knn model.
# Look at the five closest neighbors.
knn = KNeighborsRegressor(n_neighbors=5)
# Fit the model on the training data.
knn.fit(train[x_columns], train[y_column])
# Make point predictions on the test set using the fit model.
predictions = knn.predict(test[x_columns])
print(predictions)
```

<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor>

# 作业回顾：NBA球员分数预测

## ➡ 均方差计算

```
# Get the actual values for the test set.  
actual = test[y_column]  
  
# Compute the mean squared error of our predictions.  
mse = (((predictions - actual) ** 2).sum()) / len(predictions)  
print(mse)
```



# Sklearn

<http://scikit-learn.org/stable/index.html>

- Scikit-learn是数据挖掘与分析的简单而有效的工具。依赖于NumPy, SciPy和matplotlib。
- 它主要包含以下几部分内容:
- 从功能来分:
  - classification
  - Regression
  - Clustering
  - Dimensionality reduction
  - Model selection
  - Preprocessing

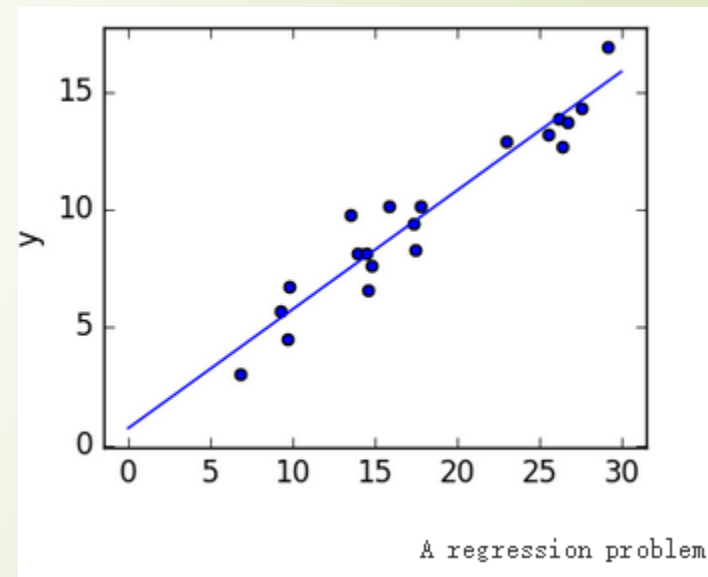
# Sklearn

- sklearn.discriminant\_analysis: Discriminant Analysis
- sklearn.linear\_model: Generalized Linear Models
- sklearn.manifold: Manifold Learning
- sklearn.metrics: Metrics
- sklearn.mixture: Gaussian Mixture Models
- sklearn.multiclass: Multiclass and multilabel classification
- sklearn.multioutput: Multioutput regression and classification
- sklearn.naive\_bayes: Naive Bayes
- sklearn.neighbors: Nearest Neighbors
- sklearn.calibration: Probability Calibration
- sklearn.cross\_decomposition: Cross decomposition
- sklearn.pipeline: Pipeline
- sklearn.preprocessing: Preprocessing and Normalization
- sklearn.random\_projection: Random projection
- sklearn.semi\_supervised: Semi-Supervised Learning
- sklearn.svm: Support Vector Machines
- sklearn.tree: Decision Tree
- sklearn.utils: Utilities

经常用到的有clustering, classification(svm, tree, linear regression 等), decomposition, preprocessing, metrics等, 所以先从这些地方学起来

# 练习： 分类问题

- 什么是预测（回归），什么是分类？
- 回归任务：对一组数据的简单最佳拟合线。
- 再次，这是一个模型拟合数据的例子，但我们的重点是模型可以对新数据进行概括。该模型已从训练数据中学习，并可用于预测测试数据的结果：在这里，我们可能会得到一个 $x$ 值，并且该模型将允许我们预测 $y$ 值。





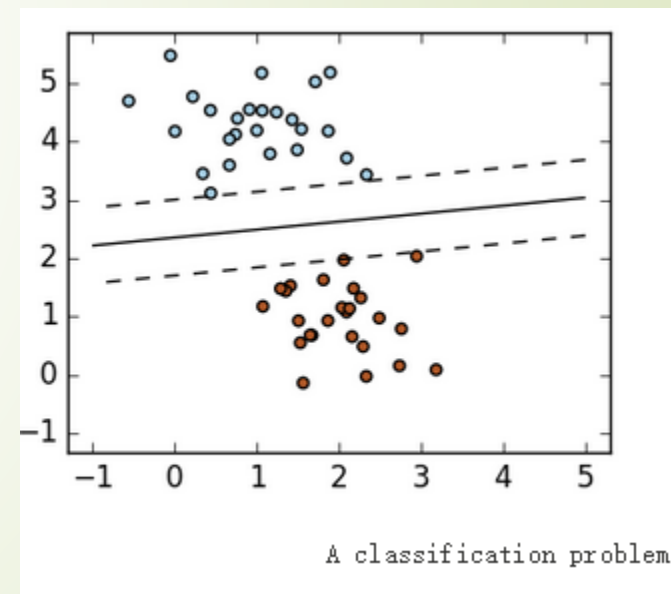
# 练习： 分类问题

- 什么是预测（回归），什么是分类？
- 分类任务：该图显示了一组二维数据，根据两个不同的类别标签着色。可以使用分类算法来绘制两个点群之间的划分边界：通过绘制这条分界线，我们已经学习了一个可以推广到新数据的模型：如果您要将另一个点放到未标记的平面上，则此算法现在可以预测它是蓝色点还是红色点。
- Iris 数据集 – 花的分类

<http://www.scipy-lectures.org/packages/scikit-learn/index.html>

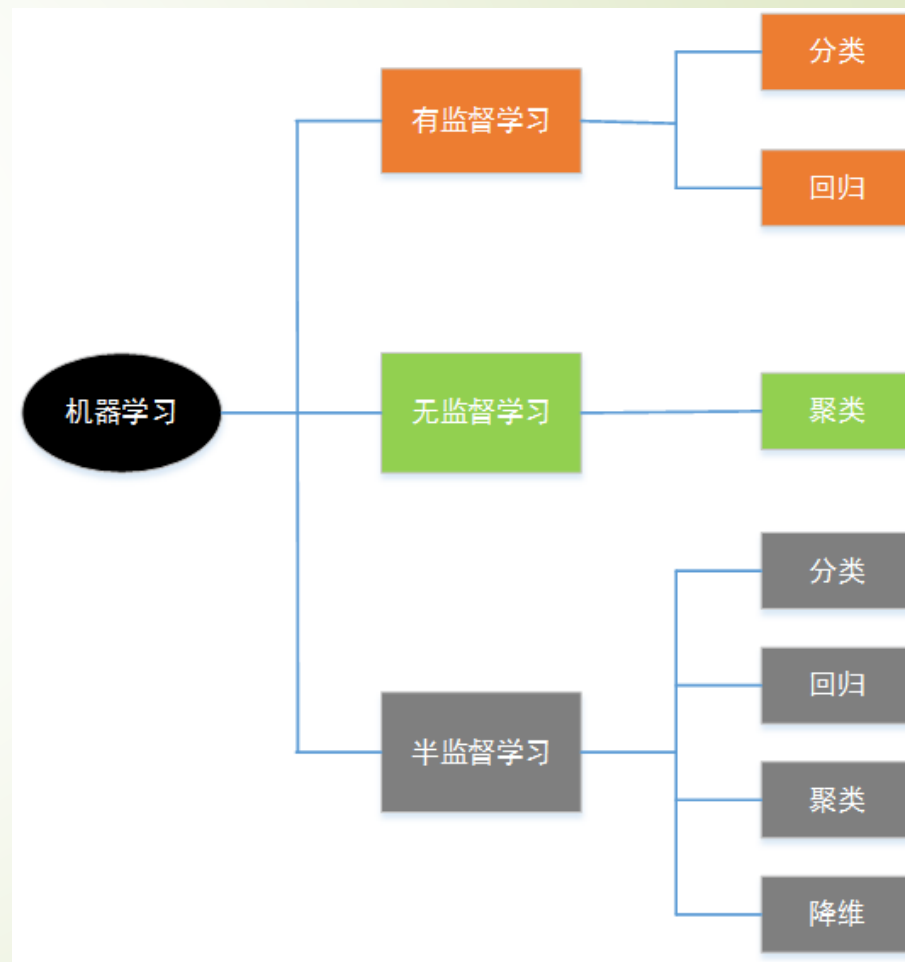
<https://zhuanlan.zhihu.com/p/31785188>

[http://scikit-learn.org/stable/auto\\_examples/datasets/plot\\_iris\\_dataset.html](http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html)



# 机器学习

- 机器学习是关于构建具有可调参数的程序，这些参数可以自动调整，以便通过适应先前看到的数据来改善它们的行为。
- 机器学习可以被认为是人工智能的一个子领域，因为这些算法可以被看作是构建模块，使计算机通过某种方式学会更智能地进行归纳、总结、预测，而不仅仅是像数据库系统那样存储和检索数据项。






# 监督学习&非监督学习

- 监督学习就是对具有标签 (lable) 的训练样本 (train data) 进行学习, 找到data和label之间的映射关系 (mapping, 更确切的说是一个function), 从而利用该映射关系对无标签的样本进行预测 (predict), 得到其标签。  
在监督学习领域, 两大研究分支是:
  - Regression (回归)
  - Classification (分类)
- 非监督学习对训练样本 (train data) 进行学习, 但只给了数据, 没有给标签 (label), 要求直接对数据进行分析建模。无监督学习里典型的例子就是聚类了。聚类的目的在于把相似的东西聚在一起, 而我们并不关心这一类是什么。
  - ✓ 比如我们去参观一个画展, 我们完全对艺术一无所知, 但是欣赏完多幅作品之后, 我们也能把它们分成不同的派别 (比如哪些更朦胧一点, 哪些更写实一些, 即使我们不知道什么叫做朦胧派, 什么叫做写实派, 但是至少我们能把他们分为两个类)。

# 学生活动

- 小组研讨K-means算法
- 任务：小组研讨K-means算法，并将你对该算法的理解展示出来，可以借助纸笔
- 分组：5个小组
- 时间：研讨（算法理解回顾）+准备（写剧本、排练）15分钟
- 特殊要求：自己准备道具，拒绝CS专业术语
- 目标：非计算机专业的人也能听懂
- 表演展示时间：每组2分钟
- 注意：每组展示会录像



# 互动演示

- Moodle平台上视频
- K-means clustering\_ how it works [360p].mp4
- 网址：
- <http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html>
- 思考：透过演示发现了什么问题？

# 聚类算法 K-means (K均值)

- 分类其实是从特定的数据中挖掘模式，作出判断的过程
- 聚类的目的也是把数据分类，但是事先我是不知道如何去分的，完全是算法自己来判断各条数据之间的相似性，相似的就放在一起。在聚类的结论出来之前，我完全不知道每一类有什么特点，一定要根据聚类的结果通过人的经验来分析，看看聚成的这一类大概有什么特点。
- 聚类算法有很多种（几十种），K-Means是聚类算法中的最常用的一种，算法最大的特点是简单，好理解，运算速度快，但是只能应用于连续型的数据，并且一定要在聚类前需要手工指定要分成几类。

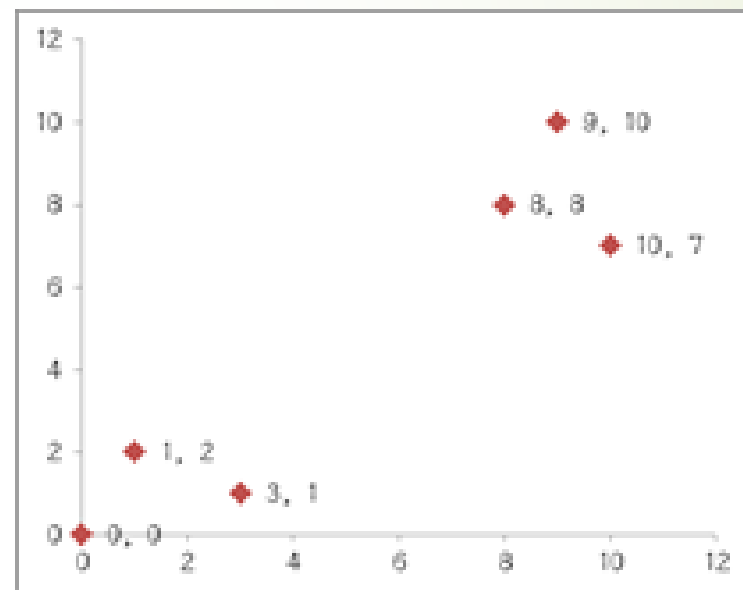
# 聚类算法 K-means (K均值)

- 下面，我们描述一下K-means算法的过程，为了尽量不用数学符号，所以描述的不是很严谨，大概就是这个意思，“物以类聚、人以群分”：
  - ✓ 首先输入k的值，即我们希望将数据集经过聚类得到k个分组。
  - ✓ 从数据集中随机选择k个数据点作为初始大哥（质心，Centroid）
  - ✓ 对集合中每一个小弟，计算与每一个大哥的距离（距离的含义后面会讲），离哪个大哥距离近，就跟定哪个大哥。
  - ✓ 这时每一个大哥手下都聚集了一票小弟，这时候召开代表大会，每一群选出新的大哥（其实是通过算法选出新的质心）。
  - ✓ 如果新大哥和老大哥之间的距离小于某一个设置的阈值（表示重新计算的质心的位置变化不大，趋于稳定，或者说收敛），可以认为我们进行的聚类已经达到期望的结果，算法终止。
  - ✓ 如果新大哥和老大哥距离变化很大，需要迭代3~5步骤。

# 聚类算法 K-means (K均值)

- 下图为已知6个点，从图上看应该分成两堆儿，前三个点一堆儿，后三个点是另一堆儿。现在手工执行K-Means，体会一下过程，同时看看结果是不是和预期一致。

	X	Y
P1	0	0
P2	1	2
P3	3	1
P4	8	8
P5	9	10
P6	10	7





# 聚类算法 K-means (K均值)

➤ case

➤ 1.选择初始大哥:  
我们就选P1和P2

➤ 2.计算小弟和大哥的距离:

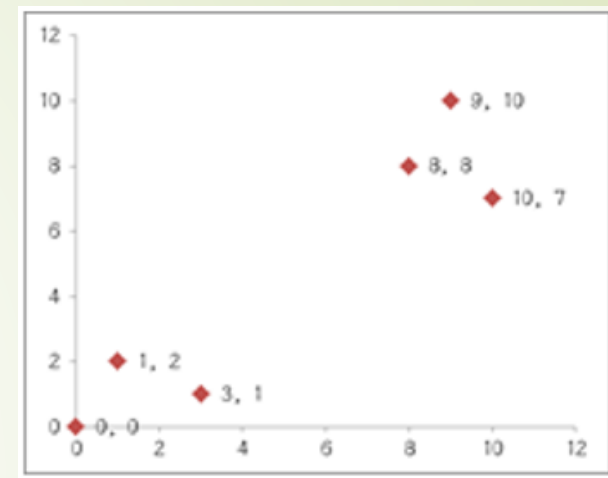
P3到P1的距离从图上也能看出来（勾股定理），是 $\sqrt{10} = 3.16$ ；P3到P2的距离 $\sqrt{(3-1)^2 + (1-2)^2} = \sqrt{5} = 2.24$ ，所以P3离P2更近，P3就跟P2混。同理，P4、P5、P6也这么算，如下：

round1

P3到P6都跟P2更近，所以第一次站队的结果是：

•组A：P1（大哥）

•组B：P2（大哥）、P3、P4、P5、P6



	P1	P2
P3	3.16	2.24
P4	11.3	9.22
P5	13.5	11.3
P6	12.2	10.3

# 聚类算法 K-means (K均值)

## ➤ 3.代表大会:

组A没啥可选的，大哥还是P1自己

组B有五个人，需要选新大哥，这里要注意选大哥的方法是每个人X坐标的平均值和Y坐标的平均值组成的新的点，为新大哥，也就是说这个大哥是“虚拟的”。

因此，B组选出新大哥的坐标为：P哥  $((1+3+8+9+10)/5, (2+1+8+10+7)/5) = (6.2, 5.6)$ 。

综合两组，新大哥为P1 (0, 0)，P哥 (6.2, 5.6)，而P2-P6重新成为小弟

## ➤ P3到P6都跟P2更近，所以第二次站队前选举后结果是：

- 组A: P1 (大哥)
- 组B: P哥 (虚拟) P2、P3、P4、P5、P6



# 聚类算法 K-means (K均值)

➤ 4.再次计算小弟到大哥的距离:

➤ round2



这时可以看到P2、P3离P1更近, P4、P5、P6离P哥更近, 所以第二次站队的结果是:

➤ 组A: P1、P2、P3

➤ 组B: P4、P5、P6 (虚拟大哥这时候消失)

	P1	P哥
P2	2.24	6.3246
P3	3.16	5.6036
P4	11.3	3
P5	13.5	5.2154
P6	12.2	4.0497

# 聚类算法 K-means (K均值)

- **5.第二届代表大会:**  
按照上一届大会的方法选出两个新的虚拟大哥: P哥1 (1.33, 1) P哥2 (9, 8.33), P1-P6都成为小弟
- 组A: p哥1 (虚拟, 大哥) P1、P2、P3
- 组B: p哥2 (虚拟, 大哥) P4、P5、P6
- **6.第三次计算小弟到大哥的距离:**
- round3
- 这时可以看到P1、P2、P3离P哥1更近, P4、P5、P6离P哥2更近, 所以第二次站队的结果是:
- 组A: P1、P2、P3
- 组B: P4、P5、P6
- 我们发现, 这次站队的结果和上次没有任何变化了, 说明已经收敛, 聚类结束, 聚类结果和我们最开始设想的结果完全一致。

	P哥1	P哥2
P1	1.4	12
P2	0.6	10
P3	1.4	9.5
P4	47	1.1
P5	70	1.7
P6	56	1.7

# 实例——NBA控球后卫聚类

- ▶ NBA媒体报道中，体育记者常关注少数球员并描绘这些球员的统计数据多么多么独特。但随着数据科学技术的发展，我们不禁对NBA球员之间的相似和差距持一个怀疑的态度。本案例让我们来一同使用Kmeans算法进一步探索球员们的聚类情况。
- ▶ 数据：nba\_2013.csv 2013-2014赛季球员的表现数据集。
- ▶ 一些选择的数据：
  - ▶ player – 球员名字
  - ▶ pos – 球员的位置 (position)
  - ▶ g – 出场比赛数量 (games)
  - ▶ pts – 球员得分 (points)
  - ▶ fg- 投篮命中率 (field goals)
  - ▶ ft- 罚球命中率 (free throws)
  - ▶ ppg-场均得分 (points per game)
  - ▶ PG-控球后卫 (Point Guard) 司职组织进攻安排战术 分担一小部分进攻任务
  - ▶ SG-Shooting Guard, 得分后卫
  - ▶ SF-Small Forward, 小前锋,
  - ▶ PF-PowerForward, 大前锋
  - ▶ C-Center, 中锋, 是内线防守最终端的防守核心

# 实例——NBA控球后卫聚类

- 控球后卫在球队中扮演着最重要的角色之一，因为他们的主要职责是为球队创造得分机会。
- 本案例使用简单的机器学习算法——聚类，可以让我们看到控卫的类型以及将类似的控卫组合在一起。使用2个特征值可以让我们轻松地对球员进行可视化，并且还可以更轻松地掌握聚类的工作方式。
- 对于控球后卫，人们普遍认为**助攻比率**是衡量球员创造得分机会数量的一个很好的衡量指标。我们也使用**每场比赛的分数**，因为有效的控球后卫不仅设置了得分机会，而且还自己进行了很多投篮。
- 重要步骤：准备数据、分析数据、算法实现

# 步骤一：准备数据

## ➤ 步骤一：读取、提取有效数据

- ✓ 我们需要位置（pos）中为PG（point guard）的数据
- ✓ 我们需要提取助攻比率和每场比赛的得分两个特征

## ➤ 提示：

### ➤ 1、读取数据

### ➤ 2、新建一个只包含数据集中的控球后卫的Dataframe

- ✓ pos中为PG的是控球后卫
- ✓ 筛选后的数据放在point\_guards里

### ➤ 3、寻找或计算特征值之一：每场得分

### ➤ 4、寻找或计算特征值之二：助攻比率

### ➤ 5、将数据画出

# 步骤一：准备数据

- 提示：
- 1、读取数据
- 2、新建一个只包含数据集中的控球后卫的Dataframe
  - ✓ pos中为PG的是控球后卫
  - ✓ 筛选后的数据放在point\_guards里

#读取数据

```
import pandas as pd
```

```
import numpy as np
```

```
nba = pd.read_csv("nba_2013.csv")
```

#新建一个point\_guards的Dataframe，放入nba中pos列为PG的内容

```
point_guards = nba[nba['pos'] == 'PG']
```



# 步骤一：准备数据

- 提示：
- 3、每场得分：由于所给dataset中并没有ppg（每场得分情况），但有pts（总得分）和g（比赛数），因此我们通过 $ppg = pts / g$ 来进行计算。
- 4、助攻比率：新建一列，atr，用于助攻比率，它是通过总助攻（ast）除以总失误数（tov）计算出来的。（这里放弃总失误为0的球员。）

#ppg计算

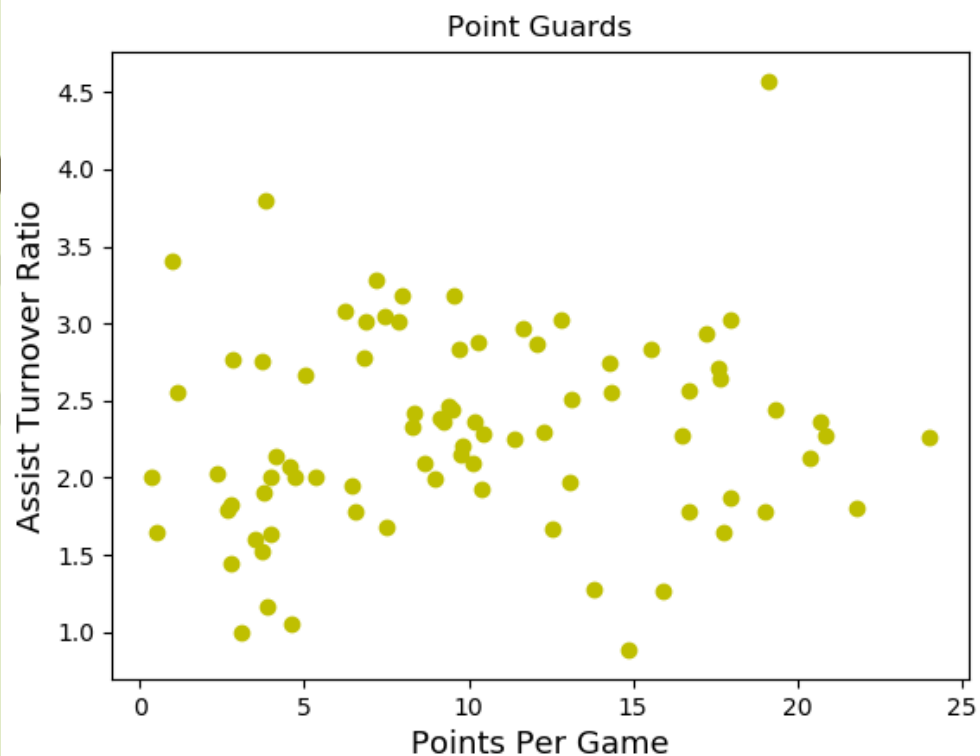
```
point_guards = point_guards[point_guards['g'] != 0]  
point_guards['ppg'] = point_guards['pts'] / point_guards['g']
```

#atr计算

```
point_guards = point_guards[point_guards['tov'] != 0]  
point_guards['atr'] = point_guards['ast'] / point_guards['tov']
```

## 步骤二：分析数据

- 步骤二、图形可视化
- 使用matplotlib在二维坐标轴上创建散点图，每个球员的每场得分（ppg）为X轴和助攻比率为Y轴（atr）。



```
#调用matplotlib函数绘图
import matplotlib.pyplot as plt
plt.scatter(point_guards['ppg'], point_guards['atr'], c='y')
plt.title("Point Guards")
plt.xlabel('Points Per Game', fontsize=13)
plt.ylabel('Assist Turnover Ratio', fontsize=13)
plt.show()
```

### Reference list

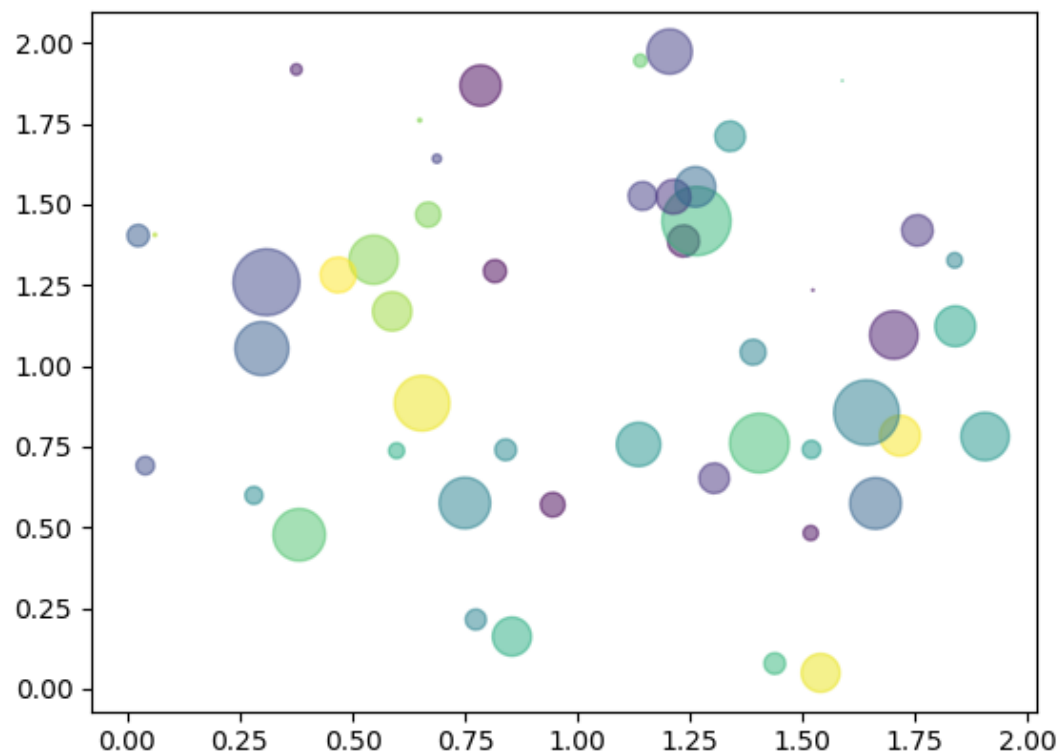
<http://blog.csdn.net/Veahlin/article/details/51941285>

<https://matplotlib.org/gallery.html>



# Matplotlib.pyplot.scatter

➡ <http://blog.csdn.net/mg2flyingff/article/details/53415353>



```
import numpy as np
import matplotlib.pyplot as plt
```

```
N = 50 # 点的个数
x = np.random.rand(N) * 2 # 随机产生50个0~2之间的x坐标
y = np.random.rand(N) * 2 # 随机产生50个0~2之间的y坐标
colors = np.random.rand(N) # 随机产生50个0~1之间的颜色值
area = np.pi * (15 * np.random.rand(N))**2 # 点的半径范围:0~15
# 画散点图
plt.scatter(x, y, s=area, c=colors, alpha=0.5, marker=(9, 3, 30))
plt.show()
```

这里用到一个matplotlib.pyplot子库中画散点图的函数

```
matplotlib.pyplot.scatter(x, y, s=20, c=None, marker='o',
cmap=None, norm=None, vmin=None, vmax=None, alpha=None,
linewidths=None, verts=None, edgecolors=None, hold=None,
data=None, **kwargs)
```

x, y基本参数还有c, s, alpha和marker, c: 为点指定的颜色数组, s: 点的面积大小, alpha: 点的颜色的透明度, marker: 指定点标记的形状。试试 '\*', '>', '8'

## 步骤二：分析数据

- 粗略的看一下上面的图，看起来有5个区域比较集中（当然有几个游离集中区域之外）。我们可以使用一种称为聚类的方法将所有控卫分成几组相似的球员。
- 虽然当我们具有针对大量的预先标记的数据时，回归和其他有监督的机器学习技术在我们有明确的标签（评价结果）时运作良好，但有些时候我们并不具有明确的标签和结果，因此需要改用无监督的机器学习算法来探索数据集内的结构。
- 有多种聚类数据的方式，但在这里我们将重点讨论基于质心的聚类。K-Means 聚类是我们将使用的一种流行的基于质心的聚类算法。K均值中的K表示我们想要将数据分割成的簇的数量。K-Means（以及大多数无监督机器学习技术）的关键部分是我们必须指定k是什么。这既有好处又有坏处，其中一个好处是我们可以选择对我们的案例最有意义的k。我们将k设置为5，因为我们希望K-Means将我们的数据分割成5个集群。（例如：客户层级调查、服装号码尺寸的制作）

## 步骤三：K-means算法实现

➡ 伪代码

选择K个点作为初始质心

repeat

    将每个点指派到最近的质心，形成K个簇

    重新计算每个簇的质心

until 簇不发生变化或达到最大迭代次数

## 步骤三：K-means算法实现

- K-Means 是一种可以重复计算迭代算法，该算法可以重新计算每个簇的质心以及属于该簇的球员，不断迭代，直到收敛。开始，我们随机选择5个球员，并将其值作为初始的质心，形成对应的5个簇
- Step 1（将点分配给簇）
- 对于每个球员，计算该球员的坐标（本案例中即为atr&ppg的值）与每个质心坐标之间的欧氏距离。将球员分配给质心最接近或与玩家值最接近的欧氏距离的聚类。
- Step2（更新簇的新质心）
- 对于每个簇，通过计算该群集中所有点（球员）的算术平均值来计算新质心。我们通过取所有X值（atr）的平均值和该群中各点的所有Y值（ppg）的平均值来计算算术平均值。
- 迭代重复步骤1和2，直到聚类不再移动并收敛。

## 步骤三：K-means算法实现

### Step1

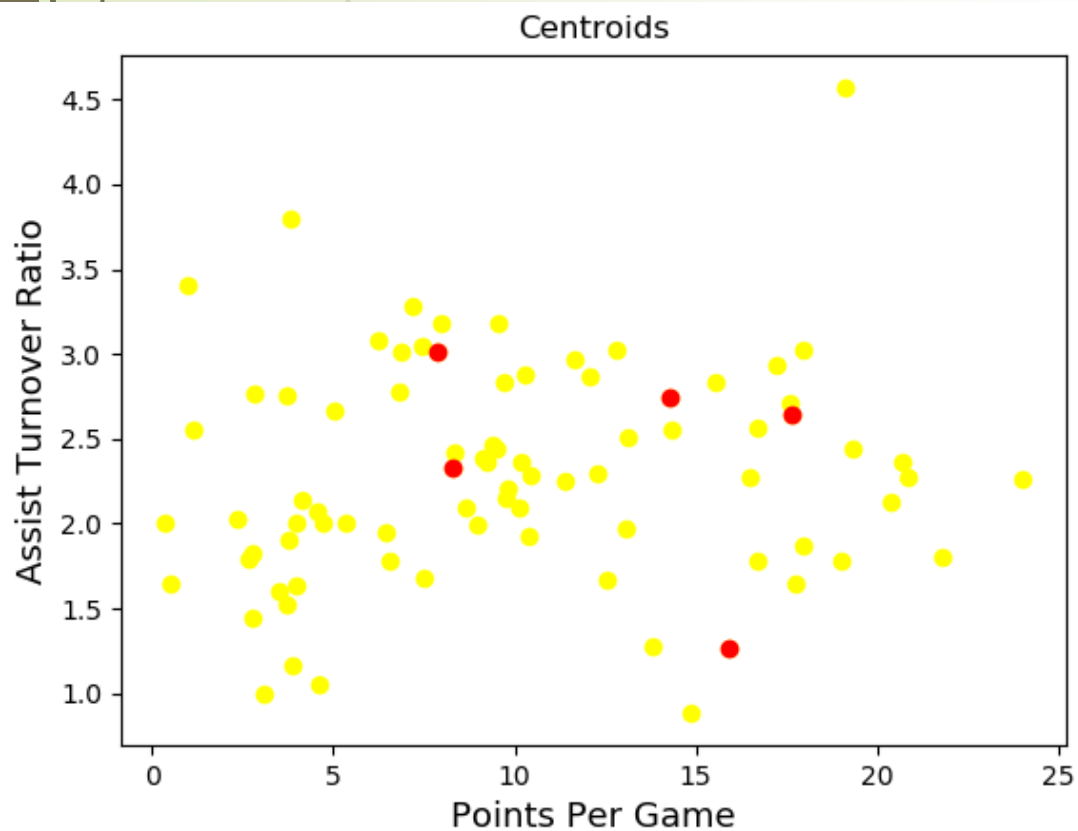
- 随机选择5个球员，作为5个簇的初始质心

```
#簇值为5
num_clusters = 5
# Use numpy's random function to generate a list, length: num_clusters, of indices
random_initial_points = np.random.choice(point_guards.index, size=num_clusters)
# centroids 随机初始化的聚类质心点，centroids是point_guards的子集
# Use the random indices to create the centroids
centroids = point_guards.loc[random_initial_points]
```

## 步骤三：K-means算法实现

### Step1

可视化初始聚类中心，中心点用红色标识，其他的点用黄色标识：



```
plt.scatter(point_guards['ppg'], point_guards['atr'], c='yellow')
plt.scatter(centroids['ppg'], centroids['atr'], c='red')
plt.title("Centroids")
plt.xlabel('Points Per Game', fontsize=13)
plt.ylabel('Assist Turnover Ratio', fontsize=13)
plt.show()
```



# 步骤三：K-means算法实现

## Step1

- 初始质心我们放在centroids Dataframe里面，但是质心位置在不断变化，直到收敛，当我们迭代新的质心值时，这个值可能是一个虚拟的，并不是任何一个球员的数值
- 因此，我们需要唯一的标识符cluster\_id来引用每个簇质心坐标和簇列表里的球员的数值（"ppg","atr"）。这里我们将质心点转化为一个字典格式，字典的键是这个簇的名称，字典的值是这个质心点的信息（"ppg","atr"）
- 字典的键，key: cluster\_id
- 字典的值，value: centroid的坐标表示为列表（ppg值，atr值）
- 为了生成cluster\_ids，我们遍历每个质心并分配一个从0到k-1的整数。例如，第一个矩心的cluster\_id为0，而第二个矩心的cluster\_id为1
- 即生成一个centroids\_dict 的字典，key是0、1、2、3、4，value是list，list内容是什么呢？是（ppg，atr）
- 编写一个函数centroids\_to\_dict，可以实现：
  - ✓ 输入质心的dataframe，生成cluster\_id，将ppg和atr值转换为坐标列表coordinates\_list
  - ✓ 创建一个cluster\_id并将ppg和 将该质心的atr值转换为坐标列表，并返回一个字典，字典key为cluster\_id，value为coordinates\_list

## 步骤三：K-means算法实现

### Step1

- 然后将质心点转化为一个字典格式，字典的键是这个簇的名称，字典的值是这个中心点的信息 ("ppg","atr")

```
def centroids_to_dict(centroids):  
    dictionary = dict()  
    # iterating counter we use to generate a cluster_id  
    # counter作为字典的键值，自增  
    counter = 0  
  
    # iterate a pandas data frame row-wise using .iterrows()  
    for index, row in centroids.iterrows():  
        coordinates = [row['ppg'], row['atr']] #list对象  
        dictionary[counter] = coordinates  
        counter += 1  
  
    return dictionary  
  
centroids_dict = centroids_to_dict(centroids)
```



# 步骤三：K-means算法实现

## Step1

- 计算欧氏距离，计算每个点到聚类中心的距离然后将每个点的聚类中心修改为离其最近的那个簇。
- 自定义函数calculate\_distance () 计算欧氏距离

```
import math
```

```
def calculate_distance(centroid, player_values):# 参数都是list对象  
    root_distance = 0
```

```
    for x in range(0, len(centroid)):  
        difference = centroid[x] - player_values[x]  
        squared_difference = difference**2  
        root_distance += squared_difference
```

```
    euclid_distance = math.sqrt(root_distance)  
    return euclid_distance
```

## 步骤三：K-means算法实现

### Step1

- 接下来需要根据基于欧氏距离将数据点分配给簇，即分配给离它最近的簇。为了简化，我们直接在point\_guard Dataframe中再添加一列 cluster\_id，表明它是（1~5）哪一簇。
- 提示：
- 自定义一个可应用于数据集每一行的函数（pandas中的apply函数）
  - ✓ 对于每个球员，通过上面的计算欧氏距离的函数计算该球员的参数与簇质心的距离，
  - ✓ 知道距离后，就可以确定该点为哪个簇，即返回cluster\_id
- 在point\_guards中生成一个新的列，用来放cluster\_id

## 步骤三：K-means算法实现

### Step1

```
def assign_to_cluster(row):  
    lowest_distance = -1 #初始值  
    closest_cluster = -1  
    #对于每一个数据，循环5次求最近的距离及cluster_id  
    for cluster_id, centroid in centroids_dict.items():  
        df_row = [row['ppg'], row['atr']]  
        euclidean_distance = calculate_distance(centroid, df_row)  
  
        if lowest_distance == -1:  
            lowest_distance = euclidean_distance  
            closest_cluster = cluster_id  
        elif euclidean_distance < lowest_distance:  
            lowest_distance = euclidean_distance  
            closest_cluster = cluster_id  
    return closest_cluster  
# 生成一个新的列cluster：存储每个节点所属的簇的键值  
point_guards['cluster'] = point_guards.apply(lambda row: assign_to_cluster(row), axis=1)
```

## 步骤三：K-means算法实现

### Step1

- 上面根据随机初始化的簇及其键值，将其它的点分配给这些簇。自定义一个函数 `visualize_clusters` 可视化第一次随机初始化的聚类图，将不同的簇用不同的颜色表示出来：

```
# Visualizing clusters
def visualize_clusters(df, num_clusters):
    colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']

    for n in range(num_clusters):
        clustered_df = df[df['cluster'] == n]
        plt.scatter(clustered_df['ppg'], clustered_df['atr'], c=colors[n])
    for n in range(num_clusters): #print([centroids_dict[n][0]], [centroids_dict[n][1]])
        plt.scatter([centroids_dict[n][0]], [centroids_dict[n][1]], c=colors[n], marker='*', s=200)

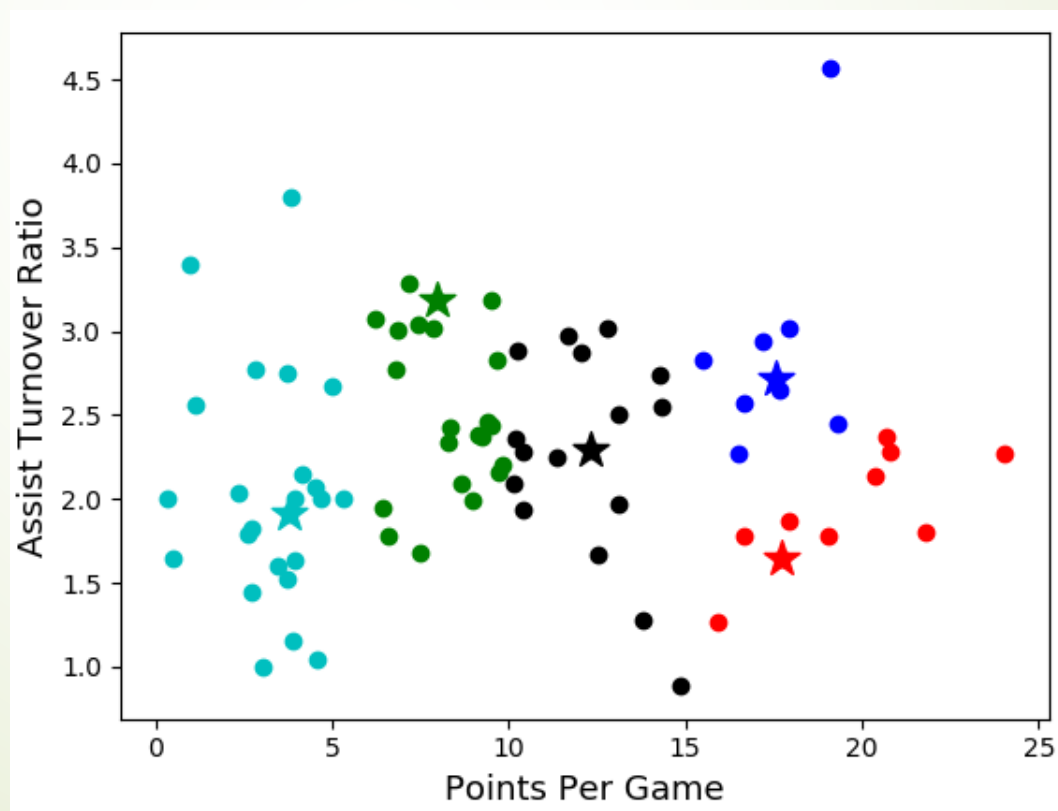
    plt.xlabel('Points Per Game', fontsize=13)
    plt.ylabel('Assist Turnover Ratio', fontsize=13)
    plt.show()

visualize_clusters(point_guards, 5)
```

## 步骤三：K-means算法实现

### Step1

- `plt.scatter([centroids_dict[n][0]], [centroids_dict[n][1]], c=colors[n], marker='*', s=200)`
- marker的不同尝试：





## 步骤三：K-means算法实现

### Step2

➤ 重新计算每个簇的质心

➤ 提示：

➤ 自定义函数

recalculate\_centroids:

- ✓ 输入point\_guards
- ✓ 使用每个cluster\_id（从0到num\_clusters - 1）找到每个簇中的所有球员
- ✓ 重新计算算数平均数
- ✓ 将cluster\_id和新的算数平均数放入new\_centroids\_dict中，生成新的字典返回

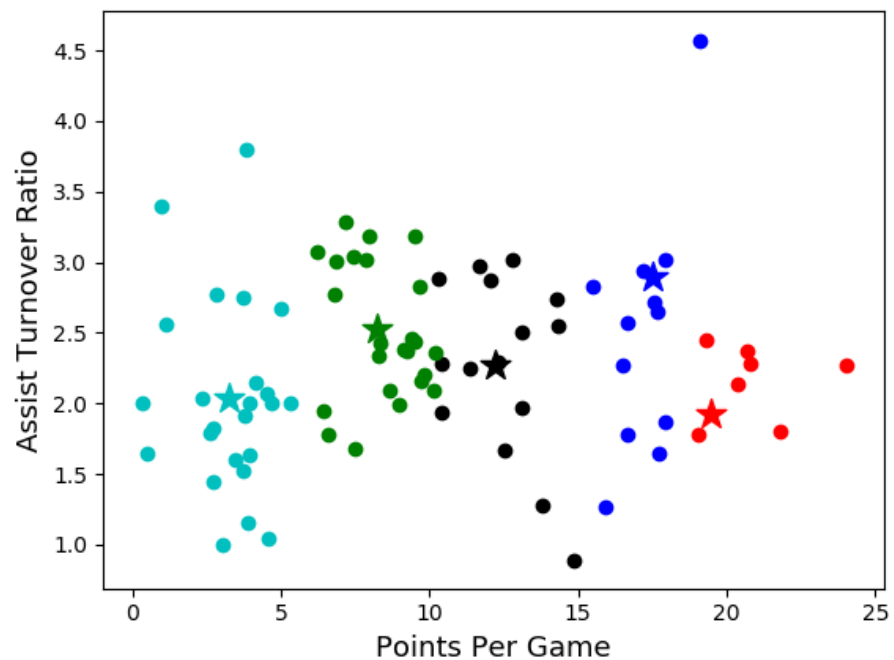
```
def recalculate_centroids(df):  
    new_centroids_dict = dict()  
    # 0..1...2...3...4  
    for cluster_id in range(0, num_clusters):  
        values_in_cluster = df[df['cluster'] == cluster_id]  
        # Calculate new centroid using mean of values in the cluster  
        new_centroid = [np.average(values_in_cluster['ppg']),  
                        np.average(values_in_cluster['atr'])]  
        new_centroids_dict[cluster_id] = new_centroid  
    return new_centroids_dict #将算数平均值放入字典key为  
cluster_id的value里  
  
centroids_dict = recalculate_centroids(point_guards)
```



## 步骤三：K-means算法实现 repeat Step1

- 重新计算了簇中心后，又要计算每个点到新簇中心的距离，将所有节点重新分给离其最近的那个簇中（跟1相差不大）：

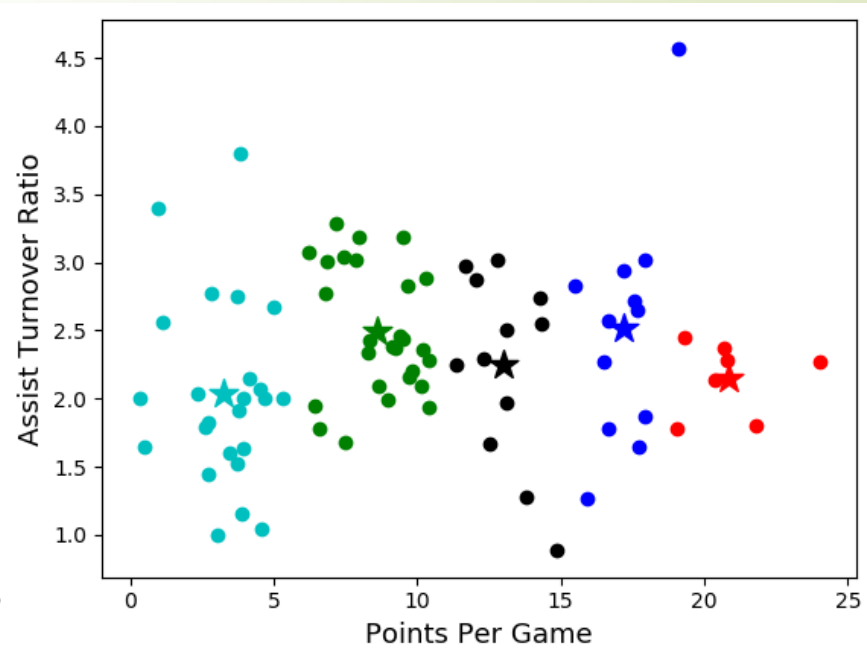
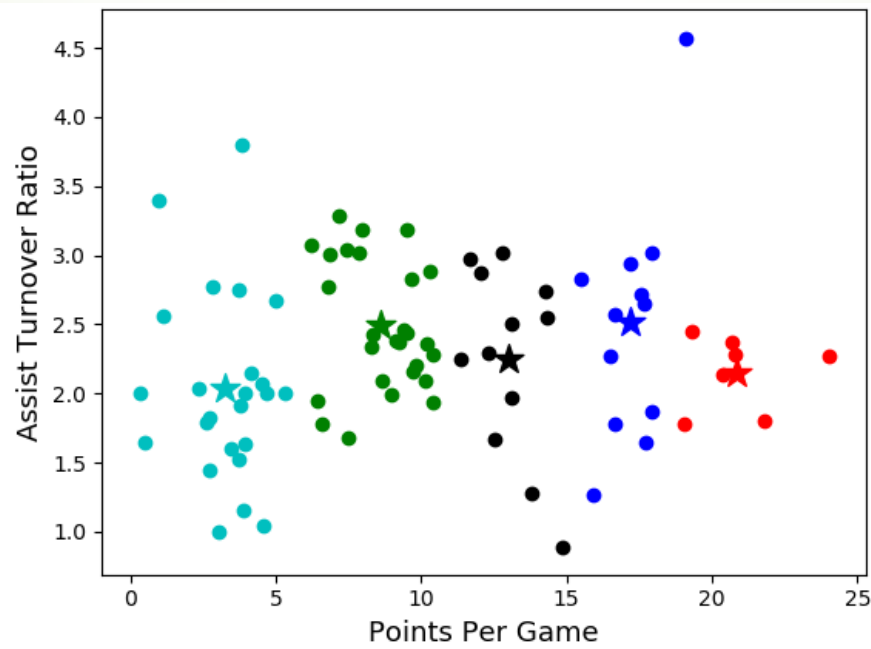
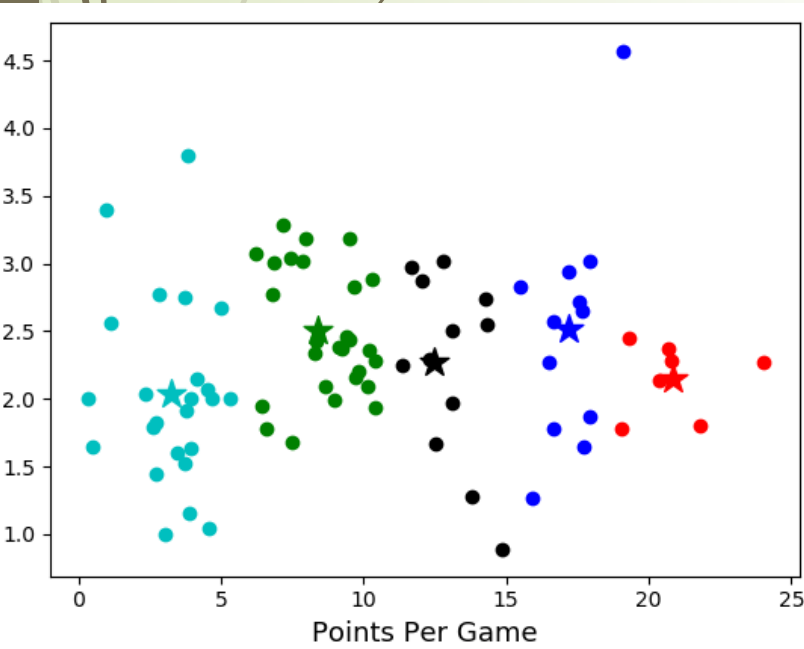
```
point_guards['cluster'] = point_guards.apply(lambda row: assign_to_cluster(row), axis=1)  
visualize_clusters(point_guards, num_clusters)
```



## 步骤三：K-means算法实现 Repeat Step 2 And Step 1

➡ 然后一直重复步骤2和步骤1，调整簇质心，再调整每个点的所属的簇值

```
centroids_dict = recalculate_centroids(point_guards)
point_guards['cluster'] = point_guards.apply(lambda row: assign_to_cluster(row), axis=1)
visualize_clusters(point_guards, num_clusters)
```

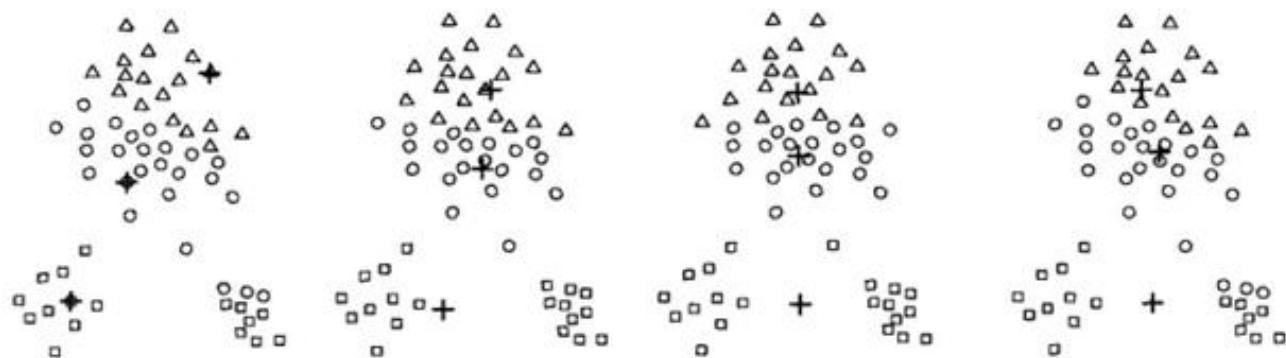


# 问题来了

- 我们的算法有没有什么问题呢？
- 局部最优解，如何解决？

## 拙劣的初始质心

当初始质心是随机的进行初始化的时候，K均值的每次运行将会产生不同的SSE,而且随机的选择初始质心结果可能很糟糕，**可能只能得到局部的最优解，而无法得到全局的最优解。**如下图所示：



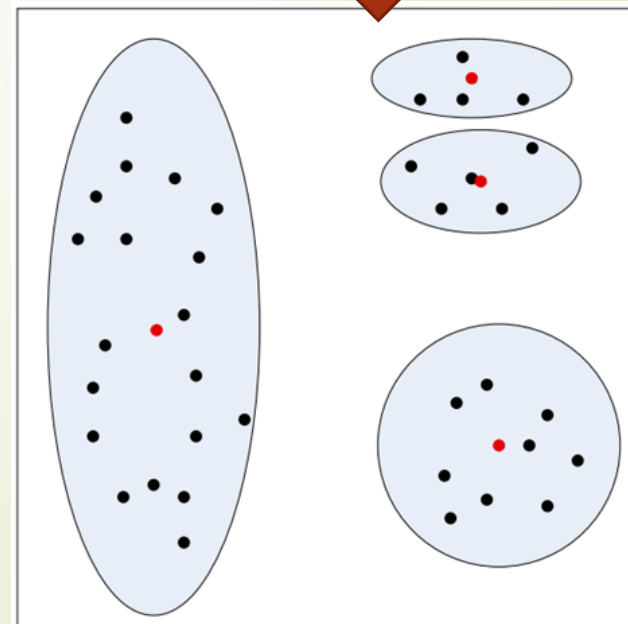
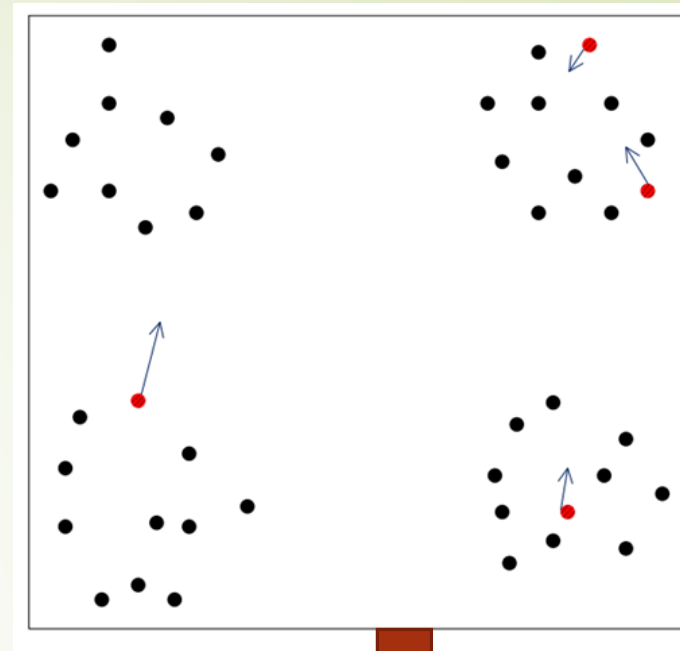
(a) 迭代 1

(b) 迭代 2

(c) 迭代 3

(d) 迭代 4

<http://blog.taoyanqi8932>



# Sklearn.cluster

- 在多次迭代过程中，我们会注意到只有一些点会在每次迭代后改变所属簇类，（特别是在两个聚类几乎重叠的地方），但是除此之外，视觉上来看，聚类在每次迭代后并没有变化很多。这意味着2件事：
- K-Means算法在迭代的过程中，对于每个簇不会引起很大的变化，因此这个算法总是收敛的并且很稳定。
- 由于K-Means算法迭代得很保守，因此最终结果与选取的初始质点有很大关系。
- 为了解决这些问题，sklearn包中的K-Means实现中做了一些智能的功能，比如重复聚类，每次随机选取质心，这比只采用一次质心选取所带来的偏差要少很多。

```
from sklearn.cluster import Kmeans
```

```
kmeans = KMeans(n_clusters=num_clusters)  
kmeans.fit(point_guards[['ppg', 'atr']])  
point_guards['cluster'] = kmeans.labels_
```

```
visualize_clusters(point_guards, num_clusters)
```

# sklearn.cluster.Kmeans

- `class sklearn.cluster.KMeans(n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto')[source]`
- `n_clusters`: 簇的个数，即你想聚成几类
- `init`: 初始簇中心的获取方法
- `n_init`: 获取初始簇中心的更迭次数，为了弥补初始质心的影响，算法默认会初始10次质心，实现算法，然后返回最好的结果。
- `max_iter`: 最大迭代次数（因为kmeans算法的实现需要迭代）
- `tol`: 容忍度，即kmeans运行准则收敛的条件
- 参考：[http://blog.csdn.net/xiaoyi\\_zhang/article/details/52269242](http://blog.csdn.net/xiaoyi_zhang/article/details/52269242)
- <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
- [http://blog.csdn.net/xiaoyi\\_zhang/article/details/52269242](http://blog.csdn.net/xiaoyi_zhang/article/details/52269242)



# 总结

- 本课程中我们通过控卫的助攻失误率和每场球赛的平均得分2个特征值将控卫分为5大类型。我们也可以利用球员的更多信息来进行聚类。得到了这些聚类信息，再去获取每簇中的球员的信息，你就会发现，一个簇中的球员的水平相当，如果一个簇中有一个球员得到的关注很大，那么该簇内其它球员同样可能得到很大的关注，因为他们属于一个高水平的球员簇中
- 同学们可以将探讨如何使用更多特征值进行聚类，以及如何在不使用质心的情况下对数据进行聚类。
- 我们也体验了使用sklearn的带来的便利，可以用简单的几行实现，以便快速获得我们需要完成的任务。在这几行中，我们运行并可视化了一个强大的K均值聚类实现的结果。在实际案例中使用K-means时，请使用sklearn实现。



# 作业

- 1、花的聚类 iris的dataset
- [http://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_cluster\\_iris.html](http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_iris.html)
- <https://www.kaggle.com/shinto/k-means-clustering/notebook>
- 2、essay
- 思考:
- 如何评价K取的好不好，聚类聚的好不好呢？
- 你能想到从哪些方面优化K-means这个算法呢？