无监督学习

胡琛

# Introduction

无监督学习的目标：利用无标签的数据学习数据的分布或数据与数据之间的关系

* 有监督学习与无监督学习最大区别在于数据是否有标签
* 无监督学习最常见的应用场景有聚类和降维

## 聚类

根据数据的 “相似性” 将数据分为多类的过程，评估两个不同样本的 ”相似性“， 通常采用的办法是计算两个样本之间的 ”距离“，使用不同的方法计算得到的样本 间的距离会关系到聚类结果的好坏。

### 计算距离的不同方法

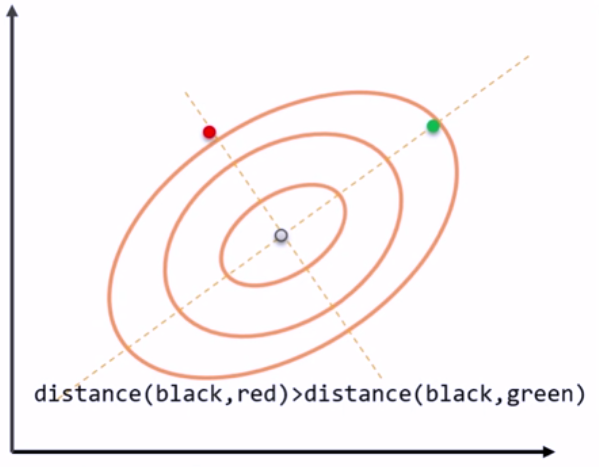
1. 欧式距离： 欧式几何中，两个点直接的距离

* \begin{equation}  
   d=\sqrt{\sum\limits\_{k=1}^{n}(x\_{1k}-x\_{2k})^2  
  \end{equation}

1. 曼哈顿距离： 也成为 ”城市街区距离“，类似于在城市中从一个十字路口到另一个 十字路口的距离

* \begin{equation}  
   d = \sum\limits\_{k=1}^n|x\_{1k}-x\_{2k}|  
  \end{equation}

1. 马氏距离：表示数据的协方差距离，是一种尺度无关的度量方式。计算时，马氏距离会 先将样本点的各个属性标准化，然后计算样本间的距离。

* \begin{equation}  
   d(x\_i,x\_j) = \sqrt{(x\_i-x\_j)^Ts^{-1}(x\_i-x\_j)}  
  \end{equation}
* 其中， 是协方差矩阵。
* 
* 马氏距离示意图

1. 夹角余弦：用向量空间中两个向量夹角的余弦值作为衡量两个样本差异大小。余弦值越接近 1，表明两个向量夹角越接近 0，即两个向量越相似。

* \begin{equation}  
   \cos(\theta) = \frac{\sum\limits\_{k=1}^nx\_{1k}x\_{2k}}  
   {\sqrt{\sum\limits\_{k=1}^nx\_{1k}^2}\sqrt{\sum\limits\_{k=1}^nx\_{2k}^2}}  
  \end{equation}

### Sklearn 中聚类模块

scikit-learn 库提供常用的聚类算法函数包含在 sklearn.cluster 这个 模块中，如 K-Means, 近邻传播算法，DBSCAN 等。

### sklearn.cluster

sklearn.cluster 模块中提供的各聚类算法函数可以使用不同的数据形式作为输入：

* 标准数据输入格式： [样本个数，特征个数] 定义的矩阵形式
* 相似性矩阵输入格式：即由 [样本数目] 定义的矩阵形式，矩阵中每一个元素为两个 样本的相似度，如 DBSCAN, AffinityPropagation 接受这种输入。如果以余弦 相似度为例，则对角线元素全为 1,矩阵中每个元素的取值范围为 [0,1].
* 模块中的代表性函数

|  |  |  |  |
| --- | --- | --- | --- |
| * 算法名称 | * 参数 | * 可扩展性 | * 相似度度量 |
| * K-means | * 聚类个数 | * 大规模数据集 | * 点间距离 |
| * DBSCAN | * 邻域大小 | * 大规模数据集 | * 点间距离 |
| * Gaussian Mixtures | * 聚类个数及其他超参 | * 复杂度高，不适合处理大规模数据集 | * 马氏距离 |
| * Birch | * 分支因子，阈值等其他超参 | * 大规模数据集 | * 两点间的欧式距离 |

## 降维

在保证数据所具有的代表性特征或者分布情况下，将高维数据转化为低维数据的过程，通常用于：

* 数据可视化
* 精简数据

### 分类与降维

* 聚类与分类都是无监督学习的典型任务，任务之间存在关联，譬如某些高维数据的分类可以通过降维 处理更好地获得；另外，学界研究也表明代表性的分类算法如 k-means 与降维算法如 NMF 之间 存在等价性。

### sklearn 库与降维

* sklearn 库提供了 7 种降维算法
* 降维过程可以被理解为对数据集的组成成分进行分解的过程，因此， sklearn 为降维模块取名为 decomposition, 调用方式为 sklearn.decomposition.

### sklearn.decomposition 中常用函数

|  |  |  |  |
| --- | --- | --- | --- |
| 算法名称 | 参数 | 可扩展性 | 适用任务 |
| PCA | 所降维度及其他超参 | 大规模数据集 | 信号处理等 |
| FastICA | 所降维度及其他超参 | 超大规模数据集 | 图形图像特征提取 |
| NMF | 所降维度及其他超参 | 大规模数据集 | 图形图像特征提取 |
| LDA | 所降维度及其他超参 | 大规模数据集 | 文本数据，主题挖掘 |

## 具体需要解决的一些问题举例

* 31 省市居民家庭消费调查
* 学生月上网时间分布调查
* 人脸图像特征抽取
* 图像分割

# K-Means

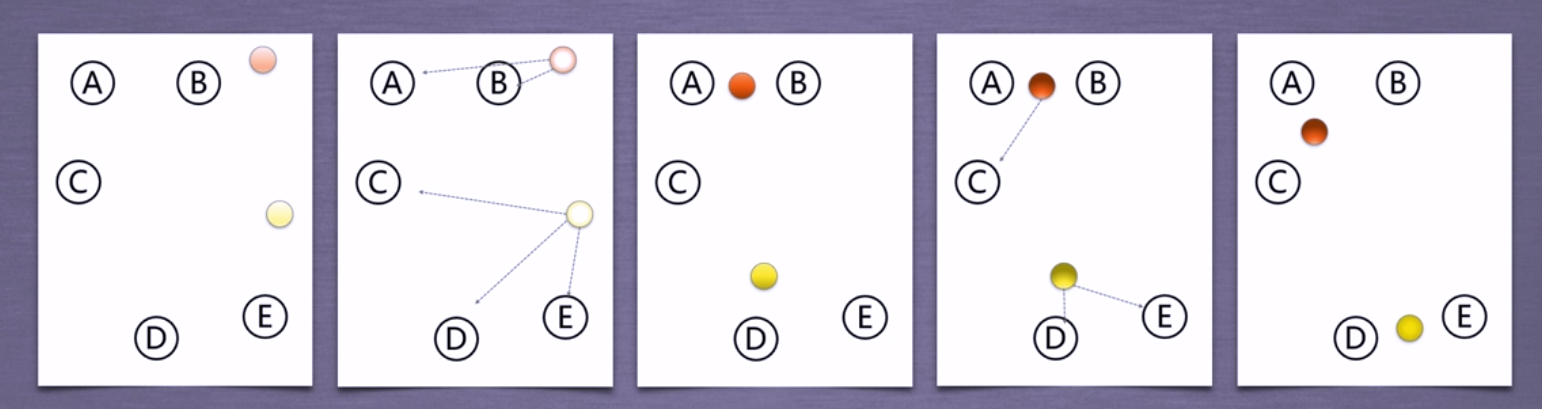
## 算法介绍

### 原理

k-means 算法以 k 为参数，把 n 个对象分成 k 个簇，使簇内具有较高的相似度，而簇间的相似度较低。

1. 随机选择 k 个点作为初始的聚类中心
2. 对于剩下的点，根据其与聚类中心的距离，将其归入最近的簇
3. 对每个簇，计算所有点的均值作为新的聚类中心
4. 重复步骤 2,3 知道聚类中心不再改变

### 聚类过程示例



聚类过程示例

## 31 省市家庭居民消费调查

### 数据介绍

现有 1999 年全国 31 个省份城镇居民家庭平均每人全年消费性支出的八个主要变量数据，这八个变量分别为： 食品、衣着、家庭设备用品及服务、医疗保健、交通和通讯、娱乐教育文化服务、居住以及杂项商品和服务。利用 已有数据，对 31 个省份进行聚类。

* 实验目的：通过聚类，了解 1999 年各个省份的消费水平在国内情况
* 技术路线：sklearn.cluster.Kmeans

### 实验过程

1. 使用算法：K-means 聚类算法
2. 实现过程：
   * 导入 sklearn 包
   * import numpy as np  
     from sklearn.cluster import KMeans
   * 加载数据，创建 K-means 算法实例，并进行训练，获得标签
     + 数据加载
     + 根据 'city.txt' 的数据格式，可以写一个 load\_data() 函数读取数据
     + def loadData(filePath):  
        fr = open(filePath, 'r+') # r+ 表示以读写方式打开一个文本文件  
        lines = fr.readlines() # readlines 表示一次读取整个文件  
        retData = []  
        retCityName = []  
        for line in lines:  
        items = line.strip().split(",")  
        retCityName.append(item[0])  
        retData.append([float(items[i]) for i in range(1, len(items))])  
        return retData, retCityName
     + 调用 K-Means 方法时可用参数：
       - nclusters: 用于指定聚类中心的个数
       - init: 初始聚类中心的初始化方法
       - maxiter: 最大的迭代次数
       - 一般调用时只需给出 nclusters 即可，init 默认为 k-means++, maxiter 默认为 300.
     + 其他参数
       - data: 加载的数据
       - label: 聚类后各数据所属的标签
       - fitpredict(): 计算簇中心以及为簇分配序号
   * if \_\_name\_\_ == '\_\_main\_\_':  
      data, cityName = loadData('city.txt')  
      km = KMeans(n\_clusters = 3)  
      label = km.fit\_predict(data)  
      expenses = np.sum(km.cluster\_centers\_, axis=1)  
       
      CityCluster = [[],[],[]]  
      for i in range(len(cityName)):  
      CityCluster[label[i]].append(cityName[i])  
      for i in range(len(CityCluster)):  
      print("Expenses: %.2f" %expenses[i])  
      print(CityCluster[i])

### 拓展与改进

计算两条数据相似性时，Sklearn 的 K-Means 默认用的是欧式距离，虽然还有余弦相似度，马氏 距离等多种方法，但没有设定计算距离方法的参数。

如果想要使用自定义的计算距离方法，可以使用 scipy.spatial.distance.cdist(), 譬如 scipy.spatial.distance.cdist(A,B,metric='cosine') 就是使用余弦距离。

# DBSCAN

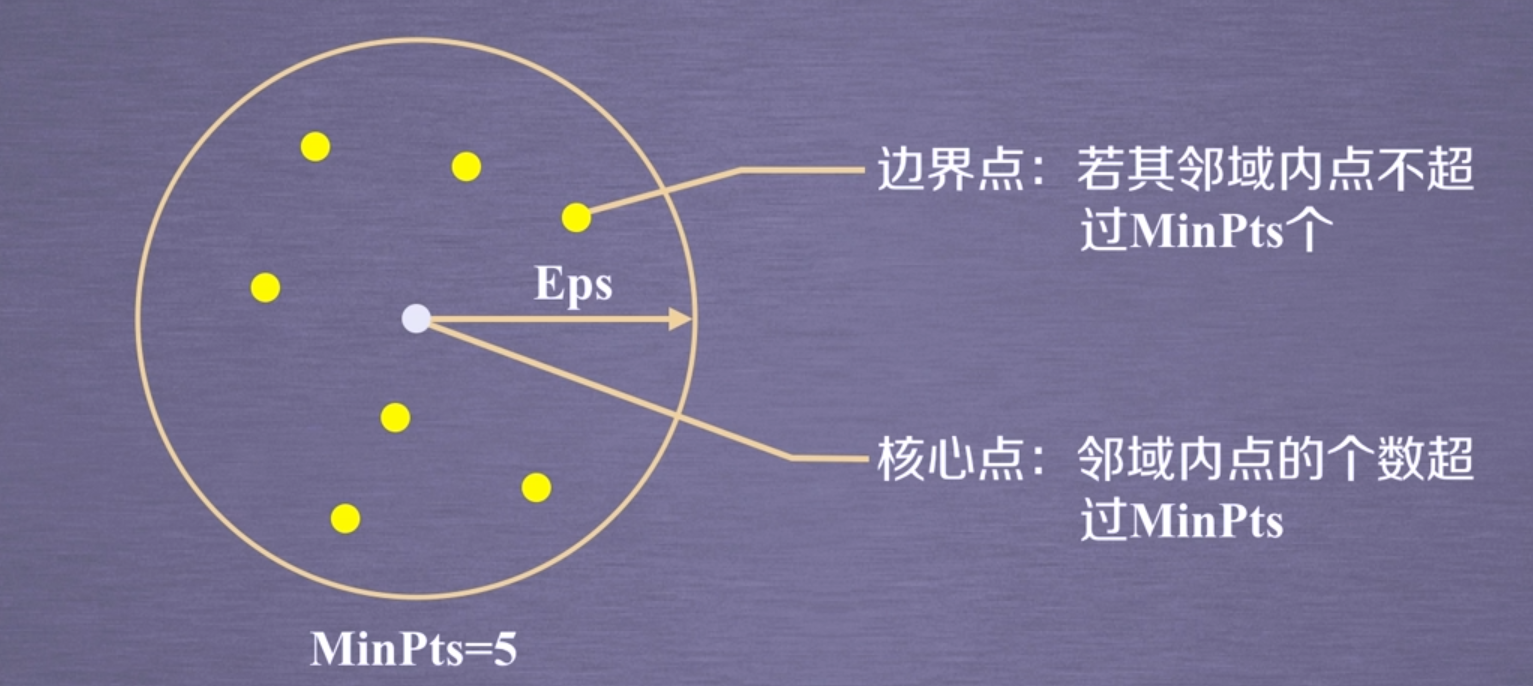
## 算法介绍

### 原理

DBSCAN 算法是一种基于密度的聚类算法：

* 聚类时候不需要指定簇的个数
* 最终簇的个数也不确定

DBSCAN 算法将数据点分为三类：



* 核心点：在半径 EPS 内含有超过 MinPts 数目的点
* 边界点：在半径 EPS 内数量小于 MinPts, 但是落在核心点的领域内
* 噪音点：既不是核心点也不是边界点的点

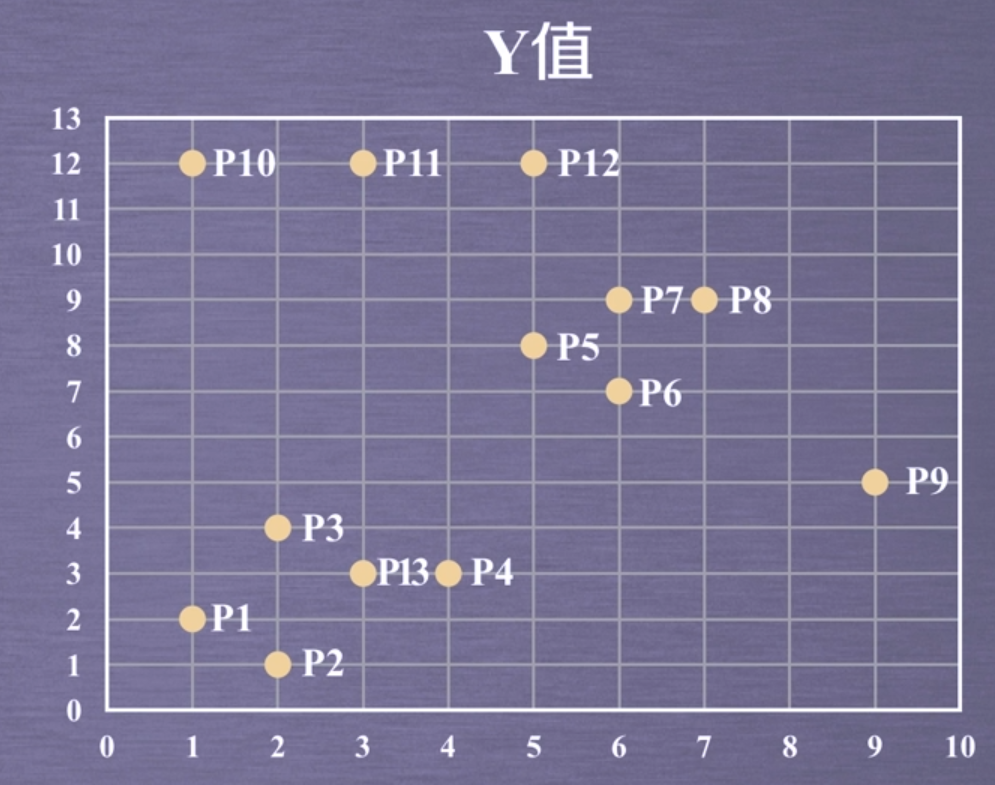
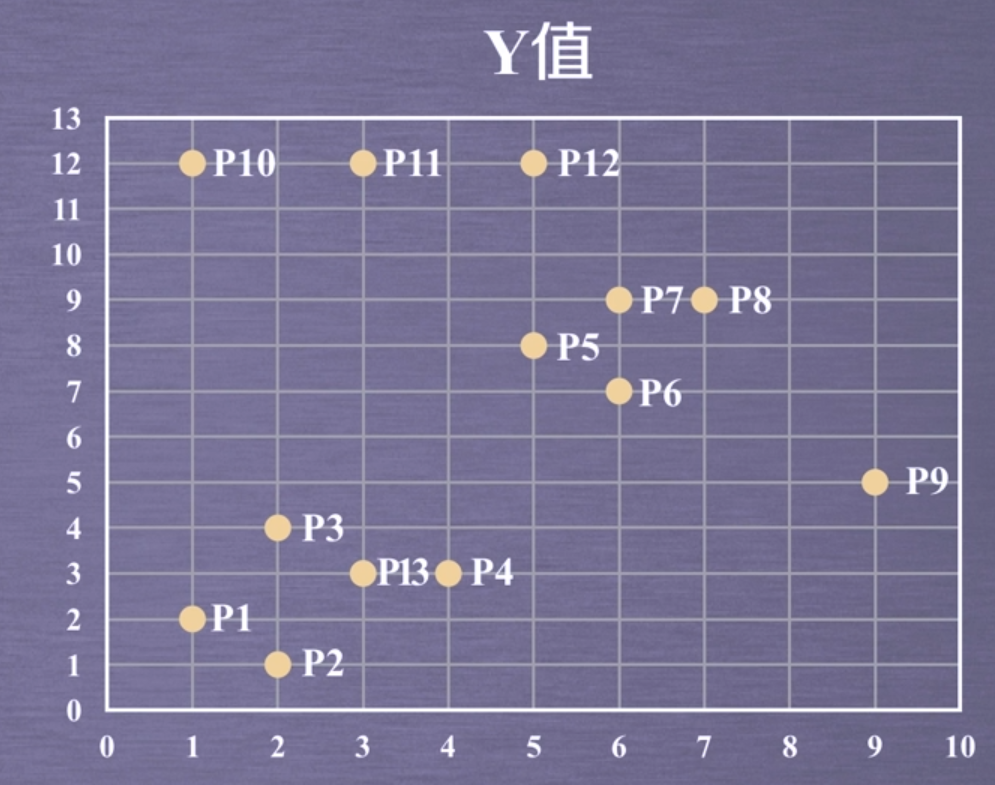
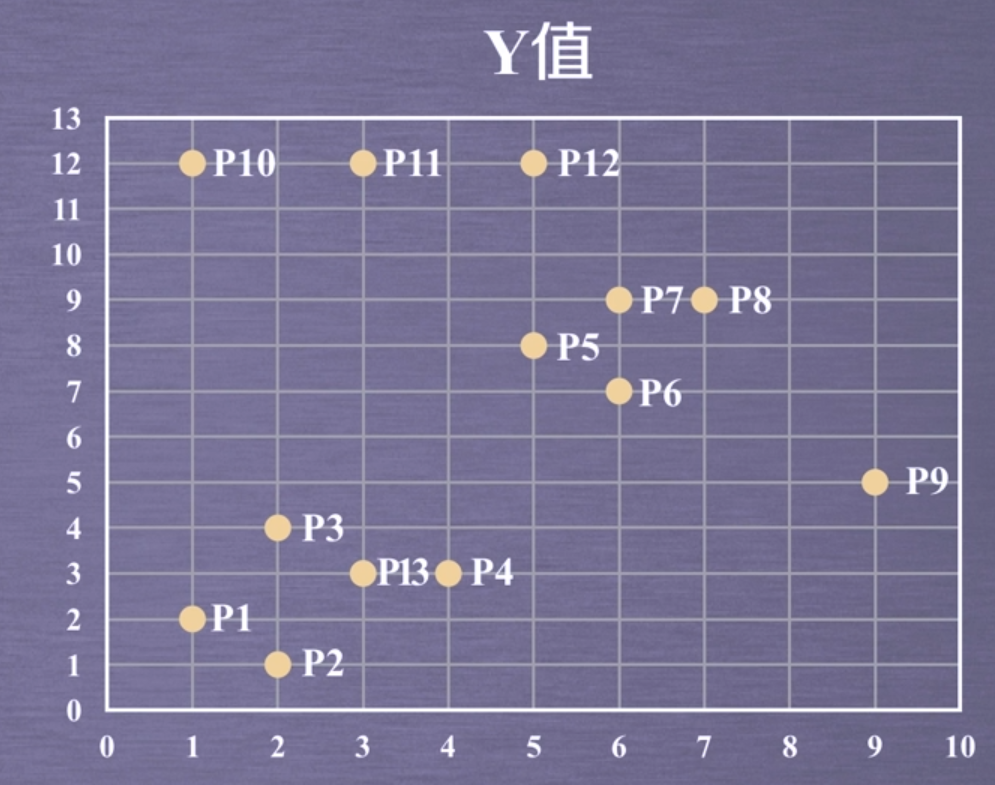
### DBSCAN 算法流程

* 将所有点标记为核心点、边界点和噪声点
* 删除噪声点
* 为距离在 EPS 之内的所有核心点之间赋予一条边
* 每组联通的核心点形成一个簇
* 将每个边界点指派到一个与之关联的核心点的簇中(哪一个核心点的半径范围内)

1. 举例

* 有如下 13 个样本点，使用 DBSCAN 进行聚类

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | * P1 | * P2 | * P3 | * P4 | * P5 | * P6 | * P7 | * P8 | * P9 | * P10 | * P11 | * P12 | * P13 |
| * X | * 1 | * 2 | * 2 | * 4 | * 5 | * 6 | * 6 | * 7 | * 9 | * 1 | * 3 | * 5 | * 3 |
| * Y | * 2 | * 1 | * 4 | * 3 | * 8 | * 7 | * 9 | * 9 | * 5 | * 12 | * 12 | * 12 | * 3 |

* 取 EPS=3，MinPts=3，根据 DBSCAN 对所有点进行聚类：
  + 对每个点计算其领域 Eps=3 内的点的集合
  + 
  + 集合内点的个数超过 MinPts=3 的点为核心点
  + 查看剩余点是否在核心点领域内，如果在，则为边界点，否则为噪声点
  + 
  + 将距离不超过 EPS=3 的点互相连接，构成一个簇，核心点领域内的点也会被加入到这个簇中
  + 

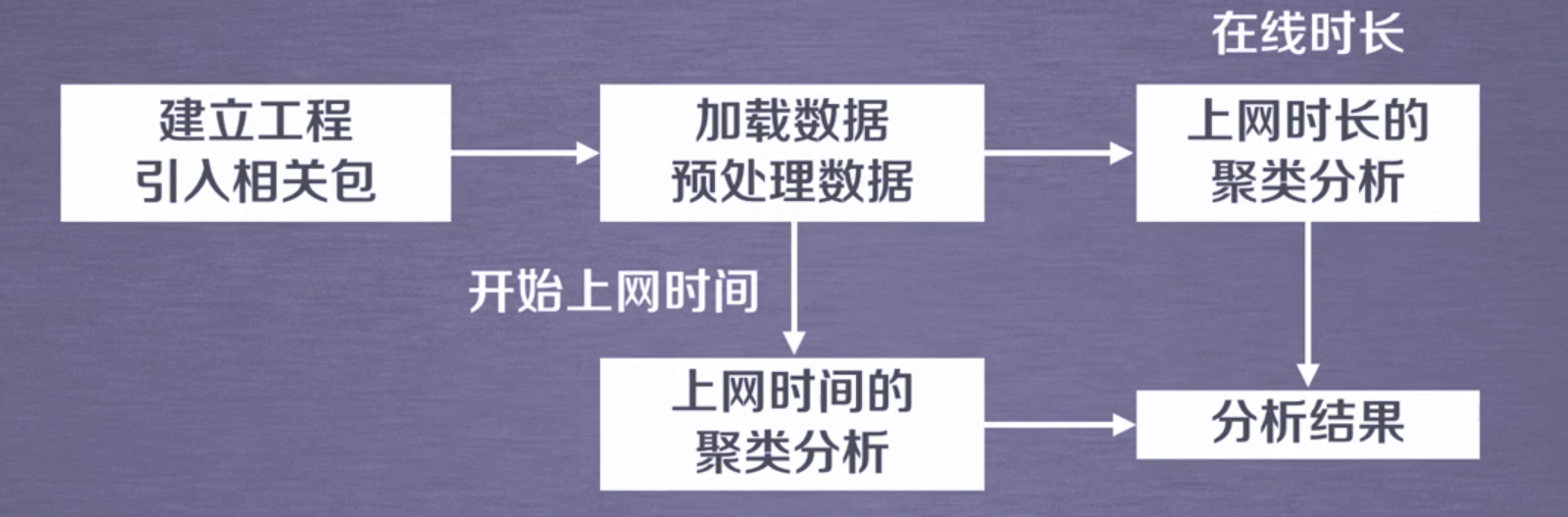
## 大学生上网使用情况调查

### 数据介绍

现有大学生校园网的日志数据，290 条大学生的校园网使用情况数据，数据包括用户 ID, 设备的 MAC 地址，IP 地址， 开始上网时间，停止上网时间，上网时长，校园网套餐等。

* 实验目的：利用已有数据，分析学生上网时间和上网时长模式
* 技术路线：sklearn.cluster.DBSCAN

### 实验过程



### 具体流程

1. 建立工程，导入 sklearn 相关包

* import numpy as np  
  from sklearn import DBSCAN
* 其中， DBSCAN 主要参数有：
  + 'eps': 两个样本被看作邻居节点的最大距离
  + minsamples: 簇的样本数
  + metric: 表示距离计算方式
  + 譬如：
  + sklearn.cluster.DBSCAN(eps=0.5,min\_samples=5,metric='euclidean')

1. 读入数据并进行处理

* import numpy as np  
  import sklearn.cluster as skc  
  from sklearn import metrics  
  import matplotlib.pyplot as plt  
    
  mac2id = dict()  
  onlinetimes = []  
  f = open('TestData.txt')  
  for line in f:  
   mac = line.split(',')[2]  
   onlinetime=int(line.split(',')[6])  
   starttime=int(line.split(',')[4].split(' ')[1].split(':')[0])  
   if mac not in mac2id:  
   mac2id[mac] = len(onlinetimes)  
   onlinetimes.append((starttime,onlinetime))  
   else:  
   onlinetimes[mac2id[mac]] = [(starttime,onlinetime)]  
  real\_X = np.array(onlinetimes).reshape((-1,2))

1. 上网时间聚类，创建 DBSCAN 算法实例，并进行训练，获得标签

* X = real\_X[:, 0:1]  
    
  db = skc.DBSCAN(eps=0.01, min\_samples=20).fit(X)  
  labels = db.labels\_  
    
  print("Labels:"+Labels)  
    
  ratio = len(labels[labels[:] == -1])/len(labels)  
  print("Noise ratio: %.2f"%ratio)  
    
  n\_clusters = len(set(labels)) - (1 if -1 in labels else 0)  
    
  print("Estimated number of clusters: %d" % n\_clusters\_)  
  print("Silhouette Coefficient: %.3f"%metrics.silhouette\_score(X,labels))  
    
  for i in range(n\_clusters\_):  
   print("Cluster ",i,":")  
   print(list(X[labels==i].flatten()))

1. 输出标签，查看结果
2. 画出直方图

* plt.hist(X,24)  
  plt.show()

### 注意事项

1. 轮廓系数
   * 定义样本 到同簇其他样本的平均距离为 , 越小则表明样本 越 应该被聚类到该簇，将 成为样本 的簇内不相似度
   * 定义样本 到其他簇 的所有样本的平均距离 , 称为样本 与 簇 的不相似度，定义为样本 的簇间不相似度，定义样本 的簇间不相似 度为
   * 显然， 越小， 越大则越好，因此，定义样本 的轮廓系数 (silhouette):
   * \begin{equation}  
      s(i) = \frac{b(i)-a(i)}{\max{a(i),b(i)}}  
     \end{equation}
   * 当 为 1 时，聚类效果最好，为 时，聚类效果最差