# 技术路线

TRS旅游信息推荐系统，是一个基于照片分享的旅游信息推荐系统。系统通过分析用户上传的照片，通过照片的拍摄地点，对照片聚类，从而获取景点信息；通过对用户历史行为（上传过的照片）进行分析，利用建立好的推荐模型向用户推荐景点信息，向用户展示与景点相关的照片和地理位置信息。TRS旅游信息推荐系统是一个简洁好用，用户交互良好的WEB系统。

系统总体使用Django来构建WEB系统，前端页面通过使用BootStrap来构建用户交互良好，简洁漂亮的系统页面。使用Mysql数据库存储系统信息。使用Python及其相关库(pandas, geopy, exifreader, sklearn)等进行数据分析。使用Spark进行数据分析。使用百度地图API展示景点位置信息。

# 用户管理

用户数据导入：从YFCC 100M数据集中提取10000多条用户数据，作为系统预用数据。

新用户注册：提供注册功能，实现新用户的注册。

用户登录：实现用户登录，基于用户进行个性化推荐。



1. 用户登录/注册流程图

# 照片管理

照片上传：用户上传其旅游照片。

信息提取：分析照片的EXIF信息，从中提取出照片拍摄时间，拍摄地等信息。使用geopy库将GEO信息转化为实际地址。

照片删除：用户可选择将其上传过的照片删除。

# 景点推荐

基于照片数据集，可以对其聚类得出景点数据集，在景点数据集的基础上搭建旅游景点推荐模型。

## 数据预处理

我们使用YFCC 100M数据集作为原始照片数据集。YFCC 100M数据库是2014年来基于雅虎Flickr的影像数据库。该库由1亿条产生于2004年至2014年间的多条媒体数据组成，其中包含了9920万的照片数据以及80万条视频数据。数据集并不包含照片或视频数据，而是一个文本数据文档，文档中每一行都是一条照片或视频的元数据。

数据预处理步骤如下：

1) 从原始照片数据集中提取具有GEO信息的照片信息，GEO信息为照片拍摄地的经纬度信息。

2) 可以根据经纬度获取坐标点的实际地址，再通过分析实际地址后判断此坐标点是否位于中国。

3) 可以使用geopy包来进行经纬度到实际地址的转换。但geopy需联网使用，在处理大量数据时非常耗时。所以可以先根据中国的经纬度范围大致筛选，然后使用geopy进行精确筛选。

## 景点聚类

使用DBSCAN算法，对照片信息聚类，获取景点信息。

DBSCAN是一种基于密度的聚类算法，这类密度聚类算法一般假定类别可以通过样本分布的紧密程度决定。同一类别的样本，他们之间的紧密相连的，也就是说，在该类别任意样本周围不远处一定有同类别的样本存在。

通过将紧密相连的样本划为一类，这样就得到了一个聚类类别。通过将所有各组紧密相连的样本划为各个不同的类别，则我们就得到了最终的所有聚类类别结果。

### DBSCAN密度定义

DBSCAN(Density-Based Spatial Clustering of Applications with Noise，具有噪声的基于密度的聚类方法)是一种很典型的密度聚类算法，和K-Means，BIRCH这些一般只适用于凸样本集的聚类相比，DBSCAN既可以适用于凸样本集，也可以适用于非凸样本集。

DBSCAN是基于一组邻域来描述样本集的紧密程度的，参数(ϵ， MinPts)用来描述邻域的样本分布紧密程度。其中，ϵ描述了某一样本的邻域距离阈值，MinPts描述了某一样本的距离为ϵ的邻域中样本个数的阈值。

假设样本集是，则DBSCAN具体的密度描述定义如下：

1） ϵ-邻域：对于，其ϵ-邻域包含样本集D中与的距离不大于ϵ的子样本集，即， 这个子样本集的个数记为

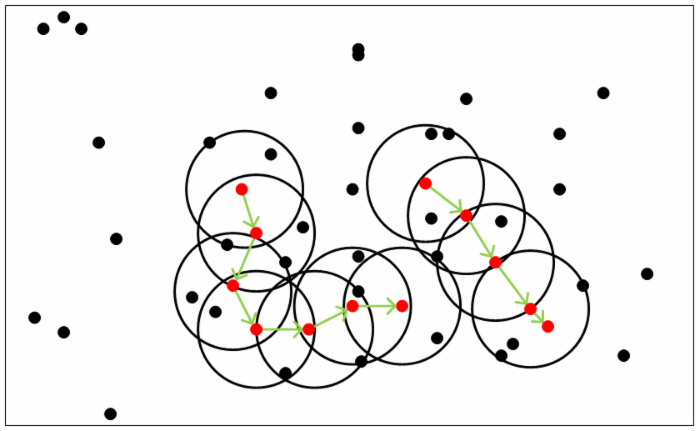
2) 核心对象：对于任一样本 ，如果其ϵ-邻域对应的 至少包含MinPts个样本，即如果，则是核心对象。

3）密度直达：如果位于的ϵ-邻域中，且是核心对象，则称由密度直达。注意反之不一定成立，即此时不能说由密度直达， 除非且也是核心对象。

4）密度可达：对于和，如果存在样本样本序列，满足 ，且由密度直达，则称由密度可达。也就是说，密度可达满足传递性。此时序列中的传递样本均为核心对象，因为只有核心对象才能使其他样本密度直达。注意密度可达也不满足对称性，这个可以由密度直达的不对称性得出。

5）密度相连：对于和，如果存在核心对象样本，使和均由密度可达，则称和密度相连。注意密度相连关系是满足对称性的。

从下图可以很容易看出理解上述定义，图中MinPts=5，红色的点都是核心对象，因为其ϵ-邻域至少有5个样本。黑色的样本是非核心对象。所有核心对象密度直达的样本在以红色核心对象为中心的超球体内，如果不在超球体内，则不能密度直达。图中用绿色箭头连起来的核心对象组成了密度可达的样本序列。在这些密度可达的样本序列的ϵ-邻域内所有的样本相互都是密度相连的。



1. DBSCAN效果说明

### DBSCAN聚类算法

DBSCAN的聚类定义很简单：由密度可达关系导出的最大密度相连的样本集合，即为我们最终聚类的一个类别，或者说一个簇。

这个DBSCAN的簇里面可以有一个或者多个核心对象。如果只有一个核心对象，则簇里其他的非核心对象样本都在这个核心对象的[Math Processing Error]-邻域里；如果有多个核心对象，则簇里的任意一个核心对象的[Math Processing Error]-邻域中一定有一个其他的核心对象，否则这两个核心对象无法密度可达。这些核心对象的[Math Processing Error]-邻域里所有的样本的集合组成的一个DBSCAN聚类簇。

那么怎么才能找到这样的簇样本集合呢？DBSCAN使用的方法很简单，它任意选择一个没有类别的核心对象作为种子，然后找到所有这个核心对象能够密度可达的样本集合，即为一个聚类簇。接着继续选择另一个没有类别的核心对象去寻找密度可达的样本集合，这样就得到另一个聚类簇。一直运行到所有核心对象都有类别为止。

|  |  |
| --- | --- |
| **算法1：DBSCAN聚类算法** | |
| **输入** | 样本集，邻域参数(ϵ,MinPts), 样本距离度量方式 |
| **输出** | 簇划分C |
| **1** | 初始化核心对象集合, 初始化聚类簇数k=0，初始化未访问样本集合, 簇划分 |
| **2** | 对于j=1,2,...m, 按下面的步骤找出所有的核心对象：   * 1. 通过距离度量方式，找到样本的ϵ-邻域子样本集   2. 如果子样本集样本个数满足， 将样本加入核心对象样本集合： |
| **3** | 如果核心对象集合，则算法结束，否则转入步骤4. |
| **4** | 在核心对象集合Ω中，随机选择一个核心对象o，初始化当前簇核心对象队列, 初始化类别序号k=k+1，初始化当前簇样本集合, 更新未访问样本集合 |
| **5** | 如果当前簇核心对象队列，则当前聚类簇生成完毕, 更新簇划分, 更新核心对象集合， 转入步骤3。 |
| **6** | 在当前簇核心对象队列中取出一个核心对象o′,通过邻域距离阈值ϵ找出所有的ϵ-邻域子样本集，令 更新当前簇样本集合, 更新未访问样本集合 更新，转入步骤5. |
| **7** | 输出结果： 簇划分 |

和传统的K-Means算法相比，DBSCAN最大的不同就是不需要输入类别数k，当然它最大的优势是可以发现任意形状的聚类簇，而不是像K-Means，一般仅仅使用于凸的样本集聚类。同时它在聚类的同时还可以找出异常点，这点和BIRCH算法类似。

以位于北京的38293张照片数据为例，不同参数下DBSCAN聚类效果如下表。

1. 不同参数下DBSCAN聚类效果

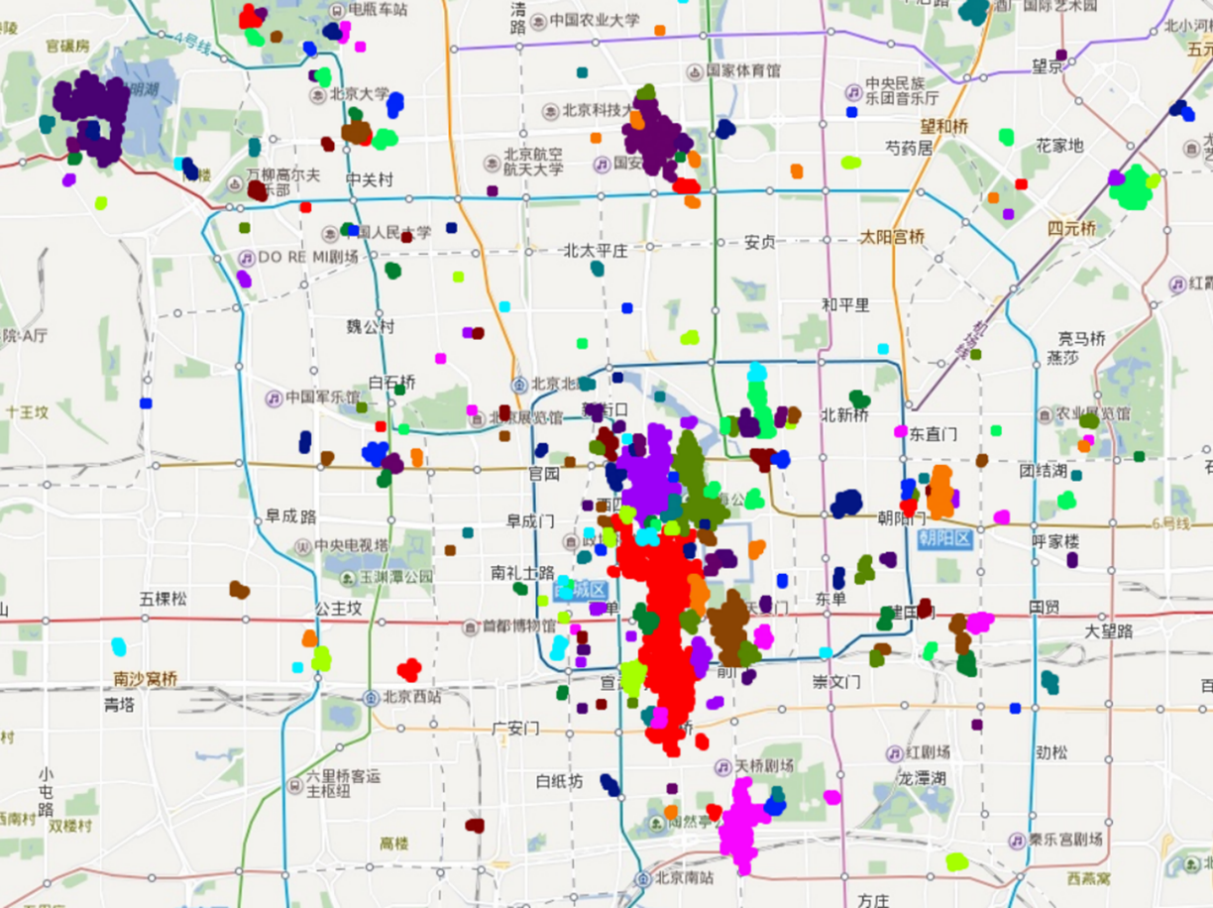
|  |  |  |  |
| --- | --- | --- | --- |
| ϵ | minPts | 簇数 | 噪点率 |
| 0.0005 | 15 | 422 | 37.23 |
| 0.0007 | 15 | 379 | 31.45 |
| 0.001 | 15 | 330 | 26.16 |
| 0.002 | 15 | 246 | 15.31 |
| 0.005 | 15 | 120 | 7.01 |

ϵ=0.001， minPts=15时，每个省份的聚类效果如下表。

1. 不同省份的聚类效果

| 省份 | 数据量 | 簇数 | 噪点率 | 省份 | 总数据 | 簇数 | 噪点率 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 澳门特别行政区 | 5955 | 38 | 11.89 | 贵州省 | 944 | 14 | 51.38 |
| 香港特别行政区 | 65943 | 380 | 19.47 | 浙江省 | 5890 | 80 | 51.65 |
| 台湾省 | 272537 | 2463 | 21.32 | 安徽省 | 1752 | 21 | 53.65 |
| 上海市 | 31871 | 259 | 22.11 | 重庆市 | 2358 | 28 | 53.86 |
| 北京市 | 38293 | 330 | 26.16 | 辽宁省 | 1998 | 16 | 54.1 |
| 陕西省 | 5410 | 78 | 31.99 | 黑龙江省 | 884 | 14 | 54.98 |
| 海南省 | 1361 | 14 | 35.05 | 四川省 | 5957 | 59 | 55.08 |
| 江西省 | 1039 | 19 | 40.04 | 湖北省 | 2139 | 26 | 56.9 |
| 天津市 | 1607 | 21 | 41.57 | 内蒙古自治区 | 1081 | 13 | 59.2 |
| 福建省 | 3088 | 45 | 44.07 | 河南省 | 967 | 14 | 60.29 |
| 广东省 | 12762 | 150 | 47.02 | 湖南省 | 1520 | 19 | 62.57 |
| 西藏自治区 | 4553 | 51 | 47.22 | 山东省 | 2467 | 24 | 62.83 |
| 江苏省 | 6512 | 81 | 47.71 | 甘肃省 | 1217 | 15 | 66.64 |
| 吉林省 | 768 | 9 | 47.92 | 青海省 | 746 | 9 | 70.24 |
| 山西省 | 1323 | 24 | 49.06 | 新疆维吾尔自治区 | 1987 | 20 | 70.86 |
| 云南省 | 6332 | 78 | 49.94 | 河北省 | 1344 | 12 | 74.03 |
| 广西壮族自治区 | 4161 | 56 | 50.47 | 宁夏回族自治区 | 46 | 0 | 100 |

ϵ=0.001， minPts=15时，聚类效果在百度地图上的投影效果如下图。



1. 聚类效果在百度地图上的投影效果图

一般来说，如果数据集是稠密的，并且数据集不是凸的，那么用DBSCAN会比K-Means聚类效果好很多。如果数据集不是稠密的，则不推荐用DBSCAN来聚类。

DBSCAN的主要优点有：

1）可以对任意形状的稠密数据集进行聚类，相对的，K-Means之类的聚类算法一般只适用于凸数据集。

2）可以在聚类的同时发现异常点，对数据集中的异常点不敏感。

3）聚类结果没有偏倚，相对的，K-Means之类的聚类算法初始值对聚类结果有很大影响。

DBSCAN的主要缺点有：

1）如果样本集的密度不均匀、聚类间距差相差很大时，聚类质量较差，这时用DBSCAN聚类一般不适合。

2）如果样本集较大时，聚类收敛时间较长，此时可以对搜索最近邻时建立的KD树或者球树进行规模限制来改进。

3）调参相对于传统的K-Means之类的聚类算法稍复杂，主要需要对距离阈值[Math Processing Error]，邻域样本数阈值MinPts联合调参，不同的参数组合对最后的聚类效果有较大影响。

## 推荐模型

构建推荐模型，使用基于物品的协同过滤算法，ALS 矩阵分解算法等，使用投票法则整合算法。

### 推荐算法概述

推荐算法是非常古老的，在机器学习还没有兴起的时候就有需求和应用了。概括来说，可以分为以下5种：

1）基于内容的推荐：这一类一般依赖于自然语言处理NLP的一些知识，通过挖掘文本的TF-IDF特征向量，来得到用户的偏好，进而做推荐。这类推荐算法可以找到用户独特的小众喜好，而且还有较好的解释性。这一类由于需要NLP的基础，本文就不多讲，在后面专门讲NLP的时候再讨论。

2）协调过滤推荐：本文后面要专门讲的内容。协调过滤是推荐算法中目前最主流的种类，花样繁多，在工业界已经有了很多广泛的应用。它的优点是不需要太多特定领域的知识，可以通过基于统计的机器学习算法来得到较好的推荐效果。最大的优点是工程上容易实现，可以方便应用到产品中。目前绝大多数实际应用的推荐算法都是协同过滤推荐算法。

3）混合推荐：这个类似我们机器学习中的集成学习，博才众长，通过多个推荐算法的结合，得到一个更好的推荐算法，起到三个臭皮匠顶一个诸葛亮的作用。比如通过建立多个推荐算法的模型，最后用投票法决定最终的推荐结果。混合推荐理论上不会比单一任何一种推荐算法差，但是使用混合推荐，算法复杂度就提高了，在实际应用中有使用，但是并没有单一的协调过滤推荐算法，比如逻辑回归之类的二分类推荐算法广泛。

4）基于规则的推荐：这类算法常见的比如基于最多用户点击，最多用户浏览等，属于大众型的推荐方法，在目前的大数据时代并不主流。

5）基于人口统计信息的推荐：这一类是最简单的推荐算法了，它只是简单的根据系统用户的基本信息发现用户的相关程度，然后进行推荐，目前在大型系统中已经较少使用。

### 协同过滤

协同过滤(Collaborative Filtering)作为推荐算法中最经典的类型，包括在线的协同和离线的过滤两部分。所谓在线协同，就是通过在线数据找到用户可能喜欢的物品，而离线过滤，则是过滤掉一些不值得推荐的数据，比比如推荐值评分低的数据，或者虽然推荐值高但是用户已经购买的数据。

协同过滤的模型一般为m个物品，m个用户的数据，只有部分用户和部分数据之间是有评分数据的，其它部分评分是空白，此时我们要用已有的部分稀疏数据来预测那些空白的物品和数据之间的评分关系，找到最高评分的物品推荐给用户。

一般来说，协同过滤推荐分为三种类型。第一种是基于用户(user-based)的协同过滤，第二种是基于项目(item-based)的协同过滤，第三种是基于模型(model based)的协同过滤。

基于用户(user-based)的协同过滤主要考虑的是用户和用户之间的相似度，只要找出相似用户喜欢的物品，并预测目标用户对对应物品的评分，就可以找到评分最高的若干个物品推荐给用户。而基于项目(item-based)的协同过滤和基于用户的协同过滤类似，只不过这时我们转向找到物品和物品之间的相似度，只有找到了目标用户对某些物品的评分，那么我们就可以对相似度高的类似物品进行预测，将评分最高的若干个相似物品推荐给用户。比如你在网上买了一本机器学习相关的书，网站马上会推荐一堆机器学习，大数据相关的书给你，这里就明显用到了基于项目的协同过滤思想。

我们可以简单比较下基于用户的协同过滤和基于项目的协同过滤：基于用户的协同过滤需要在线找用户和用户之间的相似度关系，计算复杂度肯定会比基于基于项目的协同过滤高。但是可以帮助用户找到新类别的有惊喜的物品。而基于项目的协同过滤，由于考虑的物品的相似性一段时间不会改变，因此可以很容易的离线计算，准确度一般也可以接受，但是推荐的多样性来说，就很难带给用户惊喜了。一般对于小型的推荐系统来说，基于项目的协同过滤肯定是主流。但是如果是大型的推荐系统来说，则可以考虑基于用户的协同过滤，当然更加可以考虑我们的第三种类型，基于模型的协同过滤。

### ALS矩阵分解

用矩阵分解做协同过滤是目前使用也很广泛的一种方法。由于传统的奇异值分解SVD要求矩阵不能有缺失数据，必须是稠密的，而我们的用户物品评分矩阵是一个很典型的稀疏矩阵，直接使用传统的SVD到协同过滤是比较复杂的。

目前主流的矩阵分解推荐算法主要是SVD的一些变种，比如FunkSVD，BiasSVD和SVD++。这些算法和传统SVD的最大区别是不再要求将矩阵分解为的形式，而变是两个低秩矩阵的乘积形式。

协同过滤提出了一种支持不完整评分矩阵的矩阵分解方法,不用对评分矩阵进行估值填充,有很好的推荐精度。Spark MLlib中实现的基于ALS（最小交替二乘）矩阵分解协同过滤算法。

用户对物品的评分行为可以表示成一个评分矩阵，表示m个用户对n各物品的评分情况。其中，表示用户对物品的评分。但是，用户不会对所有物品评分，所以这个矩A很多元素都是空的，是一个稀疏矩阵。协同过滤提出了一种支持不完整评分矩阵的矩阵分解方法,不用对评分矩阵进行估值填充。在推荐系统中，我们希望得到用户对所有物品的评分情况，如果用户没有对一个物品评分，那么就需要预测用户是否会对该物品评分，以及会打多少分。这就是所谓的“矩阵补全”。

ALS 的核心假设是：评分矩阵A是近似低秩的，即一个评分矩阵 可以用两个小矩阵和的乘积来近似。

我们把评分理解成相似度，那么评分矩阵就可以由用户喜好特征矩阵和产品特征矩阵的乘积。

我们使用用户喜好特征矩阵中的第i个用户的特征向量，和产品特征矩阵第j个产品的特征向量来预测评分矩阵中的。我们可以得出一下的矩阵分解模型的损失函数为：

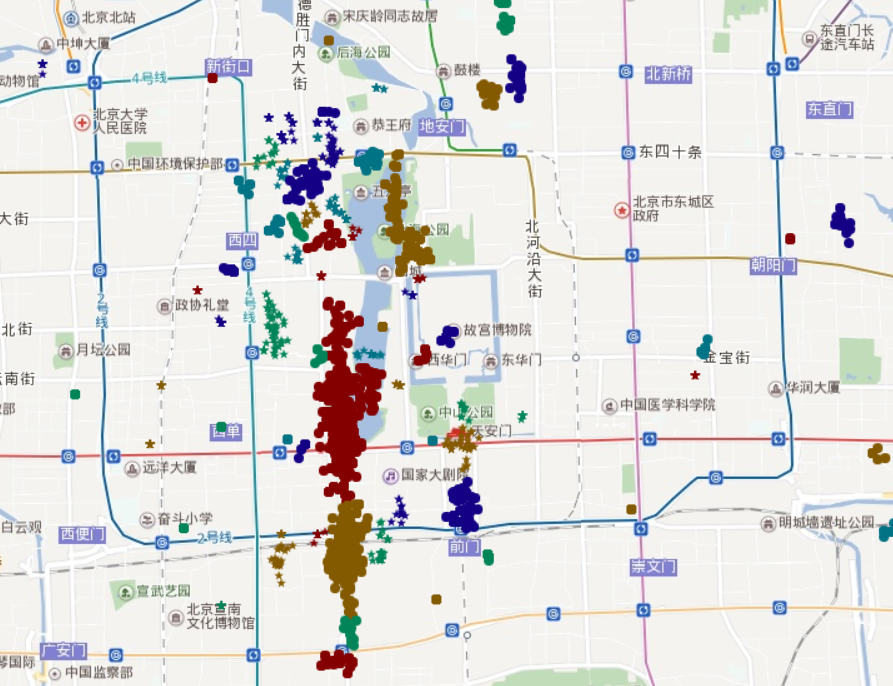
有了损失函数之后，下面就开始介绍优化方法。通常的优化方法分为两种：交叉最小二乘法（alternative least squares）和随机梯度下降法（stochastic gradient descent）。Spark使用的是交叉最小二乘法（ALS）来最优化损失函数。算法的思想就是：我们先随机生成然后固定它求解，再固定求解，这样交替进行下去，直到取得最优解min(C)。因为每步迭代都会降低误差，并且误差是有下界的，所以 ALS 一定会收敛。但由于问题是非凸的，ALS 并不保证会收敛到全局最优解。但在实际应用中，ALS 对初始点不是很敏感，是否全局最优解造成的影响并不大。

|  |  |
| --- | --- |
| **算法3：ALS矩阵分解算法** | |
| **输入** | 评分矩阵 ，最大迭代次数K，均方根误差阈值a，正则化系数lambda |
| **输出** | 用户喜好特征矩阵，产品特征矩阵 |
| **1** | 随机初始化, |
| **2** | 循环直至收敛或到达最大迭代次数 |
| **3** | 固定，求偏导，得到更新公式，更新 |
| **4** | 固定，求偏导，得到更新公式，更新 |
| **5** | 返回， |

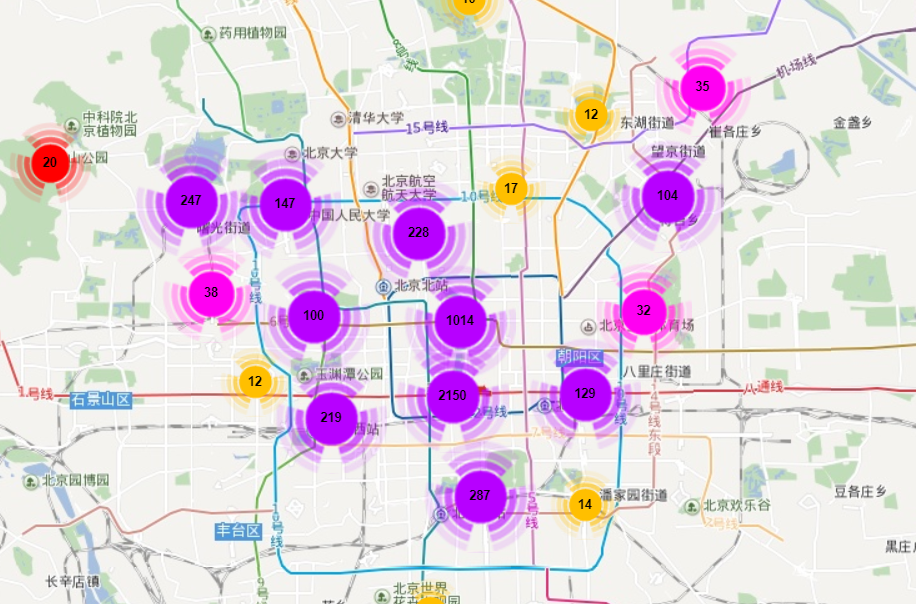
# 地图LBS

利用百度地图提供的API接口，结合景点中每张照片的GEO信息，可以很方便的在实际地图上投影景点信息。

值得注意的是，照片的GEO信息是基于GPS系统的，GPS采用的是WGS坐标系，是国际通用的一种地心坐标系，而百度地图的坐标系采用的是BD-09，是 GCJ-02的变种， GCJ-02是国内官方采用的一种坐标系。



1. 利用百度地图显示景点聚类效果



1. 利用百度地图显示景点聚类效果