Statistics and Application 统计学与应用, 2019, 8(2), 203-217
Published Online April 2019 in Hans. http://www.hanspub.org/journal/sa
https://doi.org/10.12677/sa.2019.82023

Hans汉斯

Research on the Division of the Take-Out Order Region Based on Improved Weighted K-Means Clustering

Yunqing Ma, Chuanxin Zhang

Shanghai University of International Business and Economics, Shanghai
Email: 506143247@qq.com

Received: Mar. 6th, 2019; accepted: Mar. 21st, 2019; published: Mar. 29th, 2019

Abstract

The take-away industry is booming and people's demands for the quality of take-away services are constantly improving. Since the take-out area of the take-out is basically fixed, it is the key to improve efficiency by reasonably dividing the area of the take-out of the seller and assigning the number of the sellers in each area. Based on the query algorithm model, this project analyzes the take-out data of Shanghai during a certain period of time in 2017, trying to obtain a more reasonable division standard for the take-out order area and give the division. K-Means is a common partitioning clustering algorithm based on the inability of centralized system frameworks to process and analyze massive amounts of data. However, the K-Means clustering algorithm cannot be used for the two-dimensional point set with weight information. Therefore, it is especially necessary to study an improved Weighted K-Means algorithm. This project defines the weighted centroid and weighted distance, proposes a new Weighted K-Means algorithm, and uses the two methods before and after the improvement to deal with the take-out information of Shanghai take-out orders, and gives a reasonable and feasible division of the take-out area of the take-out. Comparing the results of the two methods, the improved Weighted K-Means is not only feasible, but also better in regional division. At the same time, using this method to make a new division of the take-out order area helps to optimize the existing take-away model, improve the take-out efficiency and customer satisfaction.

Keywords

Clustering Algorithm, K-Means Algorithm, Weighted K-Means Algorithm, Python

基于改进的Weighted K-Means聚类的外卖员接单区域划分问题研究

马云卿，张传鑫

文章引用: 马云卿, 张传鑫. 基于改进的 Weighted K-Means 聚类的外卖员接单区域划分问题研究[J]. 统计学与应用, 2019, 8(2): 203-217. DOI: 10.12677/sa.2019.82023

上海对外经贸大学，上海
Email: 506143247@qq.com

收稿日期：2019年3月6日；录用日期：2019年3月21日；发布日期：2019年3月29日

摘　要

外卖行业蓬勃发展，人们对于外卖服务质量的要求也不断提升。因外卖员接单区域基本固定，合理划分外卖员负责区域并分配每个区域外卖员人数成为提升效率的关键。本项目基于查询算法模型，分析上海市2017年某时段的外卖数据，试图得到一个对于外卖接单区域的较为合理的划分标准并给出该划分。K-Means是一种常见的划分聚类算法，是在集中式系统框架无法对海量数据进行处理分析的基础上提出的。然而对于有权重的二维点集无法使用K-Means聚类算法，因此研究一种改进的Weighted K-Means算法显得尤为必要。本项目定义带权质心和带权距离，提出了新的Weighted K-Means算法，并使用改进前后的两种方法处理上海市外卖接单信息，给出合理可行的外卖员接单区域划分。对比两种方法的结果，改进的Weighted K-Means不仅方法可行，区域划分表现也更优秀。与此同时，使用该方法对外卖接单区域进行新的划分，有助于优化现有外卖模式、提升外卖效率以及顾客满意度。

关键词

聚类算法，K-Means算法，Weighted K-Means算法，Python

Copyright © 2019 by authors and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

1.1. 课题缘起

打包形式是最早出现的外卖形式，虽然古老，却延续至今。随着电话、手机、网络的普及，外卖行业得到迅速的发展。外卖形式经历了从餐厅打包、电话订购、网站订购，到微信手机端订购的演变。

在线订餐模式最早在美国兴起，在美国兴起当然有它独特的文化优势，比如国外的人更喜欢提前预定用餐。目前国内的在线订餐还是处于混乱的诸侯混战状态，各大网站各自占据着自己的一方水土，但是随着马太效应越来越明显，在线订餐市场也在进行一轮新的洗牌，拥有充足资金的订餐网站正在迅速地扩大各大城市分站，占领市场份额。

国内方面，2008 年下半年起，在扩大内需的大环境下，商务部提出在扩大内需、拉动消费方面，将大力发展餐饮业。以消费促发展，在三方面大力发展餐饮行业。首先，大力发展大众化餐饮。其次，重点关注和解决餐饮行业的放心消费问题，严把食品原料进货关。第三，积极推进节能环保工作，推动行业节能减排纵深发展。

中国餐饮业正积极进军海外市场。中国的餐饮市场中，正餐以中式正餐为主，西式正餐逐渐兴起，肯德基、麦当劳、必胜客等，是市场中的主力，中式快餐已经蓬勃发展，但当前尚无法与“洋快餐”相抗衡。随着中国经济及旅游业的发展，餐饮行业的前景看好，在未来几年内，中国餐饮业经营模式将多元化发展，国际化进程将加快，而且绿色餐饮必将成为时尚。中国人每年花 1 万亿用于吃盒饭，超过 3

亿中国人的午餐是吃盒饭快餐。发展速度很惊人。

根据艾瑞咨询的数据显示：2015 年，我国网民规模约 6.88 亿人，其中大约 50%的网民使用过 O2O 外卖平台订餐；2015 年，我国餐饮 O2O 市场规模为 1615.5 亿元，预计在 2018 年餐饮 O2O 市场将突破 2897.9 亿元。面对餐饮外卖这一市场的不断扩大，越来越多的企业将工作重心放在餐饮外卖服务上，再加上如今互联网技术的发展，餐饮外卖 O2O 这一模式迅速发展，餐饮外卖 O2O 平台如雨后春笋般涌现。2013 年底，阿里巴巴推出了口碑外卖，“饿了么”公司则完成了 C 轮融资，2014 年美团推出新的外卖模块，同年 4 月百度外卖也正式上线，2018 年 3 月，滴滴外卖在南京、无锡、福州、长沙、济南、厦门、温州、成都和宁波九个城市上线。外卖 O2O 平台消费者流量大、消费频率高，弥补了学校和企业食堂的局限性、为消费者提供了便利，因此发展得极为迅速[1]。

2017 年，外卖行业市场规模呈现持续增长态势，交易额近千亿元人民币。随之而来的，是人们对于外卖服务质量不断提升的要求，而其中最为重要的，就是外卖送达速度。然而，商家制作外卖菜品往往时间相对固定，外卖员花在路上的时间也基本统一，因而提升外卖效率，最重要的就是提升外卖员接单效率。

1.2. 问题提出

一般的外卖流程如下：

1) 接听外卖员电话(或客户端接收外卖下单信息)；
2) 准确规范记录信息；
3) 厨房下单并加工制作；
4) 打包出品等待送餐员；
5) 送餐员取餐并送往客户指定地点；
6) 按时到达指定地点并交付；
7) 客户接到外卖，核对确认无误后结束外卖流程。

为减少从商家接到订单到外卖员开始送餐这段时间，并提供更高质量高速度的服务，配送单位往往会两个措施并举，一方面利用旅行商方法设计最优配送路线以节约配送时间，一方面对于外卖员接单区域进行划分。前者显而易见能促使配送时间更短，而后一种方案被采用则是为了避免出现同一时间在一个区域“人多单少”，另一个区域“单多人少”的情况发生。而利用旅行商方法设计最优配送路线，以节约配送时间已有较完善的理论基础和算法实现，故在本文中不作考虑[2]。

因此，外卖员接单区域往往被固定在某个特定区域，即外卖员只可以在自己所属区域接单，不可以超出范围接单(但可以送至区域外的客户处)。由此，问题便被转化，合理划分外卖员负责的区域并合理分配每个区域外卖员的人数成为提升效率的关键。

本项目试图使用艺中区域划分的方法，达到以下的结果：

1) 被划分的区域不会重叠，没有相互之间覆盖。
2) 同一区域之间的个体相对集中，这样每一个快递员运送物资时，不会花过多时间在路程上。
3) 每一区域的外卖员数量大致相等。

如何进行合理而高效的区域划分，在当前还是一个可以深入研究的领域。本文尝试基于某一类查找算法，处理上海市外卖接单信息，最终给出一个合理可行的外卖员接单区域的划分，从而优化现有外卖模式、提升外卖效率以及顾客满意度。

1.3. 研究方法

近年来，数据挖掘成为越来越热门的一个研究方向，而聚类作为数据挖掘的主要方法之一，也越来

越引起人们的关注。所谓聚类，就是把大量的 d 维数据(n 个)对象聚集成 k 个聚类(k < n)，使得同一个聚类内对象的相似性尽可能最大，而不同聚类内对象的相似性尽量达到最小。由于聚类的特殊性，在外卖系统中使用聚类算法进行配送区域的划分可以使得得到的区域比较紧密，因而更符合实际情况。现有的聚类算法大致可以分为四大类：划分聚类算法、层次聚类算法、密度性聚类算法、网络型聚类算法[3]。

由于划分聚类中 K-Means 法相对易于理解，便于实现，所以本系统中采用 K-Means 聚类方法进行配送区域的划分。

首先，根据数据集中商家在一段时间内接单数以及客户评分，定义并计算商家“热度”；根据“热度”以及坐标位置，实验模拟合适的查询算法模型，对于各个点(商家)进行聚类，从而得到被划分在同一个外卖区域的各商家，从而得到一个对于外卖接单区域的较为合理的划分标准并给出该划分。

其次，根据模拟出的结果，分析每个区域安排的最优的外卖员人数。最后对于模型进行检验，检验优度并判断该模型是否适用于上海市或其他城市。

1.4. 研究现状

目前已有的研究主要集中于两个方面：物流模式、外卖模式的研究，路径规划的研究。已有研究当下外卖 O2O 模式下的线下物流模式以及其所存在的问题,并基于当下外卖电商线下物流的现状提出了具有针对性和改进性的物流策略优化方案[4]。国内外现有基于多尺度路径算法、模拟退火法、粒子群算法，蚁群算法等，黄心等人使用蚁群算法对不同地址的收货点进行路径进行规划，并利用 MATLAB 软件，为送货人员设计出了最短时间路径规划[5] (黄心，吴学群，袁清冽，2017)，暂无对于外卖接单区域划分的相关研究成果。

国内关于查找算法的研究相对成熟，大致有顺序查找、二分查找、二叉排序树查找和哈希查找等算法[6] (赵刚，李昆，2010)，使用合适的查找算法能够较快得到区域划分的局部最优解或全局最优解，从而达到研究目标。

如今外卖行业多为平台派单、外卖员接单的形式进行外卖业务[7] (王倩影，2017)；因受制于区域内商家外卖正常运作需要一定量的可接单外卖员，外卖员接单区域被平台进行划分、限制在区域内接单。

1.5. 数据来源

使用 Python 爬虫技术，从“饿了么星选”网站[8]上，爬取商家名称、商家分类、评分、接单时间、商家地址、Y 轴坐标(百度地图平面坐标)、X 轴坐标(百度地图平面坐标)、平均送达时间、起送价、配送费。共 16,844 条数据。

对数据进行初步处理，以待后续数据分析。

2. 聚类模型

2.1. 聚类算法综述

聚类分析是数据挖掘中的一个很活跃的研究领域，提出了许多聚类算法。这些算法可以被分为划分算法、层次方法、基于密度方法、基于网络方法和基于模型方法[9]。

为了满足第一个要求，即“被划分的区域不会重叠，没有相互之间覆盖”，我们必须确定每一个点对某个点群的唯一归属关系，即一个点只能属于某一个点群，而不能同时属于多个点群，这样我们就必须使用一种确切的划分聚类方法，即排他聚类。最常用的排他聚类方法是 K-Means 方法[10]。

因为第二个要求，即“同一区域之间的个体相对集中，这样每一个快递员运送物资时，不会花过多时间在路程上”，要求我们划分的区域尽可能泾渭分明。这一点没有算法上的保证，但是通常来说，

K-Means 聚类算法出事聚类中心的选择对最后的结果有很大的影响。

第三个要求，即“每一区域的外卖员数量大致相等”，是难点所在。原始的 K-Means 聚类算法中并不包含节点的“权重”概念，只是简单地根据给出点集的聚集特性来分块，因此并不会保持各个聚类中点的个数的均衡，更不是和用来对带权重的点进行聚类。但在后面我们会看到，通过对原始的 K-Means 进行改进，我们可以克服 K-Means 聚类的这种不足。

2.2. K-Means 算法

K-Means 算法首先随机地在 N 个对象中选取 k 个数，作为初始聚类中心(即把 N 个对象分为 k 个簇)，采用距离作为相似性的评价指标，认为两个对象的距离越近，其相似度就越大。相似度通过一个簇中对象的平均值来计算。然后按照最小距离原则，讲 N 个对象划分到不同的簇中。最后不断迭代计算聚类中心和调整各个对象的勒边，最终使得每个对象到其判属的聚类中心的距离的平方和最小。步骤如下：

(1) 在 N 个对象中随机选取 k 个数作为初始聚类中心，即 $c_1, c_2, \cdots, c_k$；

(2) 将 N 个对象按最小距离原则找到离它最近的聚类中心 c_i，并将其划分到 c_i 所标明的簇中；

(3) 计算每个簇中对象的均值，并且该均值作为该簇新的聚类中心，即

$$c_k = \frac{1}{n}\sum_{i=1}^{n} X_i^{(k)}, k = 1, 2, \cdots, K;$$

(4) 重复(2)~(3)步，知道没有对象或很少的对象被分配到不同的簇中。

流程图如图 1 所示。

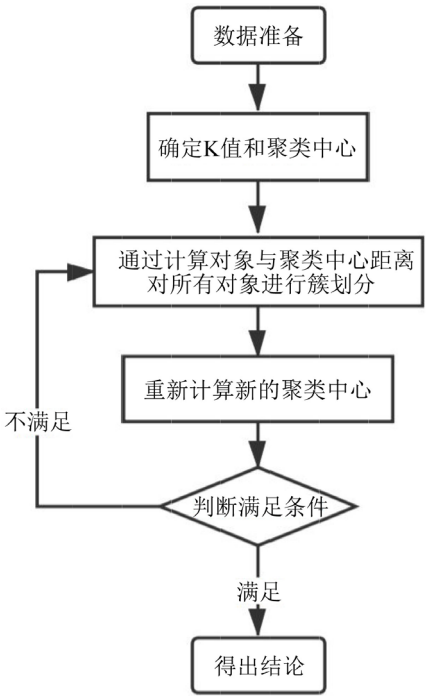

Figure 1. Algorithm flowchart of K-Means
图 1. K-Means 算法流程图

2.3. K-Means 算法的 Python 代码

K-Means 算法的 Python 代码如图 2 所示。

```
import pandas as pd #参数初始化
inputfile = '../data/waimai_data.xls' #销量及其他属性数据
outputfile = '../tmp/data_type.xls' #保存结果的文件名
k = 10 #聚类的类别
iteration = 500 #聚类最大循环次数
data = pd.read_excel(inputfile, index_col = 'Id') #读取数据
data_zs = 1.0*(data - data.mean())/data.std() #数据标准化
from sklearn.cluster import KMeans
model = KMeans(n_clusters = k, n_jobs = 4, max_iter = iteration) #分为k类，并
发数 4
model.fit(data_zs)
```

Figure 2. K-Means algorithm Python code
图 2. K-Means 算法 Python 代码

从代码可以看出，原始的 K-Means 实现中，所有点都是二维数据，即只有两个坐标向量，没有其他参考，其他的信息都被完全忽略了(节点的权重没有参与节点聚类和再聚类过程)。以下是使用原始 K-Means 聚类对于一个模拟的少量节点的聚类结果。

2.4. K-Means 算法的代码结果

取 $k = 10$，使用 Matlab 根据分类标准作图(图 3)。

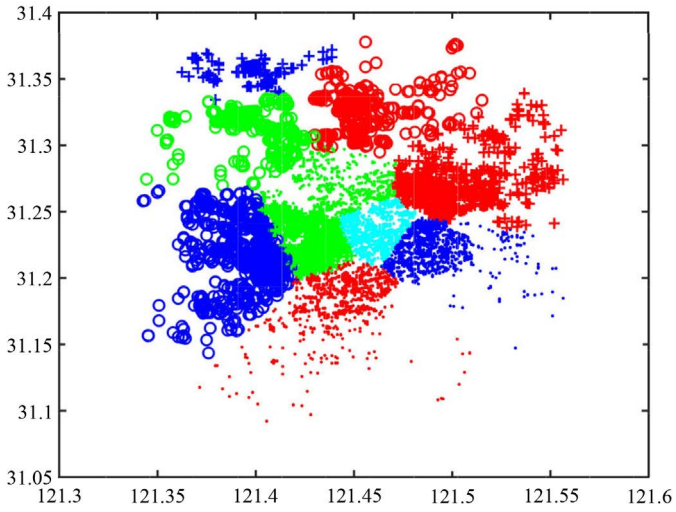

Figure 3. K-Means clustering results
图 3. K-Means 聚类结果

可以看到，聚类效果较好，但郊区因为点单人数相对较少且分散，因此区域较大，这些部分可以人为进行更细致的分类。这是因为普通的 K-Means 算法并不考虑聚集之间的均衡性，因而虽然适合用来做识别，但在具体的划分应用上，特别是在定量定比的划分上，用处不大。

计算训练集中所有观测到最近质心的平方距离之和，得到 $Dis = 2.8902$。

3. 改进的 K-Means 算法

3.1. 改进的基本思路

原始的 K-Means 聚类算法没有将节点的权重参与到聚类划分的过程中去，完全无视节点权重。我们

需要一种聚类方法，从聚类出事时即开始考虑权重的影响，并且将权重作为一种自发调解聚类大小的工具。

如何将节点权重这一因素考虑到划分中来呢？我们先思考一下划分的实质：判断一个点属于哪一个聚类点集的依据是，依次比较该节点到这些点集中心的聚类然后选取最近的距离，将其加入那个点集。在做距离比较的时候，我们采用的比例是 1:1，对于点 d_i，其到聚类 C_k 中心 c_k 的划分距离为：

$$d_i = \sqrt{\left(x_k - x_i\right)^2 + \left(y_k - y_i\right)^2}$$

现在我们假设在做距离比较时不再采用 1:1 的比率，而是采用和权重有关的度量方式，对于点 d_i，其到聚类 C_k 中心 c_k 的划分距离为：

$$d_i = \frac{\sqrt{\left(x_k - x_i\right)^2 + \left(y_k - y_i\right)^2}}{\text{weight}\left(C_k\right)^{\gamma}}$$

其中，$\text{weight}\left(C_k\right)$ 是对点集集合 C_k 求其总权重。γ 是衰减权重。可以看到，当 $\gamma = 0$ 时，上式和普通聚类的划分距离公式其实等价；当 $\gamma > 0$ 时，总权重越大的聚类对于点的“引力”的“折扣”越大，因而总权重约小的聚类，越有希望在下一次的聚类划分中得到新的点。

可以将不考虑权重的 K-Means 聚类认为是权重衰退系数为零的 Weighted K-Means 聚类。

3.2. Weighted K-Means 算法描述

至此完整阐述心的聚类方法，即 Weighted K-Means 划分算法。

算法首先选择 k 个对象，每个对象代表一个初始的聚类质心。对于其余的每一个对象，根据该对象与各聚类质心的距离，把它们分配到与之最相似的聚类中。在以上过程中因为初始聚类尚未形成，我们按照 $\gamma = 0$ 来进行划分。然后计算每个聚类的权重质心。

即对于由具有权重 w 和坐标 $\left(x, y\right)$ 的 n 个二维带权点组成的点阵集合 C_k，我们计算出其带权质心来。

$$x_k = \frac{\sum_{i=k_1}^{k_n} x_i w_i}{\sum_{i=k_1}^{k_n} x_i}, y_k = \frac{\sum_{i=k_1}^{k_n} y_i w_i}{\sum_{i=k_1}^{k_n} y_i}$$

然后计算出该聚类的总权：

$$w_k = \sum_{i=k_1}^{k_n} w_i$$

重复以上过程，直到对于每个点集，都求出其带权重心以及总权重来。

接下来对于全部的点进行重新聚类，因为在(1)中我们已经计算得到了初始聚类点群，因此从这步开始我们进行带权划分。计算带权距离：

$$d_{ij} = \frac{\sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2}}{w_j^{\gamma}}$$

得到 $d_{i1}, d_{i2}, \cdots, d_{ik}$ 共 k 个带权距离，取最小值 d_{if}，将点 d_i 加入划分聚类 d_f。

对所有点进行同样的划分。

用(1)中的方法重新计算每一个聚类的带权聚类重心和总权重。

重复上述过程(2)和(3)直到聚类收敛。

流程图如下：(图 4)

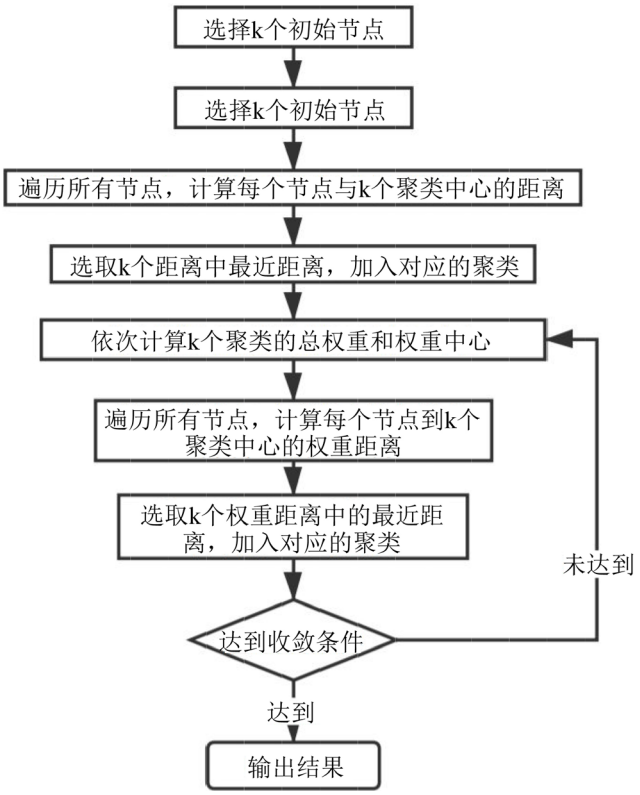

Figure 4. Algorithm flowchart of Weighted K-Means
图 4. Weighted K-Means 算法流程图

3.3. Weighted K-Means 算法的 Python 代码

因篇幅问题不予列出，详见附录。

可以看到，聚类过程分为两部分：一开始没有关于权重的信息时，我们使用等权方式划分；从第二次聚类开始，已有前次聚类的结果，因此可以计算权重重心并使用不等权的划分。

3.4. Weighted K-Means 算法的代码结果

取 $k=10$，使用 Matlab 根据分类标准作图(图 5)。

可以看到，聚类效果较好，郊区因为点单人数相对较少且分散的问题得到改善，需要人为进行更细致的分类部分变少。这是因为改进后的 Weighted K-Means 算法考虑了聚集之间的均衡性，因而虽然适合用来作识别，但在具体的划分应用上，特别是在定量定比的划分上，用处不大。

计算训练集中所有观测到最近质心的平方距离之和，得到 $Dis=1.6695$ 比改进前的方法有明显进步，可以说明该方法效果好。

4. 结论和展望

K-Means 算法作为聚类分析经典的方法之一，对大型数据集的处理效率较高，尤其是对样本分布呈现类团聚状时，可以达到很好的聚类效果。但是作为一个 NP 问题，它对初始聚类中心的选择以及 k 值的大小的确有很大的依赖性，在一定程度上限制了 K-Means 算法的实际应用能力。

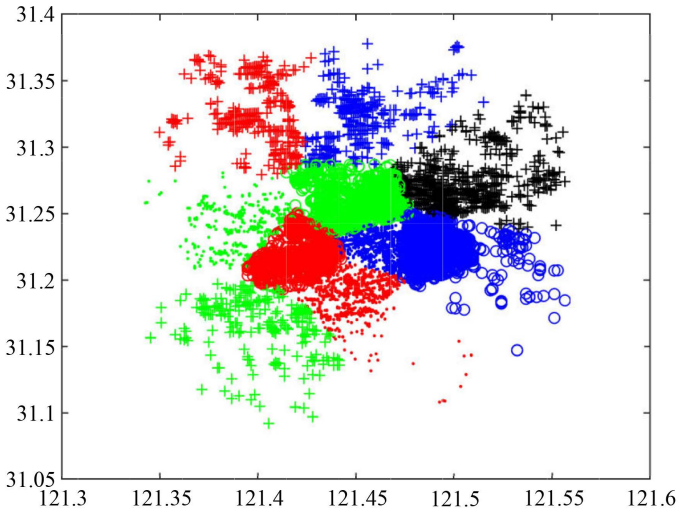

Figure 5. Weighted K-Means clustering results
图 5. Weighted K-Means 聚类结果

不幸的是，通常的 K-Means 聚类方法对于含有权重的二维坐标点集的分类束手无策，这导致了 K-Means 方法在使用上有着很大的局限性。因此本文尝试通过对算法的改进，达到加入权重信息的目的。

本文通过 K-Means 和 Weighted K-Means 算法分别对上海外卖员接单数据进行分类，处理了上海市外卖接单信息，最终给出合理可行的外卖员接单区域的划分。并进行了对比研究，结果发现 Weighted K-Means 聚类方法不仅能够很好地参考点的权重信息，并且在实际运用中，使得分类结果更加有效。通过上述聚类方法的实现，本文也成功给出了一种全新的划分外卖接单区域的方法，并优化现有外卖模式、提升外卖效率以及顾客满意度。

然而，K-Means 聚类算法具有很好的伸缩性，但是对于聚类初始点的选择很挑剔。良好的初始节点选择算法，对于提高聚类收敛效率，获得合理的收敛结果有很重要的意义。然而因为研究水平有限，无法对随机选取初始聚类点这种并不合适的方法进行改进。

另外的一个问题存在于聚类收敛性上。以上实验没有碰到聚类无法自然收敛的情况，而在实际中，聚类无法收敛的点集虽然极其少见，但依旧存在。

基金项目

本文为 2018 年上海市大学生创新创业训练计划项目“基于查找算法的外卖员接单区域划分问题研究”（项目编号 201810273105）的研究成果之一。

致　谢

感谢 2018 年上海市大学生创新创业训练计划项目(项目编号 201810273105，项目名称“基于查找算法的外卖员接单区域划分问题研究”)的资助。

参考文献

[1] 龚心怡, 苏燕欣, 滕明宏, 钱永贵. 当前外卖配送模式中的问题及对策分析[J]. 中国商论, 2019(3): 32-33.

[2] 李桃迎, 吕晓宁, 李峰, 陈燕. 考虑动态需求的外卖配送路径优化模型及算法[J]. 控制与决策, 2019, 34(2): 406-413.

[3] 陈霞. 计算机数据挖掘技术的开发及其应用[J]. 现代营销(经营版), 2019(2): 130-132.

[4] 王妍捷. 外卖O2O市场下的物流策略研究[J]. 物流技术, 2015, 34(10): 23-25.
[5] 黄心, 吴学群, 袁清冽. 蚁群算法在外卖配送路径规划中的应用[J]. 价值工程, 2017, 36(5): 65-67.
[6] 赵刚, 李昆. 几种查找算法的比较[J]. 科技信息, 2010(9): 152-168.
[7] 王倩影. 外卖O2O行业配送模式选择研究[D]: [硕士学位论文]. 北京: 北京交通大学, 2017.
[8] 饿了么星选[EB/OL]. https://star.ele.me/waimai/shop/, 2018-10.
[9] 李翠霞, 于剑. 一种模糊聚类算法归类的研究[J]. 北京交通大学学报: 自然科学版, 2005, 29(2): 17-21.
[10] 刘叶, 吴晟, 周海河, 吴兴蛟, 韩林峄. 基于K-means聚类算法优化方法的研究[J]. 信息技术, 2019, 43(1): 66-70.

附　录

改进的 WeightedK-Means 算法的 Python 代码

```
def weighted_kmeans(X, n_clusters, weights=None, n_init=10, max_iter=300, tol=1e-4):
    if len(X.shape) == 1:
        X = X[:,np.newaxis]
    if n_init<= 0:
        raise ValueError("Number of iteration n_init=%d must be bigger than zero." % n_init)
    if n_clusters< 0:
        raise ValueError("Number of clusters n_clusters=%d must be at least 1." % n_clusters)
    if n_clusters>X.shape[0]:
        raise ValueError("Number of clusters n_clusters=%d is larger than number of samples %d"
                         % (n_clusters, X.shape[0]))
    X = np.array(X, dtype=float)
tol = _tolerance(X, tol)
    # subtract mean of X for more accurate distance computations
X_mean = X.mean(axis=0)
    X -= X_mean
    # precompute squared norms of data points
x_squared_norms = np.einsum('ij,ij->i', X, X)
best_labels, best_inertia, best_centers = None, None, None
    # For a single thread, less memory is needed if we just store one set
    # of the best results (as opposed to one set per run per thread).
    for it in range(n_init):
        # run a k-means once
        labels, inertia, centers, n_iter_ = _weighted_kmeans_single(
            X, n_clusters, weights=weights, max_iter=max_iter, tol=tol,
x_squared_norms=x_squared_norms)
        # determine if these results are the best so far
        if best_inertia is None or inertia <best_inertia:
best_labels = labels.copy()
best_centers = centers.copy()
best_inertia = inertia
    # add mean of X to original distribution
best_centers += X_mean
    return best_centers, best_labels, best_inertia
def _weighted_kmeans_single(X, n_clusters, x_squared_norms, weights=None, max_iter=300, tol=1e-4):
best_labels, best_inertia, best_centers = None, None, None
    # init
centers = _k_init(X, n_clusters, x_squared_norms=x_squared_norms)
```

```
        # Allocate memory to store the distances for each sample to its
        # closer center for reallocation in case of ties
        distances = np.zeros(shape=(X.shape[0],), dtype=np.float64)
        # iterations
        for i in range(max_iter):
centers_old = centers.copy()
            # labels assignment is also called the E-step of EM
            labels, inertia = \
                _labels_inertia(X, x_squared_norms, centers, weights=weights,
                                distances=distances)
            # computation of the means is also called the M-step of EM
centers = _weighted_kmeans._centers_dense(X, labels, n_clusters, distances, weights)
            if best_inertia is None or inertia <best_inertia:
best_labels = labels.copy()
best_centers = centers.copy()
best_inertia = inertia
            # break if diff is less than tol at this iteration
            diff = centers_old - centers
            if (diff * diff).sum() <= tol:
                break
        return best_labels, best_inertia, best_centers, i + 1
def _labels_inertia_precompute_dense(X, x_squared_norms, centers, distances, weights=None):
n_samples = X.shape[0]
        k = centers.shape[0]
all_distances = euclidean_distances(centers, X, x_squared_norms,
                                                    squared=True)
        labels = np.empty(n_samples, dtype=np.int32)
labels.fill(-1)
mindist = np.empty(n_samples)
mindist.fill(np.infty)
        for center_id in range(k):
dist = all_distances[center_id]
labels[dist<mindist] = center_id
mindist = np.minimum(dist, mindist)
        if n_samples == distances.shape[0]:
            # distances will be changed in-place
distances[:] = mindist
        if weights is None:
            inertia = mindist.sum()
```

```
        else:
            inertia = (mindist * weights).sum()
        return labels, inertia
def _labels_inertia(X, x_squared_norms, centers, weights=None, distances=None):
n_samples = X.shape[0]
        # set the default value of centers to -1 to be able to detect any anomaly
        # easily
labels = -np.ones(n_samples, np.int32)
if distances is None:
            distances = np.zeros(shape=(0,), dtype=np.float64)
        # distances will be changed in-place
        return _labels_inertia_precompute_dense(X, x_squared_norms, centers, distances, weights=weights)
def _k_init(X, n_clusters, x_squared_norms, n_local_trials=None):
n_samples, n_features = X.shape
centers = np.empty((n_clusters, n_features))
        assert x_squared_normsis not None, 'x_squared_norms None in _k_init'
        # Set the number of local seeding trials if none is given
        if n_local_trials is None:
            # This is what Arthur/Vassilvitskii tried, but did not report
            # specific results for other than mentioning in the conclusion
            # that it helped.
n_local_trials = 2 + int(np.log(n_clusters))
        # Pick first center randomly
center_id = random.randint(n_samples)
centers[0] = X[center_id]
        # Initialize list of closest distances and calculate current potential
closest_dist_sq = euclidean_distances(
            [centers[0]], X, Y_norm_squared=x_squared_norms, squared=True)
current_pot = closest_dist_sq.sum()
        # Pick the remaining n_clusters-1 points
        for c in range(1, n_clusters):
            # Choose center candidates by sampling with probability proportional
            # to the squared distance to the closest existing center
rand_vals = random.random_sample(n_local_trials) * current_pot
candidate_ids = np.searchsorted(closest_dist_sq.cumsum(), rand_vals)
            # Compute distances to center candidates
distance_to_candidates = euclidean_distances(
                X[candidate_ids], X, Y_norm_squared=x_squared_norms, squared=True)
            # Decide which candidate is the best
```

```
best_candidate = None
best_pot = None
best_dist_sq = None
            for trial in range(n_local_trials):
                # Compute potential when including center candidate
new_dist_sq = np.minimum(closest_dist_sq,
distance_to_candidates[trial])
new_pot = new_dist_sq.sum()
                # Store result if it is the best local trial so far
                if (best_candidate is None) or (new_pot<best_pot):
best_candidate = candidate_ids[trial]
best_pot = new_pot
best_dist_sq = new_dist_sq
            # Permanently add best center candidate found in local tries
centers[c] = X[best_candidate]
current_pot = best_pot
closest_dist_sq = best_dist_sq
    return centers
def _tolerance(X, tol):
    variances = np.var(X, axis=0)
    return np.mean(variances) * tol
class WeightedKMeans():
    def __init__(self, n_clusters=8, n_init=10, max_iter=300, tol=1e-4):
self.n_clusters = n_clusters
self.max_iter = max_iter
self.tol = tol
self.n_init = n_init
    def _check_fitted(self):
        if not hasattr(self, "cluster_centers_"):
            raise AttributeError("Model has not been trained yet.")
    def fit(self, X, weights=None):
self.cluster_centers_, self.labels_, self.inertia_ = \
weighted_kmeans(
                    X, n_clusters=self.n_clusters, weights=weights,
n_init=self.n_init, max_iter=self.max_iter, tol=self.tol)
        return self
    def fit_predict(self, X):
        return self.fit(X).labels_
    def fit_transform(self, X, y=None):
```

```
            X = self._check_fit_data(X)
            return self.fit(X)._transform(X)
        def transform(self, X):
self._check_fitted()
            X = self._check_test_data(X)
            return self._transform(X)
        def _transform(self, X):
            return euclidean_distances(X, self.cluster_centers_)
        def predict(self, X):
self._check_fitted()
x_squared_norms = np.einsum('ij,ij->i', X, X)
            return _labels_inertia(X, x_squared_norms, self.cluster_centers_)[0]
        def score(self, X):
self._check_fitted()
x_squared_norms = np.einsum('ij,ij->i', X, X)
            return -_labels_inertia(X, x_squared_norms, self.cluster_centers_)[1]
```

Hans汉斯

知网检索的两种方式：

1. 打开知网页面 http://kns.cnki.net/kns/brief/result.aspx?dbPrefix=WWJD
   下拉列表框选择：[ISSN]，输入期刊 ISSN：2325-2251，即可查询
2. 打开知网首页 http://cnki.net/
   左侧“国际文献总库”进入，输入文章标题，即可查询

投稿请点击：http://www.hanspub.org/Submission.aspx
期刊邮箱：sa@hanspub.org