



CPIPC
中国研究生
创新实践系列大赛



中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生
数学建模竞赛

学 校 北京大学

参赛队号 19100010050

队员姓名 1. 谢荣昌
2. 于飞
3. 杨乐

中国研究生创新实践系列大赛

“华为杯”第十六届中国研究生

数学建模竞赛

题 目 基于随机搜索和聚类分析以及自编码器再优化的汽车行驶工况构建研究

摘要：

本文研究了汽车行驶工况的构建问题。主要创新点在于结合了随机搜索和聚类分析法，并使用径向基核化主成分分析和 BIRCH 层次聚类方法进行改进，最后首次提出使用自编码器直接优化已有工况的特征参数误差的方法。

问题一：我们对原始采集数据中的不良数据进行了预处理，主要包括(1)针对 GPS 信号丢失导致的时间不连续问题，我们进行了数据插值和分片，共计增加 3480 条数据，删除 144 条数据；(2)针对加减速度异常问题，我们首先用插值结果替换异常结果，之后若仍具有异常结果则直接删除，共计删除 24051 条数据。(3)针对长期停车和低速行驶情况，我们按照怠速情况处理，当怠速时长超过 180 秒时，仅保留后 180 秒数据，删除剩余数据，共计删除 20482 条数据。经过数据预处理之后，我们将得到可用于此后汽车行驶工况曲线构建的最终数据集，共计 455267 项数据记录，其中各文件记录数分别为 175152、139294、140821。

问题二：我们对问题一产生的多个分片进行运动学片段提取，主要包括(1)删除未以 0km/h 为起始或终止的片段；(2)删除行驶持续时间(不包括怠速时间)小于 20 秒的片段。最终我们提取出 2088 个运动学片段，其中各文件片段数分别为 845、648、595。

问题三：我们根据汽车行驶特点，构建了包含 11 项特征参数的汽车运动特征评估体系。为了构建具有代表性的工况曲线，首先使用了运动学片段随机搜索(选择)法，得到了最小误差率为 4.36% 的工况曲线。但该方法候选空间过大且结果存在较大随机性(多次随机搜索的所有结果的平均误差率为 14.81%)。其次我们实现了目前广泛被使用的聚类分析法，它通过将运动学片段特征进行聚类，并选择最靠近聚类中心的运动片段来组成行驶工况。该方法有效减少了计算量，但得到的工况曲线特征误差较大，为 9.40%。

考虑前两类方法的缺点，我们提出以聚类中心的邻域作为候选集，再在其中进行带约束的搜索，从而寻找最优的运动学片段组合。该方法能有效减少搜索空

间，并增加寻找优化解的概率。实验表明该算法显著优于目前已有算法，将误差率由 9.40% 下降至 3.18%。

针对降维与聚类方法，我们对主流的线性主成分 (PCA) 与 K 均值聚类结合的方法进行了创新性改进。由于运动学片段的复杂性，这里使用了径向基 (RBF) 核化主成分分析，对原始特征进行非线性降维。之后通过对离群点和孤立点更鲁棒的 BIRCH 层次聚类分析，进行运动学片段的分类。该方法进一步将误差率由 3.18% 下降至 2.94%。

最后，我们首次提出利用生成模型自编码器 (AutoEncoder) 对上述结果进行行驶工况再优化，该算法利用自编码器可重构输入的特点，在保持原工况曲线整体趋势不变的同时，优化生成结果中重要的特征参数，最终使得生成的曲线的误差率再次显著下降至 2.43%。

关键字：运动学片段法，核化主成分分析，层次聚类，自编码器，深度学习

目录

1. 问题重述	5
1.1 问题背景	5
1.2 问题提出	5
2. 问题分析	6
3. 模型假设	7
4. 符号系统	7
5. 问题一：数据预处理	8
5.1 车辆行驶状态划分	8
5.2 信号丢失及其处理方法	8
5.3 加减速速度异常及其处理方法	9
5.4 长期停车及其处理方法	10
5.5 低速行驶及其处理方法	10
5.6 长时间怠速及其处理方法	11
5.7 数据预处理结果	12
6. 问题二：运动学片段的划分	13
6.1 运动学片段的定义	13
6.2 运动学片段的划分	14
7. 问题三的建模与求解	16
7.1 解题思路与建模	16
7.2 方法原理	17
7.2.1 随机搜索	17
7.2.2 主成分分析	17
7.2.3 核化主成分分析	18
7.2.4 K 均值聚类	19
7.2.5 BIRCH 层次聚类	20
7.2.6 AutoEncoder 模型	21
7.3 特征评估体系的建立	22
7.4 特征评估参数的计算	22

7.5 模型求解	25
7.5.1 基于随机搜索方法构建行驶工况曲线	25
7.5.2 基于 PCA 和 K 均值聚类模型构建行驶工况曲线	25
7.5.3 基于 Kernel PCA 和 BIRCH 聚类模型构建行驶工况曲线	32
7.5.4 基于 AutoEncoder 模型优化汽车行驶工况曲线	36
8. 结果分析与比较	39
8.1 各方法所得汽车行驶工况的特征参数	39
8.2 各方法所得汽车行驶工况的评估结果	39
8.2.1 特征参数误差定义	39
8.2.2 各个方法所得结果的误差比较与分析	40
9. 模型评价与改进	42
9.1 模型的优点	42
9.2 模型的缺点	42
9.3 模型的改进与推广	42
10. 参考文献	44
附录 A 特征参数提取程序	45
附录 B 误差计算程序	46
附录 C 主成分分析与聚类	48
附录 D 邻域随机搜索	48
附录 E AutoEncoder	49

1. 问题重述

1.1 问题背景

随着我国社会经济的高速发展，机动车保有量呈现出不断提升的状态。尽管汽车为人们的出行提供了极大的便利，但同时也给我国城市带来了日益增长的城市交通压力以及一系列亟待解决的环境污染、能源危机等问题。汽车行驶工况（Driving Cycle）又称车辆测试循环，作为汽车工业一项共性的关键技术，是车辆在一定的行驶环境下，在一定的道路上行驶时的速度-时间曲线[1]，可用于评估燃油消耗量、汽车污染物排放、新车型的开发、道路交通事故控制等。该曲线的构建常使用运动学片段划分、提取及组合的方法，其中运动学片段被定义为汽车从怠速状态开始至下一个怠速状态开始之间的车速区间[2]，通常包括一个怠速部分和一个行驶部分。因此，采用合理的汽车行驶工况对于交通规划、管理及控制策略的制定和实施具有重要的指示意义。

由于发展水平、气候条件、人们的驾驶习惯及道路交通状况存在差异，直接复制采用如欧洲的 NEDC 行驶工况、WLTC 世界轻型车测试循环等不能准确的代表中国及其不同城市实际的汽车行驶工况，以此为基础进行的一些车辆测试得到的结果也会与现实情况有差别。用它作为我国汽车开发、车辆燃油消耗量和污染物排放的测定的循环工况是缺乏合理性的。目前这方面的研究我国尚处于起步阶段，仅有北京、上海、合肥等城市构建了因地制宜的汽车行驶工况。我们很有必要提出先进的、可推广应用于构建不同城市交通状况的车辆行驶工况的方法，为分析不同城市的车辆排放、油耗状况、交通势态进而制定相应的控制策略提供科学依据。

1.2 问题提出

题目给出了 3 个数据文件，每个数据文件为同一辆车在不同时间段内以 1Hz 采样频率所采集的数据，要求利用该数据解决如下问题：

问题一：在数据获取过程中，由于采集设备、周围环境等因素的影响，可能导致数据出现错误或者偏差，原始数据的质量无法保证。其中往往包含一些不良数据值，需要首先进行数据补齐、数据清洗等一系列数据预处理操作。

问题二：根据合理的规则和方法，将经过预处理后的数据划分为多个运动学片段，同时需要给出各数据文件最终对应提取的运动学片段的数量，为之后构建汽车行驶工况曲线做好基础准备工作。

问题三：(1) 明确整体思路，从降维、聚类等多角度出发，确定构建汽车行驶工况曲线所需的各类数学模型或算法，同时需要对使用方法的原理及适用性给出介绍。(2) 选取合适的特征参数，构建特征评估体系，以此作为构建汽车行驶工况曲线的评价因子及有效性评估准则。(3) 以(2)中特征评估体系为基准，使用(1)中的方法，构建汽车行驶工况。分别计算出汽车行驶工况与处理后数据的各项运动特征指标值，采用一定的评价标准对构建的行驶工况进行精度评估和误差分析，以此验证构建的汽车行驶工况曲线的合理性。

2. 问题分析

对于问题一，我们首先需要确定预处理针对的不良数据的各类特征及相应的预处理方法：(1) 由于高层建筑覆盖、经过隧道、仪器出现故障等，致使 GPS 信号丢失，造成数据的时间不连续。对此我们的处理方法是细分情况进行数据插补或数据剔除；(2) 汽车加、减速度出现异常、发生突变的数据（题中规定普通轿车通常情况下， 0 至 $100km/h$ 最大加速度不超过 $3.97m/s^2$ ，紧急刹车最大减速度在 $7.58m/s^2$ ）。对此我们的处理办法是先寻找不满足题中规定条件的汽车加、减速度数据异常点然后使用数据插补操作进行替换处理，插值操作后仍旧包含异常加、减速度的分时片段则直接删除。(3) 长时间停车（如停车不熄火等候人、停车熄火了但采集设备仍在运行等）所采集的异常数据，我们认为当车速为 $0km/h$ 且连续出现大于或者等于一定时间长度时，视为怠速处理。(4) 当车辆出现长时间堵车、断断续续低速行驶的情况时，车速维持在 $10km/h$ 以下，也将其按照怠速情况处理。(5) 当怠速时间超过 180 秒时，我们认为出现异常情况，仅保留怠速状态的 180 秒时长。经过这 5 步数据预处理操作之后，我们将得到可用于此后汽车行驶工况曲线构建的最终数据集。

对于问题二，我们需要在构建汽车行驶工况曲线之前确定好运动学片段划分的规则并实践操作。由运动学片段的定义可知，我们将主要根据各怠速状态点来进行片段切割，将从怠速状态开始至下一个怠速状态开始之间的各个小片段作为一个数据单元。此外，可能存在异常地速度未从 $0km/h$ 开始发动或者未在 $0km/h$ 完成终止的部分运动学片段，我们将其视为错误，需要再次进行删除操作。同时运动学片段可能还需满足一些合理的筛选条件，我们应该在进行运动学片段划分的过程中将这些均纳入考虑。

对于问题三，汽车行驶工况的构建首先需要我们提出一套完整的思路和流程，并对其间所使用的方法及其科学性、合理性进行介绍。我们将主要基于随机搜索、PCA 降维-K 均值聚类组合、Kernel PCA 降维-BIRCH 聚类组合三大方法进行，并对其中所得精度最高的方法结果进行 AutoEncoder 模型优化。在进行具体的建模之前，我们还需查阅现有相关文献，总结得到用以刻画运动学状态的有效性的、典型的特征参数。在建立一个规范的特征评估指标体系之后，我们将基于特征参数评估体系，使用此前介绍的各类方法实现代表性汽车行驶工况的确定，并给出不同方法的误差分析、结果优劣比较以及优化后模型精度的改进，同时也能够较好地验证我们所使用的方法的有效性及准确性。

3. 模型假设

1. 在时间间隔 5 秒之内，车辆前后时刻的速度是高度线性相关的。若超过此时间间隔，则无法可靠的预测速度。
2. 当信号丢失在短时间内频繁出现时，代表此时数据质量较差。
3. 假设长时间堵车、断断续续低速行驶时，最高车速小于 10km/h。
4. 假设怠速时间最长不超过 180 秒，超过 180 秒为异常情况。
5. 假设一辆汽车在 0 至 100km/h 的加速时间大于 7 秒，紧急刹车最大减速度在 7.5 至 $8m/s^2$ 之间。
6. 去除异常数据，不影响数据的整体分布情况。
7. 合理的运动学片段的行驶时间，需要大于 20 秒。
8. 选择的特征参数集能够反映原数据的主要特征。

4. 符号系统

符号	含义	单位
T	运行时间	s
S	运行距离	km
v_m	平均速度	km/h
v_{mr}	平均行驶速度	km/h
v_{max}	最大速度	km/h
a_{am}	(加速段) 平均加速度	m/s^2
a_{dm}	(减速段) 平均减速度	m/s^2
a_{max}	最大加速度	m/s^2
a_{min}	最小减速度	m/s^2
v_{sd}	速度标准差	km/h
a_{asd}	加速度标准差	m/s^2
a_{dsd}	减速度标准差	m/s^2
P_i	怠速时间比	%
P_a	加速时间比	%
P_d	减速时间比	%
P_c	匀速时间比	%

5. 问题一：数据预处理

由于数据采集终端、车辆自身、周围环境的一些原因，采集到的原始数据通常存在一定的异常，包括车速数据的丢失（车速出现停滞带）、出现白噪声、长时间停车或者长期怠速等问题。经过统计可得原始记录为 496464 条，其中三个文件包含的记录数分别为 185725、145825 和 164914。针对数据异常产生的原因，我们分别给出了不同的数据预处理操作。但在进行数据预处理之前，我们首先需要对车辆的行驶状态进行明确的划分，便于进行预处理及后续操作。

5.1 车辆行驶状态划分

对于车辆加速状态、匀速状态、怠速状态、减速状态四种运行状态而言，本文按照如下标准对其进行界定：

怠速状态：车辆停止运动，但发动机保持最低转速运转的连续过程。

匀速状态：车辆运行加速度的绝对值小于 $0.1m/s^2$ 的非怠速的连续过程。

加速状态：车辆运行加速度大于 $0.1m/s^2$ 的连续过程。

减速状态：车辆运行加速度小于 $-0.1m/s^2$ 的连续过程。

5.2 信号丢失及其处理方法

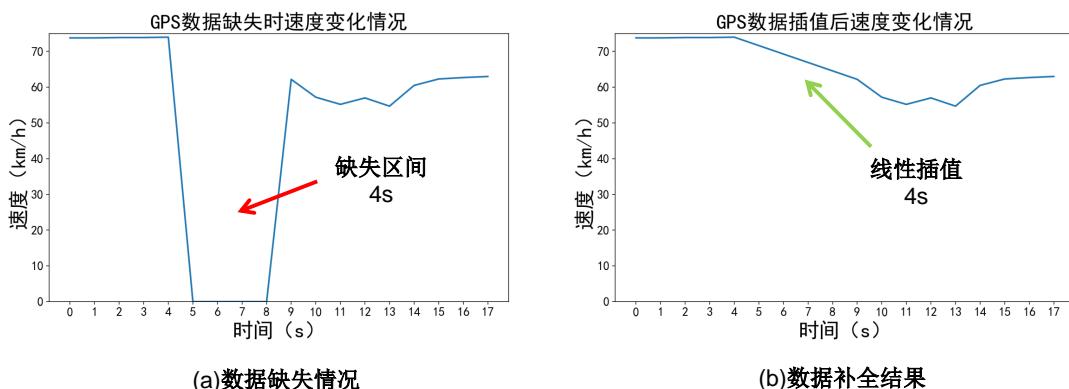


图 1 信号丢失问题处理

原始数据的采集终端主要借助于 GPS，当接收定位卫星信号受阻时，便会出现车速数据丢失。车辆在城市道路内行驶时，GPS 接收信号会不可避免的受到高层建筑、隧道等路段的遮挡，同时仪器也可能因环境或自身问题出现瞬时故障等，造成定位不成功，车速数据丢失。此外，当采集设备关闭时，数据也会出现较长的缺失段。

对于这种数据异常，我们的处理方法是：(1) 首先对原始数据的相邻时间求差，提取出时间差大于 1 秒的缺失段；(2) 当缺失段时间差小于 5 秒时，采用线性插值，把短期不连续的数据连接起来。此步骤共插补 3480 条数据；(3) 当缺失段时间差大于 5 秒时，将长时间不连续的数据分离为分时片段，并去除小于 20 秒的分时片段。此步骤共删除 144 条数据，剩余 499800 条数据。

例如，图 1(a) 中的第 5-8 秒时间段内实际上是没有记录的，此时如果不进行

处理，会严重影响数据的质量与可靠性。根据模型假设，在极短的时间内车辆前后时刻的速度是高度相关的。因此当缺失段时间差小于 5 秒时，我们利用缺失时刻前后的速度进行线性插值，最后的补全结果如图1(b)。当缺失段时间差大于 5 秒时，其速度难以预测，插值结果不够可靠。此时我们直接将此缺失段丢弃，并将数据拆分为分时片段，之后在分时片段内部进行后续操作。

5.3 加减速度异常及其处理方法

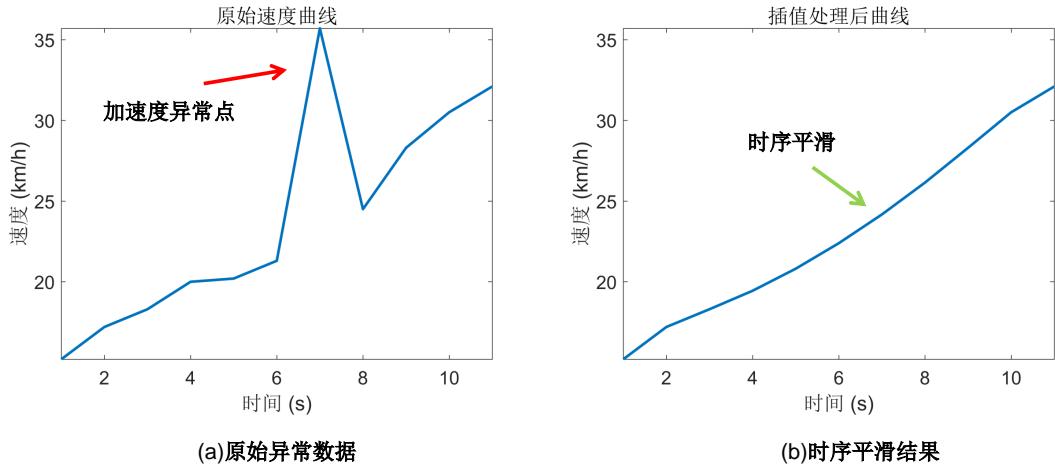


图 2 加速度异常时进行的插值处理

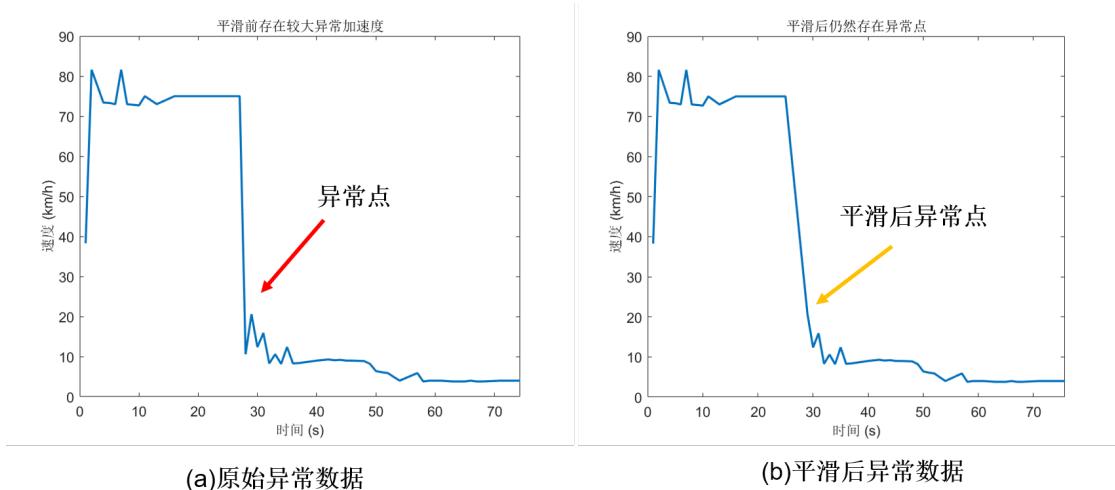


图 3 插值后仍然异常的低质量数据

由于数据采集终端可能出现突然暂停工作、定位不成功却将车速数据记录为 0，或者定位成功时也可能出现一定的车速偏差，此类现象普遍存在于数据采集过程中，常导致汽车的加、减速度出现突变异常值。

针对此类数据异常现象，我们首先搜索得到 0 至 100km/h 最大加速度超过 $3.97m/s^2$ 或者紧急刹车时最大减速度大于 $7.5m/s^2$ 的异常点，然后使用异常点的临近点（例如附近 10 秒的临近点）对其进行插值替换处理。这相当于借助异常

点的邻域信息，对其进行时序平滑处理。如图2所示，第7秒处存在加速度异常大的数据点（毛刺点），经过插值平滑后，速度曲线接近正常。

插值后如果仍旧包含异常加速度，则代表异常点附近速度也存在问题。此时可认为该分时片段可靠性较差，直接对其进行删除操作。如图3所示，该数据片段整体趋势十分不合理，一段高速段后直接接近停止，并且存在许多速度抖动的噪声。经过插值平滑后，减速度仍然十分大，因此这里直接将其删除。

此步骤中，寻找到加速度异常点共计885个。经过插值后之后，只剩下45个仍然异常的加速度点。将其分时片段丢弃后，删除了24051条数据，剩余475749条数据。

5.4 长期停车及其处理方法

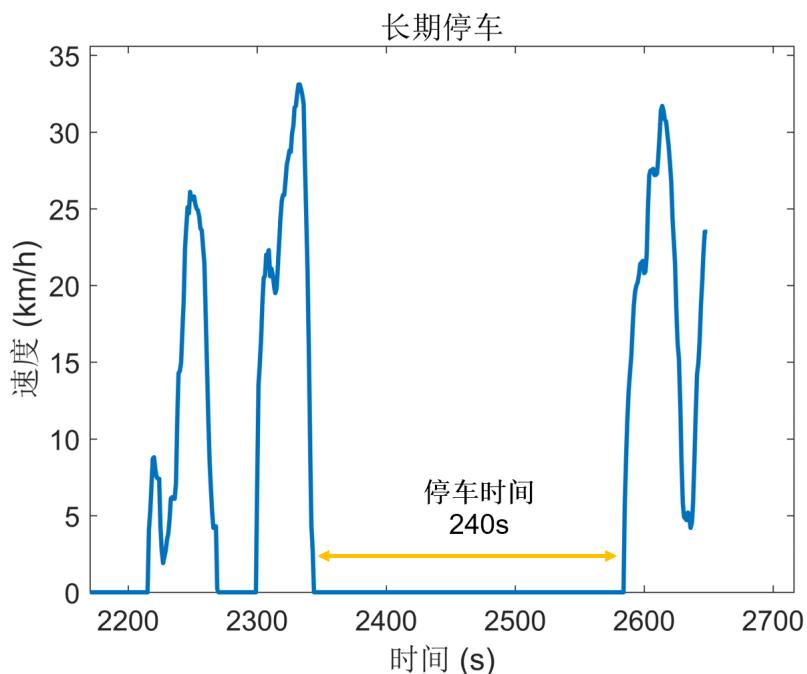


图4 检测出的停车事件

由于车主停车不熄火等候人、车辆与设备的匹配等问题造成车辆停车熄火后设备不断电、在车辆非工作时间段继续进行数据采集等多种情况的存在，长期停车所采集的数据其实也会出现异常现象。

对于长期停车问题，我们的处理方法是：检测出车速为0并且超过180秒的区间，将其标记为停车段。如图4所示，该段数据检测出长度为240秒的停车时间。由于停车时段可以看作一种异常的怠速状态，将在之后部分讲解如何处理。

5.5 低速行驶及其处理方法

由于城市存在车辆众多、道路拥挤、早晚上下班高峰等情况，可能出现车辆长时间堵车、“走走停停”（断断续续低速行驶）的现象，车速始终维持在10km/h以下。

对于这种长期低速行驶的问题，我们也按怠速情况处理。考虑到汽车起步以及刹车时，速度也可能短暂地处于10km/h以下，这种情况应该不属于低速行驶。

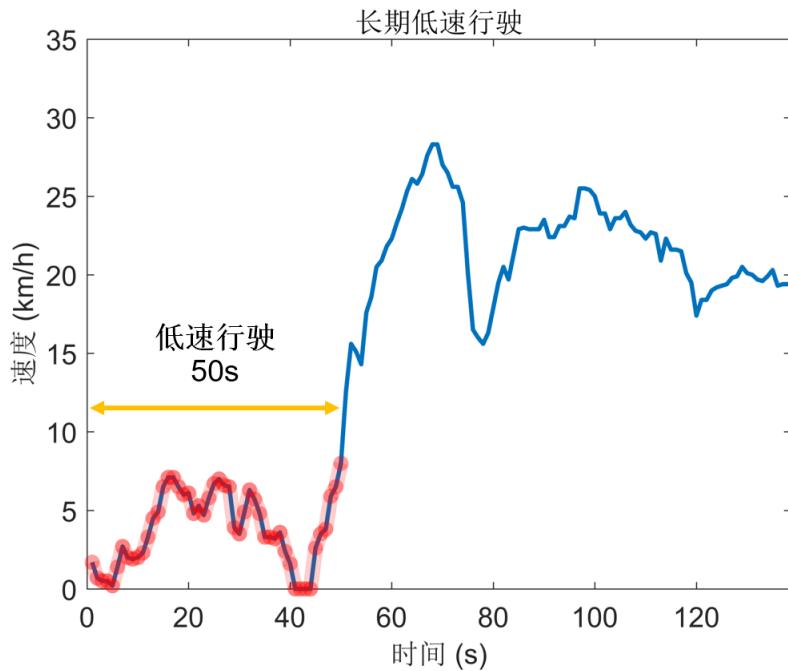


图 5 长期低速行驶事件

因此，当 30 秒内最高车速都小于 10km/h ，才将其视为低速行驶。如图5所示，标红的数据点速度都小于 10km/h ，并且持续了 50 秒，因此将这判定为长期低速行驶状态。低速行驶也可以看作是一类怠速状态。当低速行驶超过 180 秒时，其可被划分为异常怠速，处理方法如下节所示。

5.6 长时间怠速及其处理方法

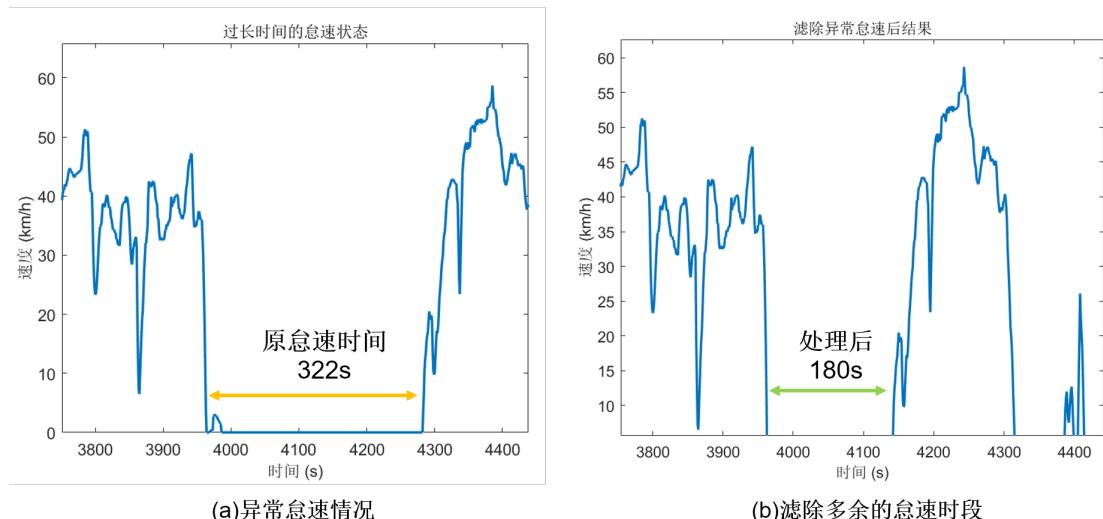


图 6 异常怠速状态的处理

怠速时间过长对于汽车行驶工况曲线构建的结果影响很大，我们一般认为，当怠速时间超过 180 秒为异常事件（包含长期停车，以及低速行驶等不同情况），怠速最长时间可按 180 秒处理。

对长时间怠速我们的处理方法是：寻找连续 180 秒的怠速状态，直接删除多余的数据，只保留最后的 180 秒用于之后的车辆行驶工况曲线的确定。如图6所示，原怠速时间达到了 322 秒，显著超出了正常的怠速时间，因为这里直接去除了该怠速状态的前 142 秒的数据。由于怠速期间，车速接近于 0，因此删去部分数据，几乎不影响加速度等参数的计算。此步骤共删除 20482 条数据，剩余 455267 条数据。

5.7 数据预处理结果

在原始记录 499944 条的基础上，经过以上一系列数据插补、数据替换、数据滤除等数据预处理操作，我们最终得到 455267 项数据记录。其中来自三个文件的数据分别有 175152、139294 以及 140821 条。

6. 问题二：运动学片段的划分

6.1 运动学片段的定义

汽车行驶工况构建的核心在于建立的代表性行驶工况能否反映实际行驶工况，对于实际行驶工况曲线的拟合精度要高。国内外汽车行驶工况的构建方法可以概括为两种方法：一种是以全部的预处理后的采集数据为基础，将全部的行驶状态看作一个连续的过程，采用统计学方法分析采集数据表现出来的各种特征，根据采集数据的路段道路等级对其进行分类，计算每个道路等级的运行时间频率，再根据各个道路相应的频率来构建汽车行驶工况曲线，这种方法的计算量较大，且数据间存在较高的相关性会导致数据冗余，不仅造成计算复杂，同时在构造工况时加入了重叠交叉的无效数据信息，给分析车辆行驶工况增加了困难。另一种则是将采集数据按照某种规则划分成许多片段，对划分的所有片段进行组合，最终构造代表性行驶工况。这些片段称为运动学片段。因为一个路段可能包含多种不同的交通特征，各种行驶状态并不完全取决于汽车的性能，也受到地理环境和道路类型等交通状况的影响，且同一路段随着时段、天气、汽车承载的成员数等的变化，也呈现不同的行驶状态。所以进行运动学片段划分的方式颇为符合实际状况。第二种方法对许多代表汽车实际行驶工况的微小片段进行分析，使得分析更为精确也更为便捷。

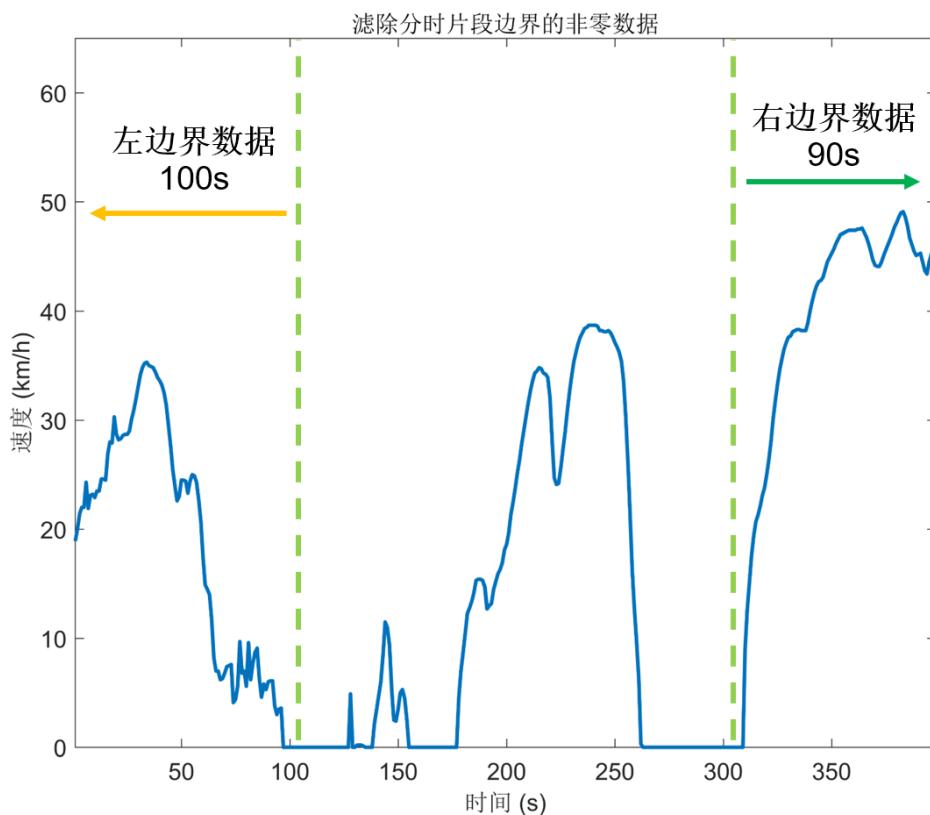


图 7 分时片段中边界数据的判定

本文采用第二种方法，按照运动学片段的定义将全部的采集数据进行划分。运动学片段是指相邻两个的停车点之间的汽车行驶过程，也就是将车辆从一个

怠速开始到下一个怠速开始的运动过程作为一个数据单元，通常包括一个怠速部分和一个行驶部分，是汽车的一段运行概况。由于各方面主客观因素的影响，运动学片段相互之间也存在较大差异，既有时间较短、速度较低的，也有时间较长、速度较高的，不同的运动学片段反映汽车在实际运行过程中的不同状况和交通特征。同时还有时间和速度相近但是怠速、匀速、加速、减速时间比不同的，所以大量研究在进行片段组合之前会先对其进行分类处理，一定程度上会提高之后片段组合的曲线拟合精度。

6.2 运动学片段的划分

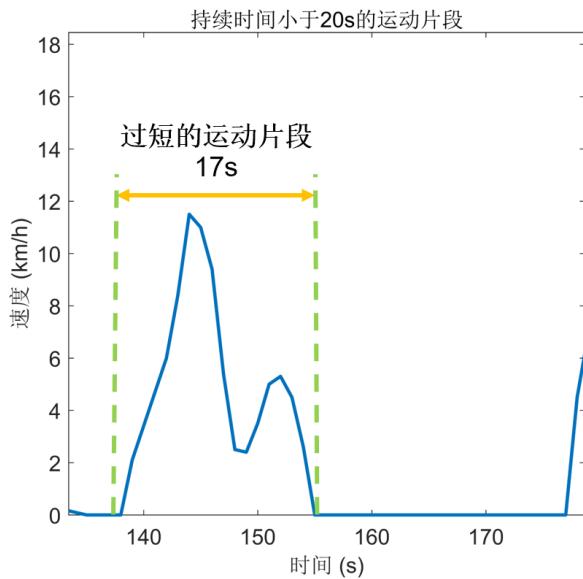


图 8 持续时间太小的运动片段

由于车辆数据采集终端存在一定的零点偏移，或者定位成功的条件下也会出现一定的车速偏差，很可能存在异常地速度未从 $0\text{km}/\text{h}$ 开始发动或者未在 $0\text{km}/\text{h}$ 完成终止的部分片段，这种情况我们将其视为数据错误。此外，由于之前将所有大于 5 秒的暂时信号丢失，我们都直接将数据拆分为了分时片段。因此，大量分时片段存在着以非零速度开始以及结束的情况，这部分数据同样也认为是干扰数据。

具体实现中，我们在每个分时片段中寻找怠速状态，将第一个怠速状态前的以及最后一个怠速状态后的时间段数据点当作边界数据点。如图7所示，左右边界都存在这大约 100 秒的干扰数据点，这些数据会被删除。

之后，将原始采集数据按照从一个怠速开始到下一个怠速开始的原则连续地分割为运动学片段。同时我们认为，运动学片段还需满足以下条件：

- (1) 运动学片段的行驶持续时间(不包括怠速阶段的时间)不少于 20 秒。
- (2) 车辆加速度在 $0 \text{--} 3.97\text{m}/\text{s}^2$ 范围内，减速度在 $-7.5\text{m}/\text{s}^2$ 范围内。

图8中所显示的短片段，时间只有 17 秒。这个短片段不符合条件 (1)，因此将会被排除。此外由于在问题一中对于数据的严格预处理使得我们已经满足条件 (2)，所以我们直接进行运动学片段的划分。在按怠速起始点分离运动片段完毕之后，继续滤除小于 20 秒的运动学片段，以满足额外条件 (1)，得到最终的运

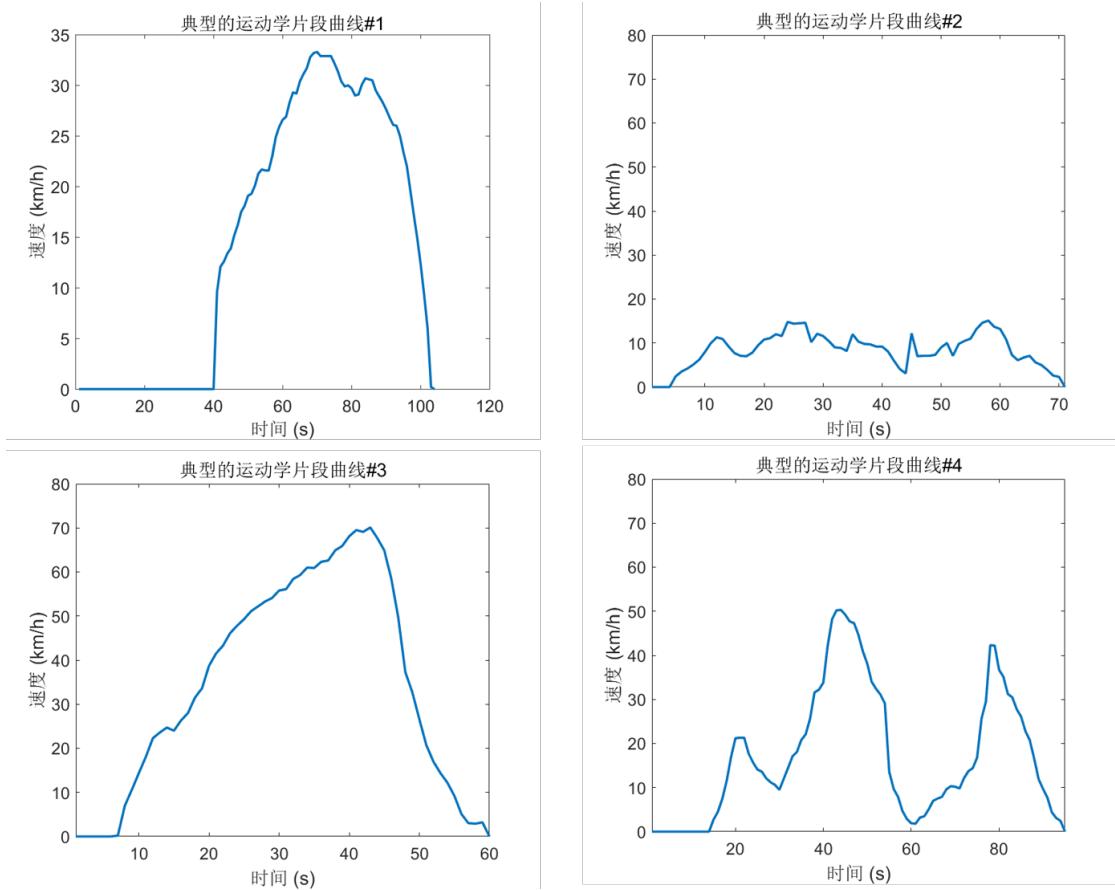


图9 典型的运动片段

动学片段划分结果。

表1 运动学片段统计数值

运动片段数量	总运行时间	总运行距离		最短时间	最长时间	平均时间	标准差
2088	331829	2510.51		23	2991	158.92	161.94

表1描述了划分得到的运动学片段统计数值。其中，前三列统计了总体数据的情况，后四列描述了各个片段持续时间的统计数据。另外，典型的运动学片段可视化如图9所示。从表1和图9可以看出，不同的运动学片段，平均速度、持续时间、速度变化特点都存在着较大的区别。因此，任何一个单独的运行学片段，都不能代表整体数据的分布情况。最终我们提取出2088个运动学片段，其中各文件片段数分别为845、648、595。

7. 问题三的建模与求解

7.1 解题思路与建模

汽车行驶工况曲线的构建是本题的核心，为了便于理解，在图10我们首先给出整体的解题思路和建模流程：

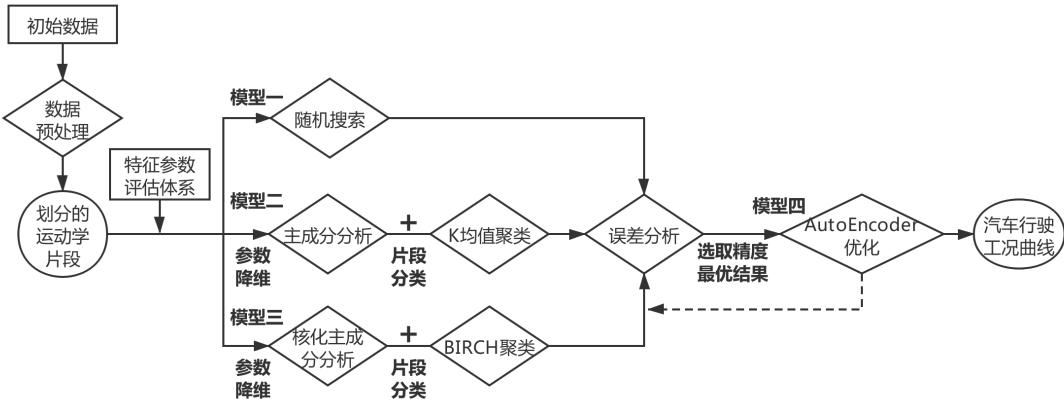


图 10 解题整体流程

数据预处理及运动学片段划分在问题一、二中已经分别得到解决。从流程图中可以看出，问题三我们需要做的包括用以描述运动学片段状态的特征参数评估体系的构建以及在此基础上构建模型进行汽车行驶工况曲线的构造，并对结果进行误差分析、精度评估。

模型一我们选取的是经典的随机搜索方法，这种方法是完全的遍历组合，并没有过多的优化搜索和计算简化，但却简单、易于理解和学习。

由于特征参数之间通常存在相关性过高、信息重叠冗余等情况，且建模者往往希望能够简化计算量，因此可以对其进行降维处理。同时部分运动学片段之间可能特征非常相似，在对运动学片段进行组合之前，进行事先聚类可能会进一步提高所得结果的拟合精度。所以我们决定先进行参数降维和片段分类处理，然后设定规则实现聚类片段的选取组合。

模型二中我们首先采用 PCA 方法对特征参数进行降维，然后采用 K 均值聚类进行片段分类，随后按照误差最小化规则进行片段组合。

模型三中我们首先使用改进的 Kernel PCA [3] 进行特征参数降维，在此基础上使用性能更为优越的 BIRCH 聚类 [4] 进行运动学片段分类，然后按照误差最小原则进行片段的选择及组合。

通过这 3 种模型分别构建出汽车行驶工况曲线后，我们将其与总体样本进行了误差分析，并选取其中相对最优的结果进行了 AutoEncoder 再度优化。该模型四得到的优化结果仍旧要进行误差分析，对行驶工况的准确性进行验证，以确定最终的汽车行驶工况曲线。

7.2 方法原理

7.2.1 随机搜索

随机搜索方法是指从整个采集数据里随机搜索“运动学片段”，组成候选工况，直到达到规定的时间长度为止[5]。这种方法虽然简单但计算复杂，因为构建的候选工况很多，我们需要从众多的候选工况中选取最为合适的代表性行驶工况。因此，所有候选工况都需进行运动特征评估体系的计算与比较。

7.2.2 主成分分析

主成分分析方法在数据降维方面的应用颇多，它是用较少的几个综合指标代替原来较多的变量指标，而且使这些较少的综合指标既能尽量多地反映原来较多变量指标所反映的信息，同时它们之间又是彼此独立的，通常用于原始数据存在信息交叉重叠、数据冗余等问题的处理[6]。假设用 p 个变量来描述研究对象，分别用 X_1, X_2, \dots, X_p 来表示，这 p 个变量构成的 p 维随机向量为 $\mathbf{X} = (X_1, X_2, \dots, X_p)^t$ 。设随机向量 X 的均值为 μ ，协方差矩阵为 Σ 。假设 X 是以 n 个标量随机变量组成的列向量，并且 μ_k 是其第 k 个元素的期望值，即， $\mu_k = E(X_k)$ ，协方差矩阵然后被定义为： $\Sigma = E\{(X - E[X])(X - E[X])^t\}$

$$= \begin{bmatrix} E(X_1 - \mu_1)(X_1 - \mu_1) & E(X_1 - \mu_1)(X_2 - \mu_2) & \dots & E(X_1 - \mu_1)(X_n - \mu_n) \\ E(X_2 - \mu_2)(X_1 - \mu_1) & E(X_2 - \mu_2)(X_2 - \mu_2) & \dots & E(X_2 - \mu_2)(X_n - \mu_n) \\ E(X_n - \mu_n)(X_1 - \mu_1) & E(X_n - \mu_n)(X_2 - \mu_2) & \dots & E(X_n - \mu_n)(X_n - \mu_n) \end{bmatrix} \quad (1)$$

对 X 进行线性变化，得到考虑原始变量的线性组合：

$$\begin{cases} Z_1 = \mu_{11}X_1 + \mu_{12}X_2 + \dots + \mu_{1p}X_p \\ Z_2 = \mu_{21}X_1 + \mu_{22}X_2 + \dots + \mu_{2p}X_p \\ \vdots \\ Z_p = \mu_{p1}X_1 + \mu_{p2}X_2 + \dots + \mu_{pp}X_p \end{cases} \quad (2)$$

主成分是不相关的线性组合 Z_1, Z_2, \dots, Z_p ，并且 Z_1 （第一主成分）是 X_1, X_2, \dots, X_p 的线性组合中方差最大者， Z_2 （第二主成分）是与 Z_1 不相关的线性组合中方差最大者， \dots ， Z_p （第 p 主成分）是与 Z_1, Z_2, \dots, Z_{p-1} 都不相关的线性组合中方差最大者。

利用主成分分析方法进行特征参数降维处理的流程如下：

第一步：设估计样本数为 m ，选取的指标数为 p ，则由估计样本的原始数据可得矩阵 $\mathbf{X} = (X_{ij})_{m \times p}$ ，其中 X_{ij} 表示第 i 秒第 j 项特征参数指标的值。

第二步：为了消除各项指标之间在量纲化和数量级上的差别，对指标数据进行标准化，得到标准化矩阵。

第三步：根据标准化数据矩阵建立协方差矩阵 R ，是反映标准化后的数据之间相关关系密切程度的统计指标，值越大，说明有必要对数据进行主成分分析。其中， $R_{ij}(i, j = 1, 2, \dots, p)$ 为原始变量 X_i 与 X_j 的相关系数。 R 为实对称矩阵（即 $R_{ij} = R_{ji}$ ），只需计算其上三角元素或下三角元素即可，其计算公式为：

$$R_{ij} = \frac{\sum_{k=1}^n (X_{kj} - \bar{X}_j)(X_{ki} - \bar{X}_i)}{\sqrt{\sum_{k=1}^n (X_{kj} - \bar{X}_j)^2 (X_{ki} - \bar{X}_i)^2}} \quad (3)$$

第四步：进行 KMO 检验及 Barlett 球形检验。其中 KMO 检验是取样的适当性检验，主要是检验所选择的变量是否包含了导致相关的根本性变量。通常情况下，如果 KMO 值小于 0.5 时，那就不太适合开展主成分分析；如果 KMO 值大于 0.6，则为“效果平庸”；如果 KMO 值大于 0.7，则为“中度适宜”；如果 KMO 值大于 0.8，则为“效果良好”。Barlett 球形检验是判读相关系数矩阵与单位阵是否有显著差异。如果相关系数矩阵近似为单位矩阵，则变量之间彼此正交，进行主成分分析没有意义。该处使用卡方检验来判断连个矩阵之间的差异。如果检验通过，说明两者之间有显著的差异，可以开展主成分分析。

第五步：检验通过后，根据协方差矩阵 R 求出特征值、主成分贡献率和累计方差贡献率，确定主成分个数。根据选取的主成分个数的原则，得到特征值大于 1、特征根变化出现突变点之前、累计贡献率达 85% 以上的特征值 $\lambda_1, \lambda_2, \dots, \lambda_m$ 所对应的主成分 $1, 2, \dots, q (q \leq p)$ ，其中整数 q 即为主成分的个数。

第六步：计算得到主成分载荷矩阵。成分矩阵显示的是主成分 Z_i 与原始指标 X_i 的相关系数 $R(Z_i, X_i)$ ，揭示了主成分与各原始变量之间的相关程度。主成分得分系数矩阵则等于主成分载荷矩阵转置矩阵的逆矩阵或逆矩阵的转置矩阵，也即成分得分系数矩阵的数值是主成分载荷除以相应的特征根得到的结果。在此基础上即可计算各主成分得分，也即各标准化原始变量与对应主成分得分系数的乘积之和。

7.2.3 核化主成分分析

主成分分析方法假定输入的数据服从高斯分布。当输入数据不服从高斯分布时，方法面临着许多问题。核化主成分分析 (Kernel PCA) 算法是一种新的特征提取方法，它是利用核技巧对经典的主分量分析法 (PCA) 进行的一种非线性推广。对于服从任意分布的观测数据 x ，Kernel PCA 首先利用一个非线性映射函数 Φ 将观测数据从输入空间映射到特征空间 F 中，使得 $\Phi(x)$ 近似高斯分布，然后再对映射数据做 PCA。因为特征空间很可能是高维空间，且 Kernel PCA 算法可表示成 F 空间中样本点的内积形式，所以它巧妙地应用了核函数而回避了求取非线性映射 Φ 的艰巨任务 [7]。同时，PCA 不进行分类的动作，而只做数据预处理，Kernel PCA 则在将非线性可分的数据通过非线性映射概率转换到一个高维空间中，在高维空间中使用 PCA 将其映射到另一个低维空间中后，还通过线性分类器对样本进行划分。与主成分分析法相比，Kernel PCA 具有能有效捕捉数据的非线性特征、对原始空间中数据的分布情况没有要求、计算复杂度相较于经典的 PCA 基本没有增加等优点，这使得它在很多领域得到了广泛应用。

核函数是通过两个向量点积来度量向量间相似度的函数。常用函数有：径向基核函数 (Gaussian RBF)、多项式核 (Polynomial)、双曲正切核 (Sigmodial) 等。

$$\begin{aligned} \text{Gaussian RBF} \quad & \mathbf{K}(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{c}\right) \\ \text{Polynomial} \quad & \mathbf{K}(\mathbf{x}, \mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y}) + \theta)^d \\ \text{Sigmodial} \quad & \mathbf{K}(\mathbf{x}, \mathbf{y}) = \tanh(\mathbf{g}(\mathbf{x} \cdot \mathbf{y}) + \theta) \end{aligned} \quad (4)$$

本文的 Kernel PCA 模型中的核函数在多次试验比较后选定为非常典型的径向基核函数 (Gaussian RBF)。

总体而言，Kernel PCA 求主成分步骤如下：

(1) 计算核矩阵 K ：

$$\mathbf{K}_{i,j} = (\Phi(x_i) \cdot \Phi(x_j)) = \mathbf{k}(x_i, x_j) \quad (5)$$

其中， $k=1,2,\dots,M$ ， \mathbf{K} 是一个 $M \times M$ 的核矩阵。

(2) 采用奇异值分解(SVD)算法求 \mathbf{K} 的特征值和特征向量；

(3) 对特征值以降序排列($\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_M$)，对前 p 个非零特征值对应的特征向量按如下公式进行规范化：

$$1 = \sum_{i,j=1}^M \boldsymbol{\alpha}_i^k \boldsymbol{\alpha}_j^k (\Phi(x_i) \cdot \Phi(x_j)) = \sum_{i,j=1}^M \boldsymbol{\alpha}_i^k \boldsymbol{\alpha}_j^k \mathbf{K}_{ij} = (\boldsymbol{\alpha}^k \cdot \mathbf{K} \boldsymbol{\alpha}^k) = \lambda_k (\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) \quad (6)$$

即在特征空间 \mathbf{F} 中须使 $\lambda_k (\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) = 1$ 。其中， λ_k 表示第 k 个非零特征值，相应的 \mathbf{K} 的特征向量按其特征值排列为 $\boldsymbol{\alpha}^1, \dots, \boldsymbol{\alpha}^p, \dots, \boldsymbol{\alpha}^M$ 。

(4) 计算一个测试样本 x 在特征向量上的投影值(主成分)并作为新特征：

$$(\mathbf{V}^k \cdot \Phi(x)) = \sum_{i=1}^M \boldsymbol{\alpha}_i^k (\Phi(x_i) \cdot \Phi(x)) = \sum_{i=1}^M \boldsymbol{\alpha}_i^k \mathbf{k}(x_i, x) \quad (7)$$

对于一个新样本 x ，提取它的主成分只需将其 \mathbf{F} 空间中相应的映射样本 $\Phi(x)$ 向 \mathbf{V}^k 做投影。

7.2.4 K 均值聚类

聚类是一种建立分类的多元统计分析方法，它能够将一批样本(或变量)数据根据其诸多特征，按照在性质上的亲疏程度在没有先验知识的情况下进行自动分类，产生多个分类结果。类内部的个体在特征上具有相似性，不同类间个体特征的差异性较大。包括基于划分、基于密度的分类等，如 k-means、k-medoids、DBSCAN 算法等。其中，k 均值聚类是最著名的无监督划分聚类算法，因其简洁和高效率，使用非常广泛。

K 均值聚类方法的流程如下[8]：

第一步：选 K 个初始聚类中心， $Z_1(1), Z_2(1), \dots, Z_K(1)$ ，其中括号内的序号为寻找聚类中心的迭代运算的次序号。聚类中心的向量值可任意设定，例如可选开始的 K 个模式样本的向量值作为初始聚类中心。

第二步：逐个将需分类的模式样本 $\{x\}$ 按最小距离准则分配给 K 个聚类中心中的某一个 $z_j(1)$ 。假设 $i=j$ 时， $D_j(k) = \min \{\|x - z_i(k)\|, i = 1, 2, \dots, K\}$ ，则 $x \in S_j(k)$ ，其中 k 为迭代运算的次序号，第一次迭代 $k=1$ ， S_j 表示第 j 个聚类，其聚类中心为 z_j 。

第三步：计算各个聚类中心的新的向量值， $Z_i(k+1)$ ， $i = 1, 2, \dots, K$ ，求各聚类域中所包含样本的均值向量：

$$z_j(k+1) = \frac{1}{N_j} \sum_{x \in S_j(k)} x, \quad j = 1, 2, \dots, K \quad (8)$$

其中 N_j 为第 j 个聚类域 S_j 中所包含的样本个数。以均值向量作为新的聚类中心，可使如下聚类准则函数最小：

$$J_j = \sum_{x \in S_j(k)} \|x - z_j(k+1)\|^2, \quad j = 1, 2, \dots, K \quad (9)$$

在这一步中要分别计算 K 个聚类中的样本均值向量，所以称之为 K 均值聚类算法。

第四步：若 $z_j(k+1) \neq z_j(k), j = 1, 2, \dots, K$ ，则返回第二步，将模式样本逐个重新分类，重复迭代运算；若 $z_j(k+1) = z_j(k), j = 1, 2, \dots, K$ ，则算法收敛，计算结束。

7.2.5 BIRCH 层次聚类

当数据量较大时, K-means 聚类方法的运行时间会较长, 同时其对初始中心的选择要求较高, 对于离群点和孤立点的敏感性很大。在此基础上, 我们选择了适用于样本量较大、节约内存、实现聚类速度更快同时还能够识别噪音点, 对数据集进行初步分类的预处理的 BIRCH 聚类算法。其全称是利用层次方法的平衡迭代规约和聚类 (Balanced Iterative Reducing and Clustering Using Hierarchies), 它是用层次方法来聚类和规约数据 [9]。

BIRCH 算法利用了一个树结构来帮助我们快速的聚类，这个树结构类似于平衡 B+ 树，一般将它称之为聚类特征树 (Clustering Feature Tree，简称 CF Tree)。这棵树的每一个节点是由若干个聚类特征 (Clustering Feature，简称 CF) 组成。聚类特征数每个节点包括叶子节点都有若干个 CF，而内部节点的 CF 有指向孩子节点的指针，所有的叶子节点用一个双向链表链接起来，如图 11 所示。

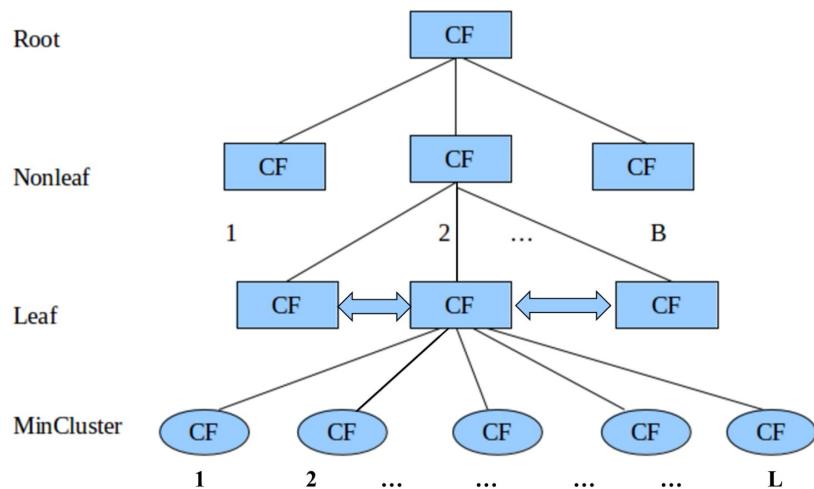


图 11 聚类特征树

在聚类特征树中，一个聚类特征 CF 是这样定义的：每一个 CF 是一个三元组，可以用 (N, LS, SS) 表示。其中 N 代表了这个 CF 中拥有的样本点的数量，这个好理解； LS 代表了这个 CF 中拥有的样本点各特征维度的和向量， SS 代表了这个 CF 中拥有的样本点各特征维度的平方和。CF 有一个很好的性质，就是

满足线性关系，也就是说：

$$CF1 + CF2 = (N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2) \quad (10)$$

这个性质从定义也很好理解。如果把这个性质放在 CF Tree 上，也就是说，在 CF Tree 中，对于每个父节点中的 CF 节点，它的 (N,LS,SS) 三元组的值等于这个 CF 节点所指向的所有子节点的三元组之和。

聚类特征树 CF Tree 的生成过程具体包括：

(1) 先定义好 CF Tree 的参数：即内部节点的最大 CF 数 B，叶子节点的最大 CF 数 L，叶节点每个 CF 的最大样本半径阈值 T。

(2) 从根节点向下寻找和新样本距离最近的叶子节点和叶子节点里最近的 CF 节点。如果新样本加入后，这个 CF 节点对应的超球体半径仍然满足小于阈值 T，则更新路径上所有的 CF 三元组，插入结束。否则转入 (3)。

(3) 如果当前叶子节点的 CF 节点个数小于阈值 L，则创建一个新的 CF 节点，放入新样本，将新的 CF 节点放入这个叶子节点，更新路径上所有的 CF 三元组，插入结束。否则转入 (4)。

(4) 将当前叶子节点划分为两个新叶子节点，选择旧叶子节点中所有 CF 元组里超球体距离最近的两个 CF 元组，分布作为两个新叶子节点的第一个 CF 节点。将其他元组和新样本元组按照距离远近原则放入对应的叶子节点。依次向上检查父节点是否也要分裂，如果需要按和叶子节点分裂方式相同。

将所有的训练集样本建立了 CF Tree，一个基本的 BIRCH 算法就完成了，对应的输出就是若干个 CF 节点，每个节点里的样本点就是一个聚类的簇。BIRCH 算法的关键就是 CF Tree 的生成，但除此之外，它还包括可以优化聚类结果的其他 3 个步骤：

(一) 将第一步建立的 CF Tree 进行筛选，去除一些异常 CF 节点，这些节点一般里面的样本点很少。对于一些超球体距离非常近的元组进行合并。

(二) 利用其它的一些聚类算法比如 K-Means 对所有的 CF 元组进行聚类，得到一颗比较好的 CF Tree。这一步的主要目的是消除由于样本读入顺序导致的不合理的树结构，以及一些由于节点 CF 个数限制导致的树结构分裂。

(三) 利用第三步生成的 CF Tree 的所有 CF 节点的质心，作为初始质心点，对所有的样本点按距离远近进行聚类。这样进一步减少了由于 CF Tree 的一些限制导致的聚类不合理的情况。

这些步骤也是 BIRCH 聚类算法性能更为优越的原因。本文在建立 CF Tree 的基础上，采用了它的其他 3 个步骤，值得指出的是，步骤(二)中我们选择采取的确实是 K-Means 方法先对 CF 元组进行初步聚类。

7.2.6 AutoEncoder 模型

AutoEncoder（自编码器）是神经网络的一种，其被广泛地用于特征降维以及无监督学习中。自编码器试图从简化编码中生成尽可能接近其原始输入的表示形式，从而达到学习数据表征的作用。自编码器可以被看作是前馈网络的一个特例。训练方法可以使用相同的技术，例如梯度下降法等。[10–12]

简单的自编码器结构如图12所示，输入 X 进编码器 f 后，得到简化编码 H ，而解码器 g 将 H 解码得到 X' 。编码器 f 和解码器 g 都可以用普通的神经网络

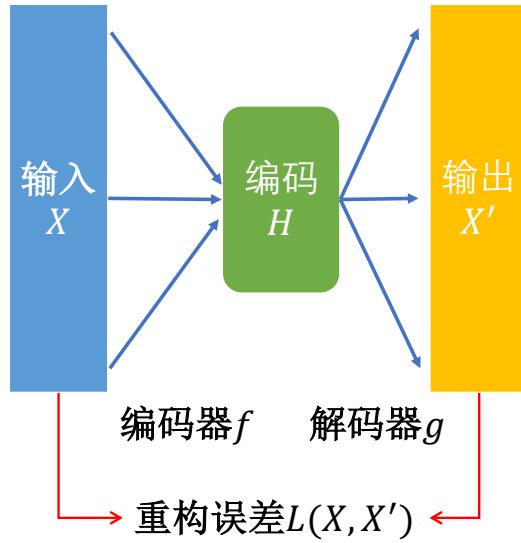


图 12 自编码器结构示意图

层来实现。网络训练的目标是最小化 X 与 X' 之间的重构损失（例如均方误差 MSE），即复原原始输入。

这里只是介绍了最基本的自编码器模型的原理。如果不加额外的约束，只会复原输入的自编码器通常没有实际意义。一般来说，会通过对得到的编码大小做出限制，或者增加额外的损失函数来使模型有特定含义。具体如何改造模型，将其应用在构建汽车工况曲线中，将在模型求解部分介绍。

7.3 特征评估体系的建立

汽车实际行驶工况是由一定数量的运动学片段组成的，代表性行驶工况则是从大量的运动学片段中选取合适的片段组合构建的。所以需要选用一些基本的特征参数反映每个运动学片段的行驶特征，从而评价最终构建的行驶工况曲线是否足够合理、有效并具有代表性。

关于运动学的特征参数 [13, 14] 有很多，其中最为重要的是速度和加速度，但是只利用这二者来描述是远远不够的，这样可能会导致信息丢失，造成整个过程的失真，严重影响构建结果的准确性。但是如果使用过多参数来刻画各个运动学片段，既会造成数据冗余，增大计算难度和计算量，也会给合理分析和解释问题造成困难。国内外不同的研究者选取的特征参数各不相同，本文在总结和参考现有相关文献的基础上，选取了 14 个最为典型的特征参数用以较为全面地描述和表征所有运动学片段，形成特征评估指标体系，包括平均速度 v_m 、平均行驶速度 v_{mr} 、最大速度 v_{max} 、(加速段) 平均加速度 a_{am} 、(减速段) 平均减速度 a_{dm} 、最大加速度 a_{max} 、最小减速度 a_{min} 、速度标准差 v_{sd} 、加速度标准差 a_{asd} 、减速度标准差 a_{dsd} 、怠速时间比 P_i 、匀速时间比 P_c 、加速时间比 P_a 、减速时间比 P_d 。这 14 项参数均参与了之后的数据降维等操作，但本文最后仅合理选取了其中 11 项进行汽车行驶工况曲线的误差评估。具体误差的计算公式请参考第 8.2.1 节。

7.4 特征评估参数的计算

特征评估参数的具体计算如下所示：

(1) 平均速度 v_m , 是指一段时间周期内, 汽车速度的算术平均值:

$$v_m = \frac{1}{N} \sum_{i=1}^N v_i / T \quad (11)$$

其中 T 为车辆运行时间, 平均速度 v_m 的单位要转换为 km/h 。

(2) 平均行驶速度 v_{mr} , 是指汽车在行驶状态下汽车速度的算术平均值, 即不包含汽车怠速状态:

$$v_m = \frac{1}{N} \sum_{i=1}^N v_i / (T - T_i) \quad (12)$$

其中 T 为车辆运行时间, T_i 为车辆怠速运行时间, 平均行驶速度 v_m 的单位要转换为 km/h 。

(3) 最大速度 v_{max}

$$v_{max} = \max \{v_i, i = 1, 2, 3, \dots, N\} \quad (13)$$

其中, v_i 为各个时刻的车辆运行速度, 最大速度 v_{max} 的单位为 km/h 。

(4)(加速段) 平均加速度 a_{am} , 即汽车在加速状态下各单位时间 (秒) 加速度的算术平均值。计算平均加速度、最大加速度等之前, 首先要明确加速度的计算公示:

$$a_{i,i+1} = \frac{v_{i+1} - v_i}{t_{i+1} - t_i} \times \frac{1000}{3600} = \frac{v_{i+1} - v_i}{3.6}, i = 1, 2, \dots, N-1 \quad (14)$$

其中, $a_{i,i+1}$ 代表第 i 秒到第 $i+1$ 秒的加速度 (单位为 m/s^2); v_i, v_{i+1} 代表第 i 秒和第 $i+1$ 秒的速度; t_i, t_{i+1} 代表第 i 秒和第 $i+1$ 秒的时刻。

在此基础上, 即可计算得到 (加速段) 平均加速度:

$$a_{am} = \frac{\sum a_i^+}{T_a^+} \quad (15)$$

其中, (加速段) 平均加速度 a_{am} 的单位为 m/s^2 , a_i^+ 为车辆运行过程中加速度 $a > 0.1 m/s^2$ 的加速度值, T_a^+ 为车辆加速总时间。

(5)(减速段) 平均减速度 a_{dm} , 即汽车在减速状态下各单位时间 (秒) 减速度的算术平均值:

$$a_{dm} = \frac{\sum a_i^-}{T_a^-} \quad (16)$$

其中, (减速段) 平均减速度 a_{dm} 的单位为 m/s^2 , a_i^- 为车辆运行过程中加速度 $a < -0.1 m/s^2$ 的减速度值, T_a^- 为车辆减速总时间。

(6) 最大加速度 a_{max}

$$a_{max} = \max \{a_i, i = 1, 2, 3, \dots, k\} \quad (17)$$

其中, 最大加速度 a_{max} 的单位为 m/s^2 , a_i 为加速状态时各个时刻的车辆加速度。

(7) 最小减速度 a_{min}

$$a_{min} = \min \{a_i, i = 1, 2, 3, \dots, N\} \quad (18)$$

其中，最小减速速度 a_{min} 的单位为 m/s^2 , a_i 为减速状态时各个时刻的车辆减速速度。

(8) 速度标准差 v_{sd} ，是指一段时间周期内，汽车速度的标准差(包括怠速状态):

$$v_{sd} = \sqrt{\frac{1}{k-1} \sum_{i=1}^k (v_i - v_m)^2}, i = 1, 2, \dots, k \quad (19)$$

其中， v_i 为各个时刻的车辆速度，速度标准差 v_{sd} 的单位为 km/h 。

(9) 加速度标准差 a_{asd} ，是指一段时间周期内，处在加速状态的汽车加速度的标准差：

$$a_{asd} = \sqrt{\frac{1}{p-1} \sum_{i=1}^p (a_i - a_{am})^2}, i = 1, 2, \dots, p \quad (20)$$

其中， a_i 为车辆加速状态时各个时刻对应的加速度，单位为 m/s^2 ， p 为处于加速状态的样本数。

(10) 减速度标准差 a_{dsd} ，是指一段时间周期内，处在减速状态的汽车减速度的标准差：

$$a_{dsd} = \sqrt{\frac{1}{q-1} \sum_{i=1}^q (a_i - a_{dm})^2}, i = 1, 2, \dots, q \quad (21)$$

其中， a_i 为车辆减速状态时各个时刻对应的减速度，单位为 m/s^2 ， q 为处于减速状态的样本数。

(11) 怠速时间比 P_i ，定义为一段时间周期内，车辆运行处于怠速状态的累计时间长度占该时间周期总时间长度的百分比：

$$P_i = \frac{T_i}{T} * 100\% \quad (22)$$

其中， T_i 为车辆行驶过程中处于怠速状态的行驶时间， T 为车辆行驶时间。怠速时间比 P_i 的单位为%。

(12) 匀速时间比 P_c ，定义为一段时间周期内，车辆运行处于匀速状态的累计时间长度占该时间周期总时间长度的百分比：

$$P_c = \frac{T_c}{T} * 100\% \quad (23)$$

其中， T_c 为车辆行驶过程中处于匀速状态的行驶时间，匀速时间比 P_c 的单位为%。

(13) 加速时间比 P_a ，定义为一段时间周期内，车辆运行处于加速状态的累计时间长度占该时间周期总时间长度的百分比：

$$P_a = \frac{T_a}{T} * 100\% \quad (24)$$

其中， T_a 为车辆行驶过程中处于加速状态的行驶时间，加速时间比 P_a 的单位为%。

(14) 减速时间比 P_c , 定义为一段时间周期内, 车辆运行处于减速状态的累计时间长度占该时间周期总时间长度的百分比:

$$P_d = \frac{T_d}{T} * 100\% \quad (25)$$

其中, T_d 为车辆行驶过程中处于怠速状态的行驶时间, 怠速时间比 P_d 的单位为%。

7.5 模型求解

7.5.1 基于随机搜索方法构建行驶工况曲线

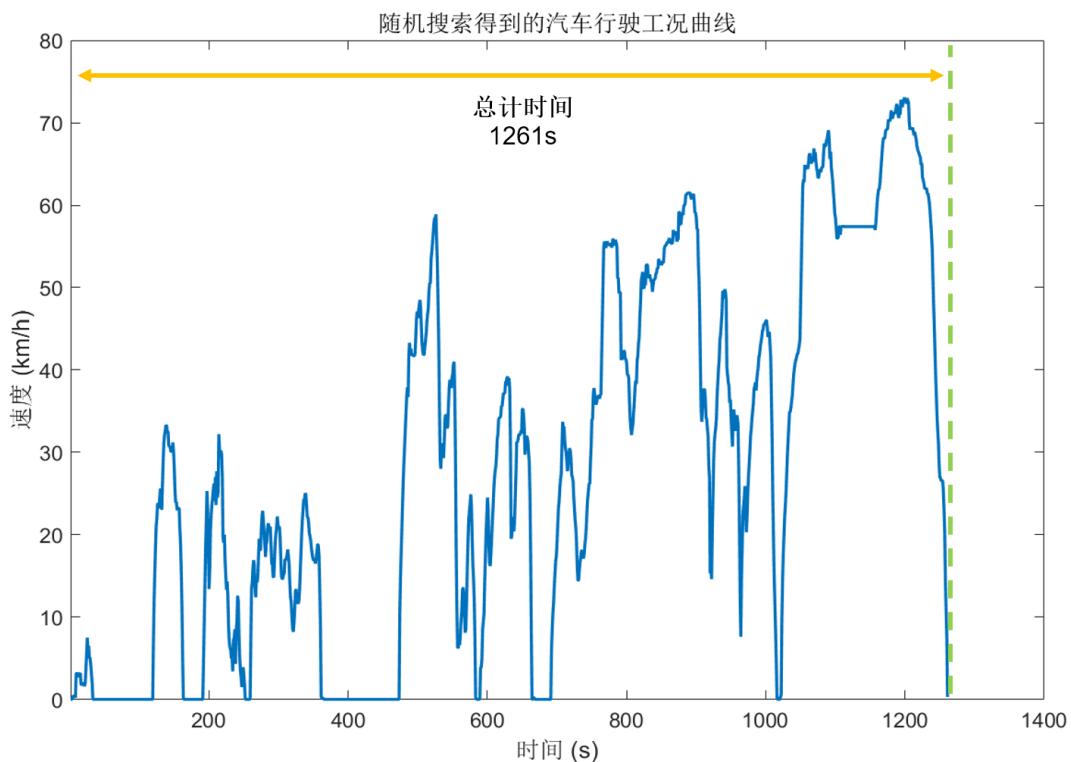


图 13 随机搜索法得到的工况曲线

根据题中规定, 需要构建一条 1200-1300 秒的汽车行驶工况曲线。在划分好 2088 个运动学片段之后, 我们计算出平均时间长度为 158 S, 大约为 8 个运动学片段进行组合。采用随机搜索方法, 我们对所有运动学片段进行随机组合, 得到 500 个符合汽车工况时间约束的组合结果, 在此基础上, 我们对这 500 个组合分别计算了与原始数据的所有特征参数值的相对误差 (误差定义在第 8.2.1 节), 选取其中误差最小的曲线作为代表汽车行驶工况曲线。如果对 500 个符号要求的曲线的误差求平均, 误差率为 14.8%, 数值较大。如果是误差最小的曲线, 则误差率为 4.3%。这一结果说明了由于搜索空间太大, 组合的质量随机性很大, 得到的最优工况也不算特别理想。

7.5.2 基于 PCA 和 K 均值聚类模型构建行驶工况曲线

PCA 降维过程及结果

在进行主成分分析之前，我们首先对特征参数数据进行了标准化处理，在此基础上，对特征参数标准化值先进行 KMO 检验及 Barlett 球形检验。结果显示，变量相关矩阵为非正定矩阵，无法进行 KMO 检验及 Barlett 球形检验，这提示我们变量之间相关性过高(都是速度、加速度变量，难以避免)。通过试验，删除 6 个变量之后可以满足条件，但因为这 6 个变量基本都是题中要求的或者必须的特征变量，综合考虑后我们对这些变量进行保留，直接进行之后的主成分分析操作。

解释的总方差表给出了特征值、主成分的方差贡献率与主成分方差的累积贡献率，如图 14 所示。主成分得分的方差，在数值上等于相关系数矩阵的各个特征根 λ ，全部特征根的总和则等于变量数目。提取平方和载入显示我们只提取了 $\lambda > 1$ 的主成分，其对应主成分方差的累计贡献率 $> 85\%$ 。我们也可以通过碎石图(图 15)大致发现特征根数值明显衰减点，主成分数目为 5，再结合之前的原 则，选取主成分数目为 5。在成分矩阵图 16 中，给出了主成分载荷矩阵，每一列

成份	解释的总方差					
	初始特征值			提取平方和载入		
	合计	方差的 %	累积 %	合计	方差的 %	累积 %
1	4.808	34.345	34.345	4.808	34.345	34.345
2	2.855	20.389	54.734	2.855	20.389	54.734
3	2.208	15.771	70.504	2.208	15.771	70.504
4	1.380	9.854	80.359	1.380	9.854	80.359
5	1.018	7.269	87.628	1.018	7.269	87.628
6	.604	4.312	91.940			
7	.465	3.324	95.263			
8	.261	1.861	97.124			
9	.149	1.067	98.191			
10	.122	.872	99.063			
11	.086	.613	99.676			
12	.037	.265	99.940			
13	.008	.060	100.000			
14	-6.228E-016	-4.449E-015	100.000			

提取方法：主成份分析。

图 14 主成分方差贡献率

载荷值都显示了各个变量与有关主成分的相关系数。如下，0.919 实际上就是主成分 1 与变量 3 平均速度的相关系数(解释程度系数)。同时，表中也给出了已提取了 5 个成分的提示。

随后计算主成分载荷矩阵转置矩阵的逆矩阵或逆矩阵的转置矩阵，也即主成分载荷除以相应的特征根得到的结果，从而得到对应成分得分系数矩阵的数值，如图 17 所示。主成分得分通过原始数据标准化结果与各自对应的成分得分系数相乘并累加即可求得。

通过主成分分析，我们总共选取了 5 个主成分分量，主成分方差的累计贡献

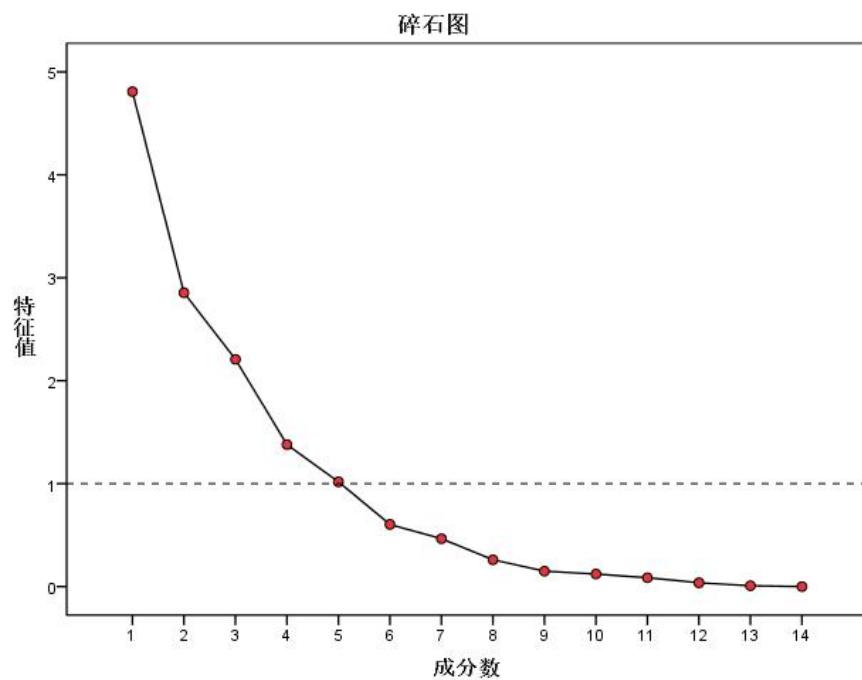


图 15 特征根数值衰减图

成份矩阵^a

	成份				
	1	2	3	4	5
Zscore(V3)	.919	-.267	-.110	.133	.057
Zscore(V4)	.885	-.042	-.348	.253	.015
Zscore(V5)	.914	-.009	-.260	.235	-.037
Zscore(V6)	.002	.609	.585	.175	-.026
Zscore(V7)	-.255	-.730	.013	.380	.210
Zscore(V8)	.415	.332	.380	.484	.063
Zscore(V9)	-.641	-.427	-.092	.351	-.224
Zscore(V10)	.813	.142	-.376	.245	-.133
Zscore(V11)	.064	.478	.689	.360	.064
Zscore(V12)	.470	.649	.052	-.480	.140
Zscore(V13)	-.539	.590	-.534	.255	-.026
Zscore(V14)	.622	-.324	.253	-.443	-.437
Zscore(V15)	.222	-.528	.697	.084	-.314
Zscore(V16)	.287	-.428	.241	-.149	.763

提取方法 : 主成份。^a

a. 已提取了 5 个成份。

图 16 成分矩阵

	成份得分系数矩阵				
	成份				
	1	2	3	4	5
Zscore(V3)	.191	-.093	-.050	.097	.056
Zscore(V4)	.184	-.015	-.157	.183	.015
Zscore(V5)	.190	-.003	-.118	.170	-.036
Zscore(V6)	.000	.213	.265	.127	-.026
Zscore(V7)	-.053	-.256	.006	.276	.206
Zscore(V8)	.086	.116	.172	.351	.062
Zscore(V9)	-.133	-.150	-.042	.254	-.220
Zscore(V10)	.169	.050	-.170	.178	-.131
Zscore(V11)	.013	.167	.312	.261	.063
Zscore(V12)	.098	.228	.024	-.348	.138
Zscore(V13)	-.112	.207	-.242	.185	-.026
Zscore(V14)	.129	-.114	.115	-.321	-.430
Zscore(V15)	.046	-.185	.315	.061	-.309
Zscore(V16)	.060	-.150	.109	-.108	.750

提取方法 : 主成份。

构成得分。

图 17 成分得分系数矩阵

率为 87.63。为了直观的展现主成分分析的结果，我们绘制了第一主成分和第二主成分的散点图如图 18。

K 均值聚类过程及结果

通过主成分分析降维后，我们的原始特征矩阵转化为具有 5 个特征分量的新特征矩阵。针对新特征矩阵，我们使用 Kmeans 算法对其进行聚类。由于无监督聚类算法并没有直接的标签用于评估聚类的效果，因此我们决定使用簇内的稠密程度和簇间的离散程度来评估聚类的效果并据此选择超参数类别 K。我们选用的指标是 Calinski-Harabasz 分数，它的计算公式为：

$$s(k) = \frac{tr(B_k)(m - k)}{tr(W_k)(k - 1)} \quad (26)$$

其中 m 为训练集样本数， k 为类别数， B_k 为类别之间的协方差矩阵， W_k 为类别内部数据的协方差矩阵， tr 为矩阵的迹。当类别内部数据的协方差越小，类别之间的协方差越大时，Calinski-Harabasz 分数越高，聚类效果也就越好。经计算，我们发现当聚类类别数为 4 时，Calinski-Harabasz 分数值为 563.3；当聚类类别数为 3 时，Calinski-Harabasz 分数值为 600.3。因此，我们决定将现有数据聚为 3 类。

聚类结果可视化如图 19 所示，2088 个短行程被聚为三类，第一类包括 534 个运动学片段，第二类包括 750 个运动学片段，第三类包括 804 个运动学片段。

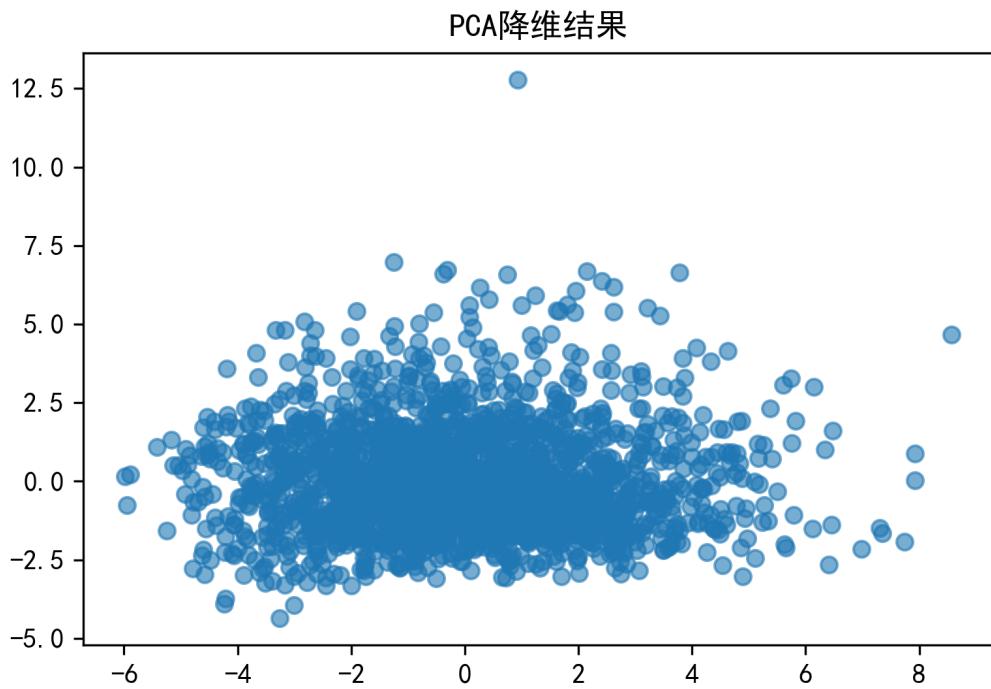


图 18 PCA 降维结果可视化

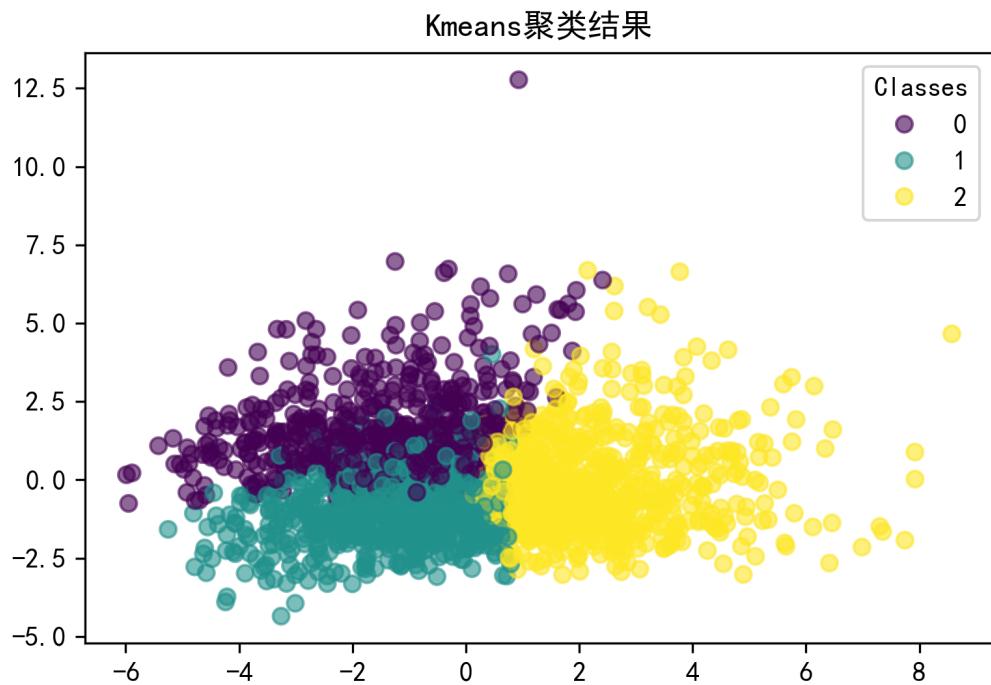


图 19 Kmeans 聚类结果可视化

我们对这三类运动学片段的特征参数进行了计算，计算结果如表 2。从表中结果来看，这三类运动片段按平均速度可划分为低速，中速，高速三个阶段，但每一段又呈现出不同的统计特征：（1）第一类运动片段平均速度最低，怠速比例最高。这主要反映出车辆在道路拥堵状况下断断续续低速行驶的行驶状态。（2）第二类运动片段的平均速度和怠速比例据处于中等水平，加速、减速、匀速比例相对平衡。这主要反映出车辆在道路较为通畅状况下中低速行驶状态。（3）第

三类运动片段的平均速度最高，怠速比例最低，加速和匀速比例之和显著高于另外两类。这主要反映出车辆在道路通畅状况下中高速行驶状态。此外，我们选取了前 1000 个运动学片段绘制了三类运动学片段关于速度和加速度的二维散点图，通过图 20，我们也能得出类似上述的结论。

表 2 三类运动片段特征参数计算结果

符号	总样本	第一类	第二类	第三类
v_m	27.24	10.40	15.87	37.20
v_{mr}	33.98	22.70	18.45	41.50
v_{max}	109.90	71.80	55.40	109.90
a_{am}	0.43	0.48	0.43	0.42
a_{dm}	-0.55	-0.66	-0.48	-0.56
a_{max}	3.96	3.94	3.94	3.96
a_{min}	-7.50	-7.08	-6.06	-7.50
v_{sd}	23.03	14.22	11.68	23.36
a_{asd}	0.37	0.40	0.37	0.37
a_{dsd}	0.57	0.61	0.42	0.61
P_i	0.20	0.54	0.14	0.10
P_a	0.32	0.20	0.33	0.36
P_d	0.26	0.16	0.31	0.28
P_c	0.22	0.10	0.22	0.25

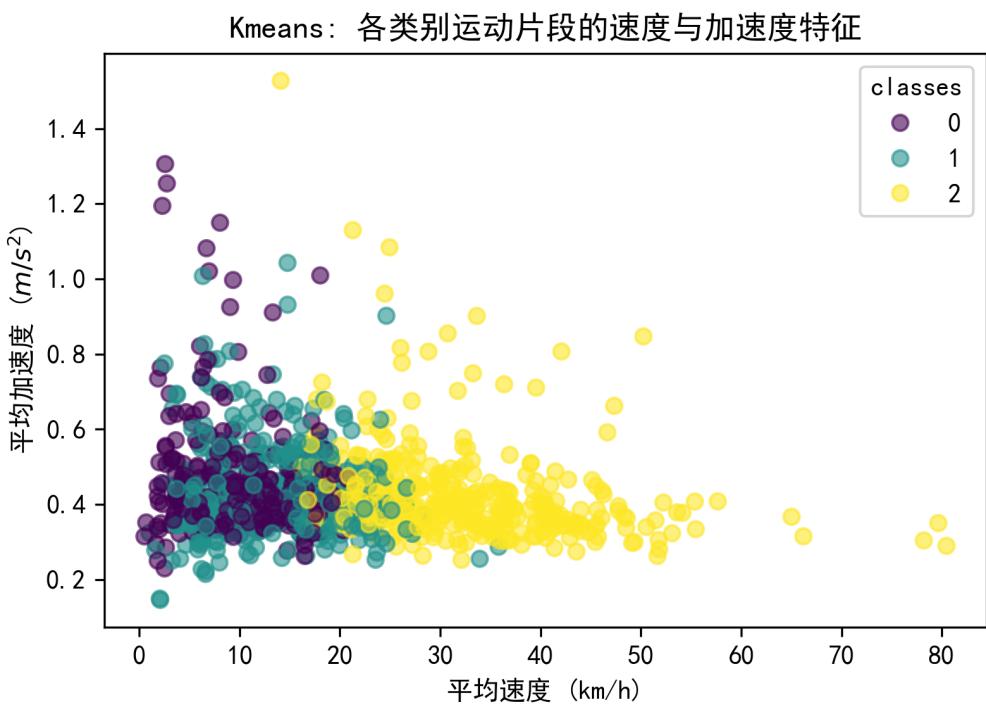


图 20 Kmeans: 各类别运动片段的速度和加速度特征散点图

构建行驶工况曲线

经过 Kmeans 聚类之后，我们将原始数据划分为三类，形成了三类运动学片段库。之后，我们采取从每类运动学片段库中选取运动学片段进行组合的方式来构建车辆的行驶工况曲线。具体过程如下：

首先，计算各类运动学片段库在整个运动学片段库所占时间比例，然后结合题目要求的车辆行驶工况曲线的时长 1200 秒，计算每类运动学片段在最终行驶工况曲线中所占的时间，计算公式如下：

$$t_i = \frac{T_i}{T_a} * t_r \quad (27)$$

其中， t_i 表示第 i 类运动学片段在最终行驶工况曲线中所占的时间， T_i 表示第 i 类运动学片段库总共持续的时间之和， T_a 表示全部运动学片段持续的时间之和， t_r 表示题目要求的车辆行驶工况曲线的时长，此处为 1200。计算结果如表 3 所示。

表 3 三类运动学片段在待求工况中的时间占比和时长

类别	时间占比 (%)	时长 (s)
第一类	19.87	238.46
第二类	44.57	534.82
第三类	35.56	426.72
待求工况	100	1200

根据计算结果，我们需要从每类运动学片段库中挑选出符合时间要求的具有代表性的运动学片段进行拼接，拼接后的曲线即为最终所要求的汽车行驶工况曲线。

在挑选具有代表性的运动学片段时，目前已有算法多采用文献 [15] 的方式，具体过程为：首先在三类运动学片段库中挑选出每一类具有代表性的运动学片段作为聚类中心，之后求取运动学片段与聚类中心的距离，然后按照距离最小的原则挑选出符合时间要求的运动学片段。如果当前运动学片段时间与已挑选片段之和超过最大时长要求（超出的时长不超过 33 秒也可接受，因为题目中要求行驶工况的总时长为 1200 至 1300），则剔除当前运动学片段，验证下一片段是否符合要求。否则，加入当前片段直至满足最终时间要求时不再加入片段。

这种直观的方式能够求得题解，但仅是局部最优解。这主要因为聚类本身是一种无监督的方法，在聚类的过程中会存在着不定因素，因此各样本点与聚类中心的距离事实上不完全对应于构建行驶工况权限时的误差大小。

因此，我们决定将随机搜索的方法引入，具体过程为：首先计算各运动学片段与其对应的聚类中心的距离，每一类按照距离最小的原则挑选出 σ 个片段（本文 $\sigma = 80$ ）作为候选集，然后每次从这三类候选集中随机搜索，选取符合时间要求的运动学片段构建关于行驶工况曲线的目标集，最后根据目标集中每条曲线的误差大小选取误差最小的曲线作为最终曲线。

和已有算法相比，我们的算法引入了更高的自由度，这使得模型有一定的概率跳出局部最优解，搜索到更好的解。我们将两种方法求得的行驶工况曲线分别

进行了误差分析发现，已有文献的方法能够搜索出一个固定的解，和总体试验数据相比，此解的误差率为 9.40%。而采用我们的方法，能够搜索出多个候选目标解集，这个解集的平均误差率为 10.29%，最小误差率仅为 **3.22%**。

此外，无论是在平均还是最小误差率，跟直接从所有运动片段中随机搜索（分别是 14.8% 和 4.36%）比，该方法都有了很大的提升。这充分说明了我们的方法确实能有效减少搜索空间，并且增加寻找到优化解的概率。

在本小节中，我们主要采用了主成分分析降维和 K 均值聚类的方法来挑选出具有代表性同时又符合时间要求的运动学片段，最后将这些片段组合成所要求的行驶工况曲线。基于这种方法所构建的行驶工况曲线如图 21。

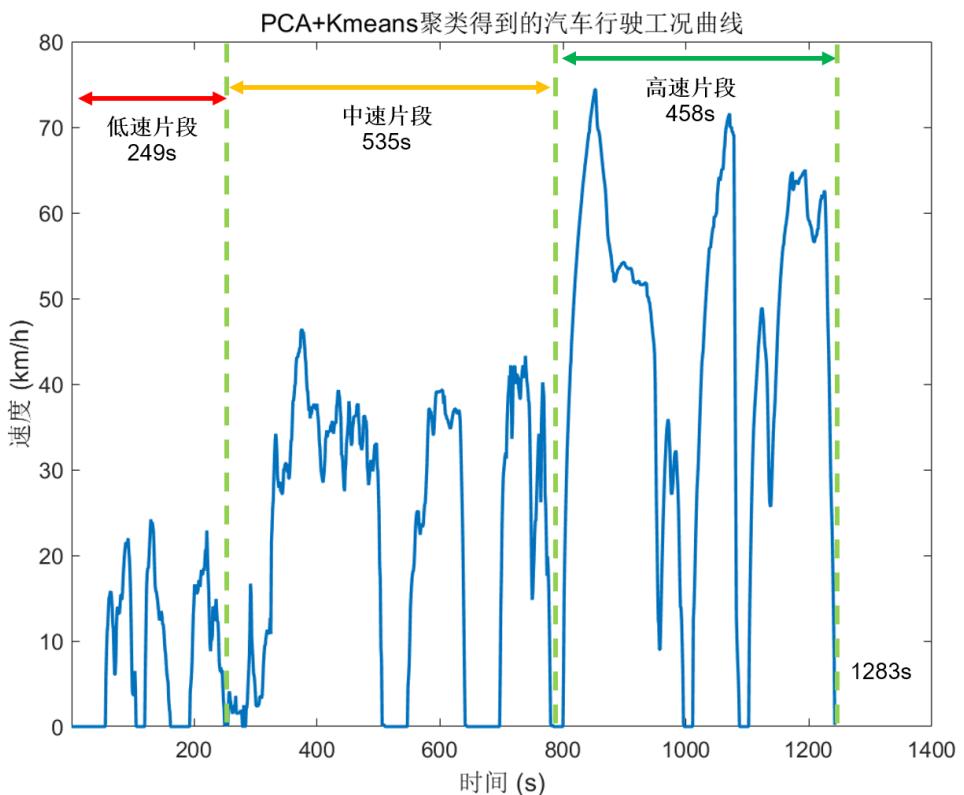


图 21 基于 PCA 和 Kmeans 聚类模型构建的行驶工况曲线

7.5.3 基于 Kernel PCA 和 BIRCH 聚类模型构建行驶工况曲线

Kernel PCA 降维过程及结果

传统的主成分分析受限于数据的线性降维，而核主成分分析 (kernel PCA) 则可实现数据的非线性降维，处理线性不可分的数据集。考虑到运动学片段的复杂性，我们决定利用核主成分分析对原始数据进行非线性降维。我们采用的核函数是径向基函数核 (RBF 核函数)。为了与传统线性主成分分析进行结果对比，我们也选取了 5 个主成分分量，并绘制了关于第一主成分和第二主成分的散点图 22。与上一节的线性 PCA 相比较，我们发现 KernelPCA 将第二主成分所占的比重增大了。

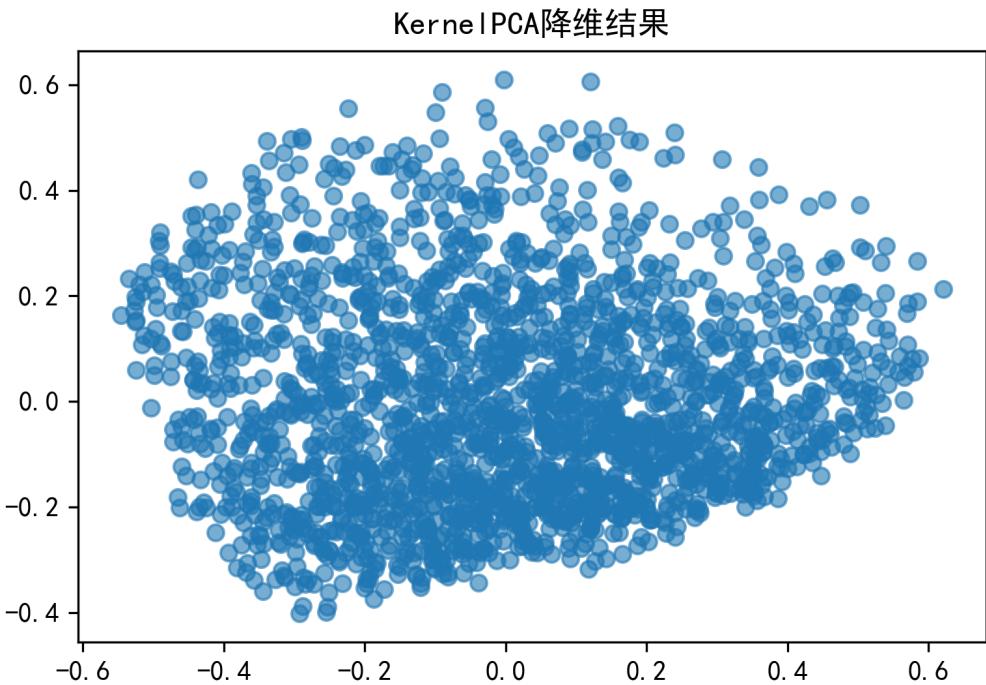


图 22 KernelPCA 降维结果可视化

BIRCH 聚类过程及结果

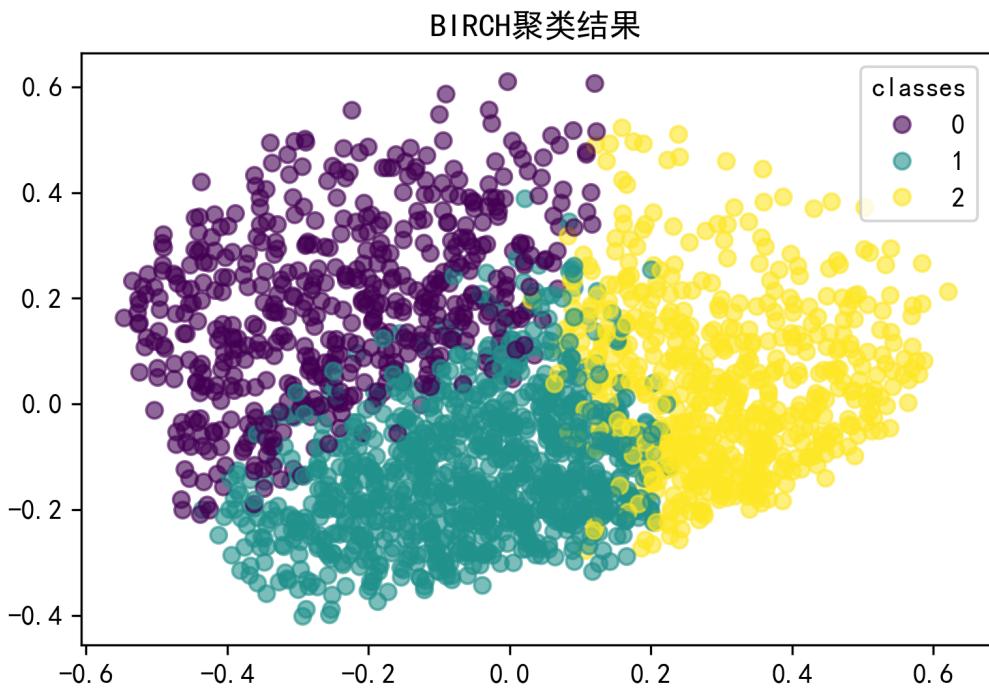


图 23 BIRCH 层次聚类结果可视化

BIRCH 层次聚类相比于简单的 Kmeans 聚类对离群点和孤立点更加鲁棒。通过计算可得关于 BIRCH 层次聚类的 Calinski-Harabasz 分数值为 644.13，相比于 Kmeans 聚类有了很大的提升，这初步表明了 KernelPCA 降维和 BIRCH 层次聚类的有效性。

聚类结果可视化如图 23 所示，2088 个短行程被聚为三类，第一类包括 538 个运动学片段，第二类包括 959 个运动学片段，第三类包括 591 个运动学片段。我们对这三类运动学片段的特征参数进行了计算并绘制了关于速度和加速度的二维散点图，如表 4 和图 24。通过观察，我们也能得出类似上一节 kmeans 处的结论。

表 4 三类运动片段特征参数计算结果

符号	总样本	第一类	第二类	第三类
v_m	27.24	9.55	18.56	39.57
v_{mr}	33.98	21.28	21.36	44.25
v_{max}	109.90	62.20	61.40	109.90
a_{am}	0.43	0.46	0.45	0.41
a_{dm}	-0.55	-0.62	-0.53	-0.56
a_{max}	3.96	3.89	3.94	3.96
a_{min}	-7.50	-6.53	-6.81	-7.50
v_{sd}	23.03	13.58	13.13	23.91
a_{asd}	0.37	0.38	0.38	0.37
a_{dsd}	0.57	0.56	0.48	0.63
P_i	0.20	0.55	0.13	0.11
P_a	0.32	0.19	0.35	0.36
P_d	0.26	0.16	0.31	0.27
P_c	0.22	0.10	0.20	0.27

构建行驶工况曲线

经过 BIRCH 聚类之后，我们将原始数据划分为三类运动学片段库，它们的时长占比为表 5 所示。

表 5 三类运动学片段在待求工况中的时间占比和时长

类别	时间占比 (%)	时长 (s)
第一类	24.45	293.35
第二类	39.91	478.93
第三类	35.54	427.72
待求工况	100	1200

根据计算结果，我们从每类运动学片段库中挑选出符合时间要求的具有代表性的运动学片段进行拼接，拼接后的曲线即为最终所要求的汽车行驶工况曲线。我们采用候选集结合随机搜索的方法，最终搜索出多个候选目标解集，这个解集的平均误差率为 8.81%，最小误差率仅为 2.94%，这再次表明了 KernelPCA 降维和 BIRCH 层次聚类的有效性。

BIRCH: 各类别运动片段的速度与加速度特征

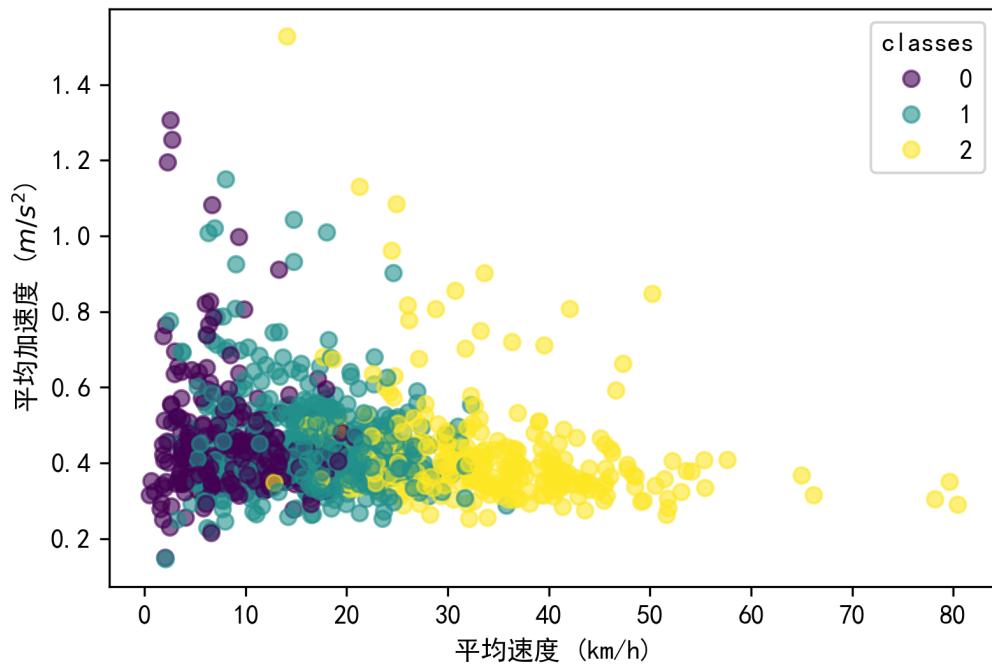


图 24 BIRCH: 各类别运动片段的速度和加速度特征散点图

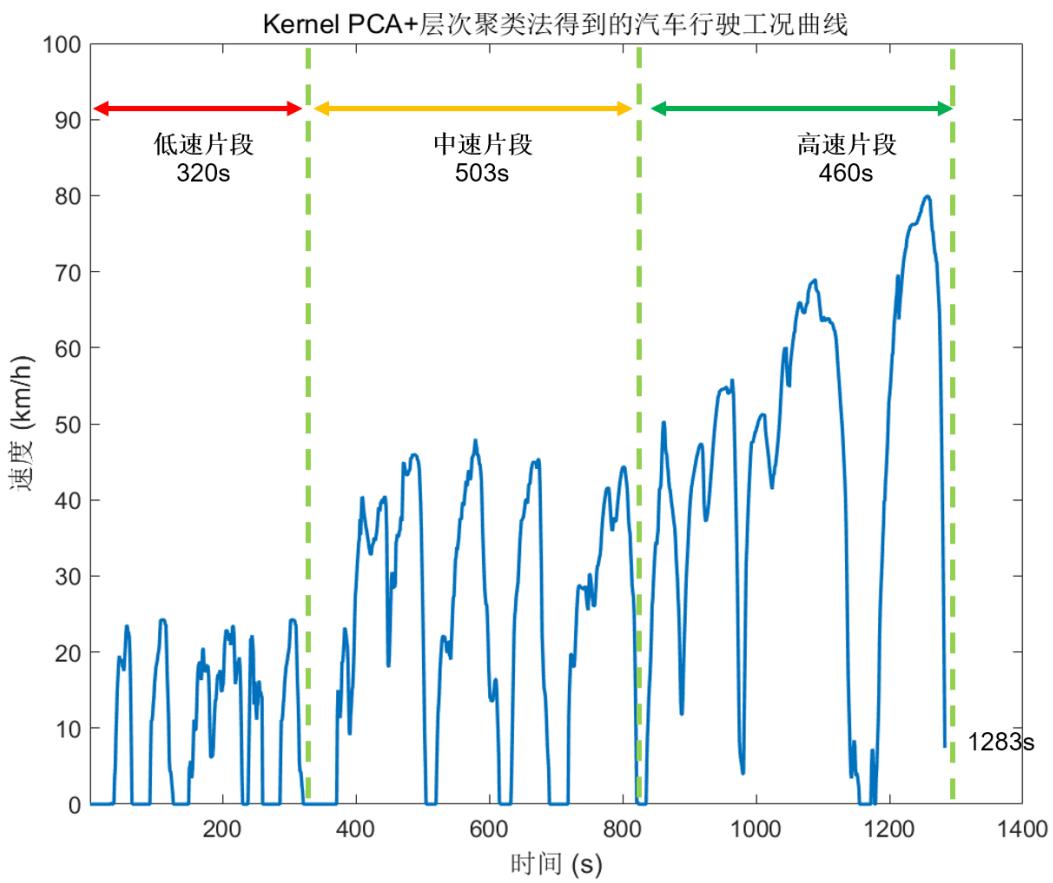


图 25 使用 Kernel PCA+ 层次聚类生成的行驶工况曲线

在本小节中，我们主要采用了核主成分分析降维和 BIRCH 层次聚类的方法来挑选出具有代表性同时又符合时间要求的运动学片段，最后将这些片段组合成所要求的行驶工况曲线。基于这种方法所构建的行驶工况曲线如图 25。

7.5.4 基于 AutoEncoder 模型优化汽车行驶工况曲线

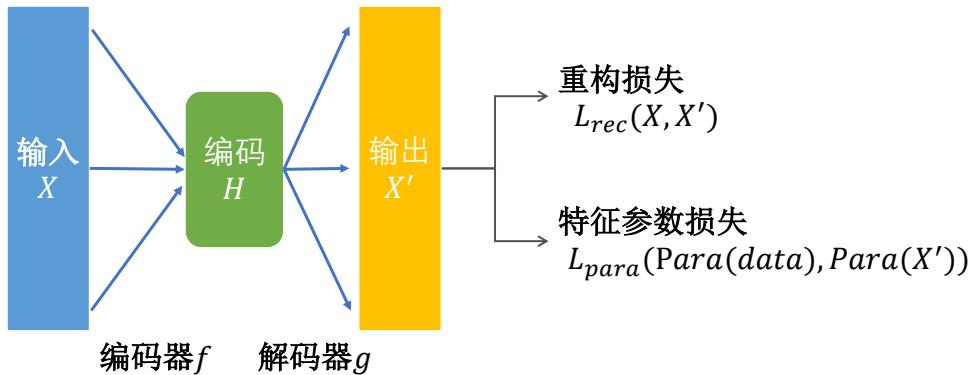


图 26 基于 AutoEncoder 模型的汽车工况优化模型

考虑问题三，其目标是寻找能体现参与数据采集汽车行驶特征的汽车行驶工况曲线。曲线的特点一是要符合真实分布，二是尽量最小化工况曲线和数据源之间的特征误差。

基于降维和聚类方法构建的汽车行驶工况，是使用数据中提取出的运行学片段来组合的。该结果满足来自于真实数据分布，同时特征的平均误差已经相对较小。但是，无法保证它一定是最优的（误差最小）。因此，我们猜想：能不能在由传统方法得到的初始行驶工况的基础上，进行微小的调整。调整的目标是使调整后的工况特征，与源数据特征之间的误差最小化。

而自编码器的一个优势恰恰在于可以加入其它的目标函数，与重构误差共同训练，使输出满足特定目标的同时尽可能接近输入数据。具体来说，可以将初始工况输入自编码器，经过编解码后输出调整后工况。目标是使其输出尽可能接近自己的同时，最小化其与原数据特征参数的误差。具体模型的示意图为图26。

假设初始汽车工况为 $X = x_1, x_2, \dots, x_T$ ，其中 x_t 是在 t 时刻的速度。则图中26的 L_{rec} 为 MSE 损失：

$$L_{rec} = \frac{1}{T} \times \sum_{t=1}^T (x_t - x'_t)^2 \quad (28)$$

特征参数误差损失 $L_{para}(P^d, P')$ 如下式所示：

$$L_{para}(P^{data}, P') = \frac{1}{M} \sum_{i=1}^M W_i |P_i^{data} - P'_i| \quad (29)$$

由第7.3节可求出源数据的特征参数 $P^{data} = Para(data)$ ，以及调整后行驶工况的特征参数 $P' = Para(X')$ 。这里的 P_i 代表了第 i 个特征参数， W_i 表示该

项特征的权重，是可以调节的超参数。最后总的目标函数是两个损失函数相加得到的：

$$L = L_{rec} + \lambda L_{para} \quad (30)$$

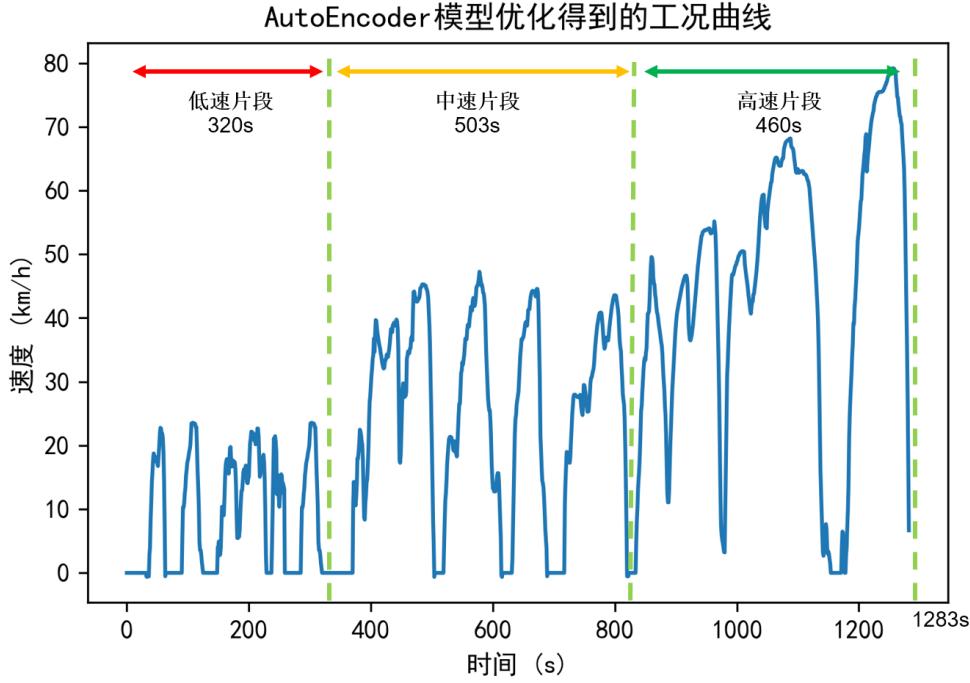


图 27 AutoEncoder 模型优化后得到的行驶工况

λ 是用于均衡两项损失的超参数。实际上，这里的第一个损失控制生成的工况曲线与之前初始的工况相差不至于太远，避免生成只符合特征参数但是无意义的速度曲线。另一方面，第二个损失项用于细微地调节工况曲线，使其特征与实验数据更加接近。

具体实现中，我们使用了 kernel PCA+ 层次聚类得到的工况曲线作为初始工况（即输入）。使用了单隐层的全连接神经网络来构造自编码器，输入层与输出层的神经元个数设置为初始工况的时间 T 。训练使用了标准的 SGD 算法迭代共 200 轮，之后网络收敛。

训练后得到的优化工况曲线如图27所示，由于重构误差的存在，优化后的工况曲线与初始工况曲线差别不是很大，直接看图难以分辨。为了更好地说明两者的区别，以及自编码器是如何优化的，我们可视化了优化后的工况与初始工况的差值（图28）。

从图28中可以看出，自编码器主要的优化策略是降低了非急速段的速度，以及其余一些十分微小的调整。而观察初始工况的特征可知，其平均速度 $v_m = 27.9 km/h$ ，而实验数据总体的平均速度 $v_m = 27.2 km/h$ 。也就是说，初始工况的速度是偏大的，自编码器寻找到的主要策略是平均地减少原来的速度，这一策略是十分符合人类直觉的。

此外，模型还进行了较难解释的微小调整，由之后的分析可知，这些调整是可以降低其他特征（比如加速度标准差）误差的。最后得到的工况曲线，平均误差率下降到了 2.43%。

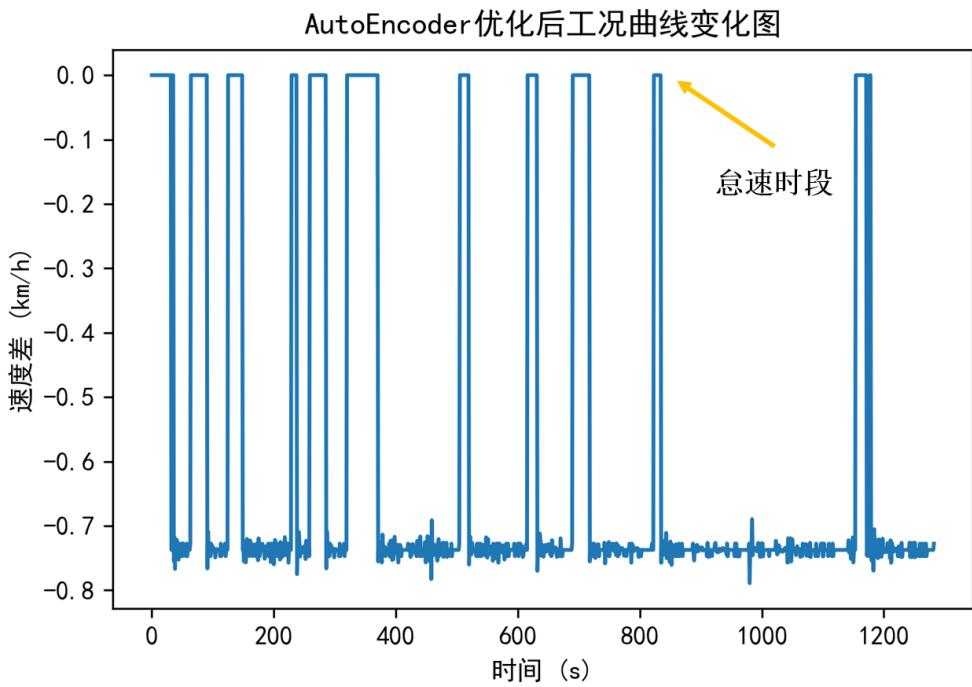


图 28 AutoEncoder 模型优化前后的速度变化

值得注意的是，我们是在平均误差已经很小的情况下优化的，此时可调整的空间很小。如果初始工况的误差率更大，则误差率下降的幅度会更大。例如，如果是随机搜索得到的工况作为初始工况，下降幅度可以达到 1%。

8. 结果分析与比较

8.1 各方法所得汽车行驶工况的特征参数

表 6 四种方法所得行驶工况的特征参数以及运行时间统计

符号	总样本	随机搜索	PCA+Kmeans	KernelPCA+BIRCH	Autoencoder
T	331829	1261	1238	1283	1283
v_m	27.24	28.46	26.96	27.91	27.25
v_{mr}	33.98	36.18	33.51	34.97	34.24
v_{max}	109.90	73.10	61.50	79.90	79.20
a_{am}	0.43	0.46	0.44	0.43	0.43
a_{dm}	-0.55	-0.58	-0.57	-0.59	-0.59
a_{max}	3.96	2.86	2.14	3.22	3.11
a_{min}	-7.50	-3.25	-3.25	-3.89	-3.78
v_{sd}	23.03	22.93	19.67	22.50	22.41
a_{asd}	0.37	0.35	0.36	0.33	0.34
a_{dsd}	0.57	0.57	0.54	0.58	0.58
P_i	0.20	0.21	0.20	0.20	0.20
P_a	0.32	0.31	0.33	0.32	0.32
P_d	0.26	0.26	0.26	0.26	0.26
P_c	0.21	0.22	0.22	0.22	0.22

表6显示了四种方法以及源实验数据（处理后数据）的特征参数，以及时间的统计结果。首先根据时间，可以看出都是符合1200s-1300s的区间要求的。此外，四类方法的特征参数与总样本特征都十分接近，说明构建的曲线基本都能满足代表实验数据的要求。

8.2 各方法所得汽车行驶工况的评估结果

8.2.1 特征参数误差定义

构建了汽车行驶工况后，需要对其与原始数据的差异进行验证。由于根据采样理论，最大速度、最大加速度和最大减速度这3个特征值损耗一定会较大。因此只选取了其他11项特征参数的相对误差来评估，误差计算公式如下：

$$E_i = \frac{|P_i^{data} - P_i|}{|P_i^{data}|} \times 100\% \quad (31)$$

其中， E_i 表示了工况曲线特征参数 P_i 与实验数据特征参数 P_i^{data} 之间的相对误差。公式中的 i 表示第 i 个特征参数。行驶工况的最终误差，是各项特征参数相对误差的平均值，即公式32所示。

$$E_{final} = \frac{1}{M} \sum_{i=1}^M E_i \quad (32)$$

8.2.2 各个方法所得结果的误差比较与分析

表 7 四种方法的特征参数的误差计算结果 (%)

符号	随机搜索	PCA+Kmeans	KernelPCA+BIRCH	Autoencoder
E_{vm}	4.49	1.01	2.48	0.06
E_{vmr}	6.47	1.38	2.93	0.76
E_{aam}	6.47	1.89	0.40	0.55
E_{adm}	3.82	2.78	6.71	6.14
E_{vsd}	0.46	14.59	2.31	2.71
E_{aasd}	7.12	3.88	11.09	10.23
E_{adsd}	0.36	6.26	1.13	0.94
E_{pi}	7.51	1.48	1.74	1.74
E_{pa}	5.08	0.94	0.06	0.06
E_{pd}	2.46	0.34	2.36	2.36
E_{pc}	3.71	0.38	1.21	1.21
E_{final}	4.36	3.18	2.95	2.43

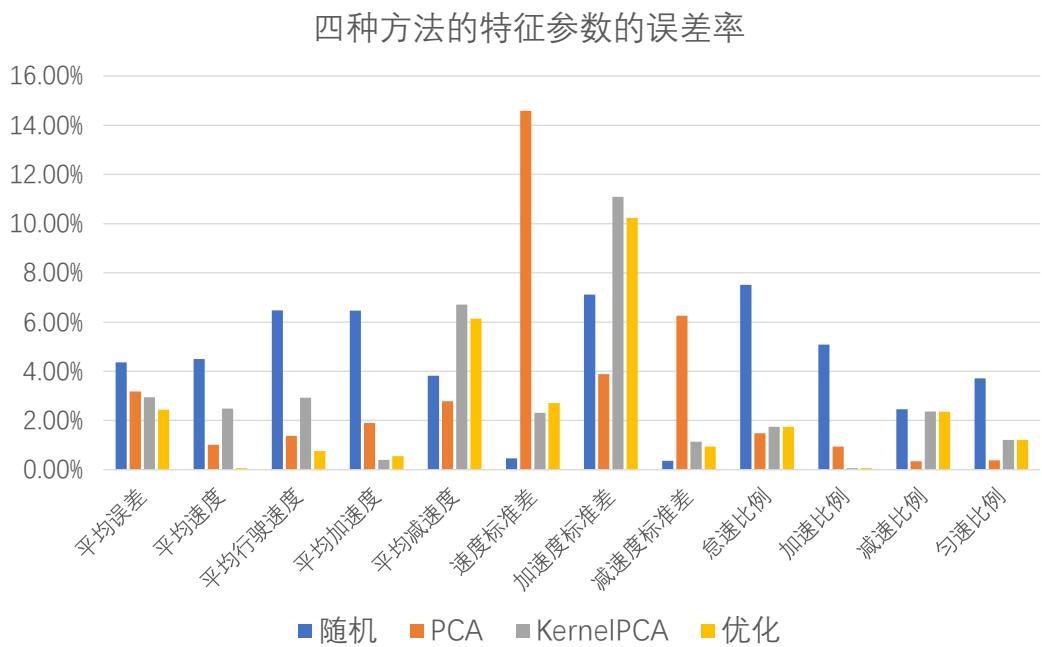


图 29 四种方法的特征参数的误差率

表7和图29对比了四类方法的特征误差。首先从平均误差 E_{final} 来看，所有

的方法误差率均小于 5%，与实际数据特征十分接近，构造的汽车工况合理有效。具体到每类方法：

- (1) 随机搜索的误差为 4.36%，是所有方法中数值最大的。
- (2) 之后 PCA+Kmeans 方法的误差降到了 3.18%，说明了聚类分析能够有效地减少搜索空间，并且增加了寻找到优化解的概率。
- (3) 加上核函数以及层次聚类后，误差进一步降低到了 2.95%。证明了核函数有助于将特征映射到更好的子空间，层次聚类产生的分类结果更有代表性。
- (4) 基于自编码器的优化方法，能够有效降低本来已经较低的误差，特别是关于平均速度和平均行驶速度的误差。此外，其他参数误差也有小幅度的降低。

综合各方面表现，各个方法的分析如下：

- (1) 随机搜索的好处是简单容易理解，精度勉强可以达到要求。但是随机性太大，得到的行驶工况曲线精度没有保障。
- (2) 聚类分析法的好处在于构造出的曲线代表性好，误差率低。并且明确地知道结果中的运动片段的类型，分别对应了现实生活的低速、中速和高速三种情况，可解释性好。
- (3) 自编码器方法通过梯度直接优化工况曲线的特征参数，使其能更符合要求。该方法适合于已经存在一条较好的工况曲线的情况。

9. 模型评价与改进

9.1 模型的优点

1. 数据预处理部分我们针对不同的数据错误或误差情况进行了全面的分类处理，在各类处理之下又进行了情况细分针对性处理。例如在数据缺失的情况下，先判断缺失段是否为短于 5 秒，满足条件则先进行插补，不满足条件再剔除，使得数据信息在最大程度上得到了保留及优化。

2. 在进行运动学片段划分时，我们在定义的基础上增加了合理的筛选条件，同时还考虑到车辆发动及停止时存在速度数据偏移的情况，对此类数据的前端或者后端再次进行了删除操作。这使得本文的运动学片段划分相对更为准确。

3. 在构建汽车行驶工况路线时，特征评估参数过多，不便于计算且数据冗余，特征参数过少，则难以准确地描述运动学片段状态，曲线构造容易失真。我们在总结、分析现有相关文献的指标体系后，选取了较为完善、合适的 14 项特征参数（14 项参数均参与了降维分析，但仅合理使用了其中 11 项用于评估曲线误差）。

4. 模型选择方面，我们逐步优化，从基础的随机搜索到 PCA 降维-K 均值聚类到改进后的 Kernel PCA 降维-BIRCH 聚类，方法不断改善，精度不断提高。

5. 在已有工况曲线的基础上，采用 AutoEncoder 模型对此前三种方法所得的相对最优结果再度进行优化，得到了更具有代表性的工况曲线。

6. 随机搜索模型一优势在于简单、易于理解和学习。模型二中 PCA 降维能够降低线性数据的冗余、重叠，K 均值聚类能够简单高效的实现小型数据的分类。模型三中 Kernel PCA 能够有效捕捉数据的非线性特征且对数据分布没有要求，BIRCH 聚类能够识别噪音、进行数据初步分类预处理，且适用于样本量较大的情况，节约内存、运行速度更快。AutoEncoder 模型四通过有意义的重构，实现对特征参数误差的进一步优化。

9.2 模型的缺点

1. 我们在模型中做出了许多假设，其中很多可能与现实存在分歧。例如，当预处理滤除的点显著影响了数据的分布，在这种情况下，拟合精度会大打折扣；聚类分析模型的前提假设是，所有运动片段都能按少数（本文采用三种）类别归纳。

2. 随机搜索模型一计算量很大，计算复杂。模型二中 PCA 模型无法处理非线性降维及分类，K 均值聚类对于初始中心点的选取过于敏感且难以应对大体量数据的分类。模型三中 Kernel PCA 虽然能够实现非线性降维，但计算成本更高，BIRCH 聚类倾向于链状聚类。AutoEncoder 模型四计算成本高，容易过拟合。

9.3 模型的改进与推广

本文提出的方法和模型可推广应用于其他城市汽车行驶工况路线的构建：

1. 文中数据预处理方法及操作因其分类细化且处理科学，在其他相关研究遭遇数据质量问题时均可对应采纳、使用。

2. 构建的特征参数评估体系对以后的其他相关研究具有较强的现实参考意

义。可根据具体情况对参数进行增减、优化。

3. 模型四可以推广到各类有初始较优值，但仍需优化的问题中，具有普遍的可参考性。若要得到更大的推广，增加的损失函数可以做特异性改进，不同的损失函数可以根据实际情况分配不同的权重。

10. 参考文献

- [1] KNEZ M, MUNEER T, JEREV B, et al. The estimation of a driving cycle for Celje and a comparison to other European cities[J]. Sustainable Cities and Society, 2014, 11(2-3): 56–60.
- [2] 仇多洋. 汽车行驶工况的构建及波动特性研究 [D]. [出版地不详]: 合肥工业大学, 2012.
- [3] SCHÖLKOPF B, SMOLA A, MÜLLER K-R. Kernel principal component analysis[C] // International conference on artificial neural networks. 1997: 583–588.
- [4] ZHANG T, RAMAKRISHNAN R, LIVNY M. BIRCH: an efficient data clustering method for very large databases[C] // ACM Sigmod Record: Vol 25. 1996: 103–114.
- [5] 姜平. 城市混合道路行驶工况的构建研究 [D]. [出版地不详]: 合肥工业大学, 2011.
- [6] LI Y, YU Z M, SONG X X. [Application of principal component analysis (PCA) for the estimation of source of heavy metal contamination in marine sediments][J]. Huan Jing Ke Xue, 2006, 27(1): 137–141.
- [7] SANG W C, LEE C, LEE J M, et al. Fault detection and identification of nonlinear processes based on kernel PCA[J]. Chemometrics and Intelligent Laboratory Systems, 2005, 75(1): 55–67.
- [8] TARIQ H, BURNEY S M A. K-Means Cluster Analysis for Image Segmentation[J]. International Journal of Computer Applications, 2014, 96(4): 1–8.
- [9] 李捷承, 陶耀东, 孙咏, 等. 基于 BIRCH 聚类的物流配送设施选址算法 [J]. 计算机系统应用, 2018, 27(9): 215–219.
- [10] GOODFELLOW I, BENGIO Y, COURVILLE A. Deep learning[M]. [S.l.]: MIT press, 2016.
- [11] LIOU C-Y, HUANG J-C, YANG W-C. Modeling word perception using the Elman network[J]. Neurocomputing, 2008, 71(16-18): 3150–3157.
- [12] LIOU C-Y, CHENG W-C, LIOU J-W, et al. Autoencoder for words[J]. Neurocomputing, 2014, 139: 84–96.
- [13] 姜平, 石琴, 陈无畏, 等. 基于小波分析的城市道路行驶工况构建的研究 [J]. 汽车工程, 2011, 33(1): 70–73.
- [14] 彭育辉, 杨辉宝, 李孟良, 等. 基于 K-均值聚类分析的城市道路汽车行驶工况构建方法研究 [J]. 汽车技术, 2017(11): 13–18.
- [15] 野晨晨, 张洪坤, 范鲁艳, 等. 沈阳市乘用车城市道路工况试验研究 [J]. 科学技术与工程, 2017, 17(21): 241–247.

附录 A 特征参数提取程序

```
feature_mat = ones(length(cell_section),16);

for i=1:length(cell_section)
    sp = cell_section{i,1};
    acc = cell_section{i,2};
    idl = cell_section{i,3};

    up_ind = acc>0.1& (~idl);
    down_ind = acc<-0.1&(~idl);
    cons_ind = (abs(acc)<=0.1) & (~idl);

    if(sum(up_ind)==0 || sum(down_ind)==0 || sum(cons_ind)==0)
        1
        continue
    end

    %运行时间
    T = length(sp);
    feature_mat(i,1) = T;
    %距离
    S = sum(sp/3.6)/1000;
    feature_mat(i,2) = S;
    % 平均速度
    mean_sp = (mean(sp));
    feature_mat(i,3) = mean_sp;
    %平均行驶速度
    run_sp = mean(sp(~idl));
    feature_mat(i,4) = run_sp;
    %最大速度
    max_sp = max(sp);
    feature_mat(i,5) = max_sp;
    %平均加速加速度
    mean_up_acc = (mean(acc(up_ind)));
    feature_mat(i,6) = mean_up_acc;
    if isnan(mean_up_acc)
        break
    end
    %平均加速减速度
    mean_down_acc = (mean(acc(down_ind)));
    feature_mat(i,7) = mean_down_acc;
    %最大加速度
    max_acc = max(acc);
    feature_mat(i,8) = max_acc;
    %最大减速度
    min_acc = min(acc);
    feature_mat(i,9) = min_acc;
    %速度标准差
    std_sp = std(sp);
    feature_mat(i,10) = std_sp;
    %加速度标准差
    std_up_acc = std(acc(up_ind));
```

```

feature_mat(i,11) = std_up_acc;
%减速度标准差
std_down_acc = std(acc(down_ind));
feature_mat(i,12) = std_down_acc;
%怠速时间比
idl_ratio = sum(idl)/length(idl);
feature_mat(i,13) = idl_ratio;

up_ratio = sum(up_ind)/length(idl);
feature_mat(i,14) = up_ratio;

down_ratio = sum(down_ind)/length(idl);
feature_mat(i,15) = down_ratio;

cons_ratio = sum(cons_ind)/length(idl);
feature_mat(i,16) = cons_ratio;
end

```

附录 B 误差计算程序

```

list =[661, 162, 593, 2011, 162, 1594, 792, 622, 1767, 1726,
       1872]
cycle = []
cycle_idl = []
cycle_acc= []
for i = 1:length(list)

    cycle = [cycle; cell_section{list(i)+1,1}];
    cycle_acc = [cycle_acc; cell_section{list(i)+1,2}];
    cycle_idl= [cycle_idl; cell_section{list(i)+1,3}];
end

sp = cycle;
acc = cycle_acc;
idl = cycle_idl;
up_ind = acc>0.1& (~idl);
down_ind = acc<-0.1&(~idl);
cons_ind = (abs(acc)<=0.1) & (~idl);

cycle_para = ones(1,16);
%运行时间
T = length(sp);
cycle_para(1) = T;
%距离
S = sum(sp/3.6)/1000;
cycle_para(2) = S;
% 平均速度
mean_sp = (mean(sp));
cycle_para(3) = mean_sp;
%平均行驶速度

```

```

run_sp = mean(sp(~idl));
cycle_para(4) = run_sp;
%最大速度
max_sp = max(sp);
cycle_para(5) = max_sp;
%平均加速加速度
mean_up_acc = (mean(acc(up_ind)));
cycle_para(6) = mean_up_acc;

%平均加速减速度
mean_down_acc = (mean(acc(down_ind)));
cycle_para(7) = mean_down_acc;
%最大加速度
max_acc = max(acc);
cycle_para(8) = max_acc;
%最大减速度
min_acc = min(acc);
cycle_para(9) = min_acc;
%速度标准差
std_sp = std(sp);
cycle_para(10) = std_sp;
%加速度标准差
std_up_acc = std(acc(up_ind));
cycle_para(11) = std_up_acc;
%减速度标准差
std_down_acc = std(acc(down_ind));
cycle_para(12) = std_down_acc;
%怠速时间比
idl_ratio = sum(idl)/length(idl);
cycle_para(13) = idl_ratio;
up_ratio = sum(up_ind)/length(idl);
cycle_para(14) = up_ratio;
down_ratio = sum(down_ind)/length(idl);
cycle_para(15) = down_ratio;
cons_ratio = sum(cons_ind)/length(idl);
cycle_para(16) = cons_ratio;

plot(cycle,'LineWidth',2)

% title('Kernel PCA+层次聚类法得到的汽车行驶工况曲线')
title('PCA+Kmeans聚类得到的汽车行驶工况曲线')
% title('典型的运动学片段曲线#1')
set(gca,'fontsize',14);
axis([-inf,1400 0 80])
xlabel('时间 (s)')
ylabel('速度 (km/h)')
cycle_para
para_list = [3 4 6 7 10:16];
mean(abs((cycle_para(para_list)-
stand_para(para_list))./stand_para(para_list)))
err = (abs((cycle_para(para_list)-
stand_para(para_list))./stand_para(para_list)))

```

附录 C 主成分分析与聚类

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.decomposition import PCA
import scipy.io as scio
import pandas as pd

data_path = './feature_array_v4.mat'
data = scio.loadmat(data_path)
feature_original = data['feature_mat']
feature = feature_original[:,2:]
print(feature.shape)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
feature = sc.fit_transform(feature)
pca = PCA(n_components=2)
pca.fit(feature)
new_feature = pca.transform(feature)

from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabaz_score
kmeans = KMeans(n_clusters=3, random_state=666).fit(new_feature)
kmeans_pre = kmeans.predict(new_feature)

fig, ax = plt.subplots(dpi=300)
scatter = ax.scatter(new_feature[:, 0], new_feature[:, 1],
alpha=.6, c=kmeans_pre[:])
legend = ax.legend(*scatter.legend_elements(), loc="upper
right", title="Classes")
# plt.figure()
plt.title('Kmeans聚类结果')
plt.savefig('Kmeans聚类结果.png')
plt.show()
print(calinski_harabaz_score(new_feature, kmeans_pre))
```

附录 D 邻域随机搜索

```
wait_part = []
wait_time = []
for i in range(3):
    similar = kmeans.transform(new_feature)
    t = kmeans.labels_ == i
    time_index = similar[:,i].argsort()[:80]
    t_label = feature_original[time_index, 0]
    wait_part.append(time_index)
    wait_time.append(list(t_label))
for i in range(3):
    all_time = np.mean(wait_time[i])
```

```

    print(result[i]/all_time)
final_result = []
final_error = []
for time in range(500):
    final_part = []
    final_time = []
    for i in range(3):
        tmp_part = []
        tmp_time = []
        count = 0
        while True:
            j = np.random.randint(0,80)
            tmp_time.append(wait_time[i][j])
            tmp_part.append(wait_part[i][j])
            if np.sum(tmp_time)>result[i] and
               np.sum(tmp_time)<(result[i]+33):
                break
            if (result[i]-np.sum(tmp_time)) < min(tmp_time):
                tmp_time.pop()
                tmp_part.pop()
            count += 1
            if count>50:
                tmp_part = []
                tmp_time = []
                count = 0
        final_part.extend(tmp_part)
        final_time.extend(tmp_time)
    para_list = [2, 3, 5, 6, 9, 10, 11, 12, 13, 14, 15]
    data_path="section_v4.mat"
    data = scio.loadmat(data_path)
    cell_section = data['cell_section']
    s_list = list(range(0,len(cell_section)))
    stand_para = compute(s_list)
    s_list = final_part
    cycle_para = compute(s_list)
    error = np.mean(np.abs((cycle_para[para_list]- \
                           stand_para[para_list])/stand_para[para_list]))
    final_result.append(final_part)
    final_error.append(error)
print(min(final_error))
print(np.mean(final_error))

```

附录 E AutoEncoder

```

import torch
import torch.nn as nn
import scipy.io as scio
import numpy as np

# Load the init cycle
data_path="32_cycle.mat"

```

```

data = scio.loadmat(data_path)
cycle = data['cycle']
cycle=cycle.reshape(-1)
para_list = [2, 3, 5, 6, 9, 10, 11, 12, 13, 14, 15]
stand = data['stand_para'].reshape(-1)[para_list]

cycle_idl = data['cycle_idl'].reshape(-1)
torch_idl = torch.ByteTensor(cycle_idl).reshape(1,-1)

# Autoencoder: refine the driving cycle
class refine_net(nn.Module):
    def __init__(self):
        super(refine_net, self).__init__()

        self.fc = nn.Linear(len(cycle) , len(cycle))
        self.out = nn.Linear(len(cycle) , len(cycle))

        nn.init.xavier_uniform_(self.fc.weight)
        nn.init.constant_(self.fc.bias, 0.1)

        nn.init.xavier_uniform_(self.out.weight)
        nn.init.constant_(self.out.bias, 0.1)
    def forward(self, x):
        x = torch.nn.functional.relu(self.fc(x))
        x = self.out(x)
        return x

def compute_para(sp) :
    sp = sp
    t_speed = sp*10/3.6
    acc =torch.ones_like(sp)
    acc[0,:-1] = t_speed[0,1:] - t_speed[0,:-1]

    up_ind = (acc>0.1)& (~idl)
    down_ind = (acc<-0.1)&(~idl)
    cons_ind = (abs(acc)<=0.1) & (~idl)

    # 平均速度
    mean_sp = (torch.mean(sp))

    #平均行驶速度
    run_sp = torch.mean(sp[~idl])

    #平均加速加速度
    mean_up_acc = (torch.mean(acc[up_ind]))

    #平均加速减速度
    mean_down_acc = (torch.mean(acc[down_ind]))

    #速度标准差
    std_sp = torch.std(sp);

    #加速度标准差
    std_up_acc = torch.std(acc[up_ind])

```

```

#减速度标准差
std_down_acc = torch.std(acc[down_ind])

#怠速时间比
idl_ratio = torch.sum(idl).float()/len(idl[0])
up_ratio = torch.sum(up_ind).float()/len(idl[0])
down_ratio = torch.sum(down_ind).float()/len(idl[0])
cons_ratio = torch.sum(cons_ind).float()/len(idl[0])

return [mean_sp, run_sp, mean_up_acc, mean_down_acc, std_sp,
        std_up_acc, std_down_acc, idl_ratio, up_ratio, down_ratio,
        cons_ratio]

def total_loss(x, rec_x, y):
    loss_mse = torch.nn.functional.mse_loss(x, rec_x)
    para = compute_para(x)
    loss_para = 0
    weight= [200,200 ,1,1,1,1,1,1,1,1,1]
    for i in range(11):
        loss_para = loss_para+
            weight[i]*torch.abs((para[i]-y[0][i])/y[0][i])
    return loss_mse, loss_para

x =torch.FloatTensor(cycle).reshape(1,-1)/10
x_rec =torch.FloatTensor(cycle).reshape(1,-1)/10
x.requires_grad=True
y = torch.FloatTensor(stand
    ).reshape(1,-1)/torch.Tensor([10,10,1,1,10,1,1,1,1,1])
idl = torch_idl

# Autoencoder
net = refine_net()
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)

# Iterations
for i in range(1,200):
    pred = net(x)
    loss_mse ,loss_para= total_loss(pred,x_rec,y)
    loss = loss_mse +0.001* loss_para
    loss = 1* loss
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    print('Train MSE Loss: {:.6f}'.format(loss_mse.item()))
    print('Train Para Loss: {:.6f}'.format(loss_para.item()))

```