

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»
Факультет электроники и вычислительной техники
Кафедра «Системы автоматизированного проектирования и поискового
конструирования»

Утверждаю

И.о. зав. кафедрой «САПР и ПК»

_____ М. В. Щербаков
(подпись) (инициалы, фамилия)

«_____» _____ 2016 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к _____ магистерской диссертации на тему
(наименование вида работ)
Метод кластеризации предпочтений жителей города по
перемещению

Автор _____ Чечеткин Илья Александрович
(подпись и дата подписания) (фамилия, имя, отчество)

Обозначение _____ МД-40461806-10.27-13-16.81
(обозначение документа)

Группа _____
(шифр группы)

Направление _____ 09.04.01 «Информатика и вычислительная техника»,
программа
(код по ОКСО, наименование направления, программы)

Руководитель работы _____ М. В. Щербаков
(подпись и дата подписания) (инициалы и фамилия)

Консультанты по разделам:

_____ (краткое наименование раздела) _____ (подпись и дата подписания) _____ (инициалы и фамилия)

_____ (краткое наименование раздела) _____ (подпись и дата подписания) _____ (инициалы и фамилия)

_____ (краткое наименование раздела) _____ (подпись и дата подписания) _____ (инициалы и фамилия)

Нормоконтролер _____ Н. П. Садовникова
(подпись, дата подписания) (инициалы и фамилия)

Волгоград 2016 г.

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»
Факультет электроники и вычислительной техники
Кафедра «Системы автоматизированного проектирования и поискового
конструирования»

Утверждаю

И.о. зав. кафедрой «САПР и ПК»

М. В. Щербаков

(подпись)

(инициалы, фамилия)

«_____» 2016 г.

Задание на магистерскую диссертацию
(наименование вида работ)

Студента Чечеткина Ильи Александровича
(фамилия, имя, отчество)

Код кафедры 10.27 Группа САПР-2п1

Тема Метод кластеризации предпочтений жителей города по перемещению

Утверждена приказом по ВолгГТУ от «__» 2016 г. №_____

Срок предоставления готовой работы _____
(дата, подпись студента)

Содержание основной части пояснительной записки

Перечень графических материалов

- 1) _____
- 2) _____
- 3) _____
- 4) _____
- 5) _____
- 6) _____
- 7) _____
- 8) _____
- 9) _____
- 10) _____

Руководитель работы _____
(подпись и дата подписания)

М. В. Щербаков
(инициалы и фамилия)

Консультанты по разделам:

(краткое наименование раздела)	(подпись и дата подписания)	(инициалы и фамилия)
(краткое наименование раздела)	(подпись и дата подписания)	(инициалы и фамилия)
(краткое наименование раздела)	(подпись и дата подписания)	(инициалы и фамилия)

Аннотация

Развитие городов сопряжено с модернизацией существующей сети общественного транспорта. Современные коммуникационные технологии позволяют собирать данные о перемещениях людей в городской среде, на основе которых можно сделать выводы о предпочтениях людей по перемещению внутри городской среды. Фактически, подобные предпочтения можно рассматривать как требования к структуре сети общественного транспорта. Имея данные о ежедневных передвижениях людей, мы можем понять реальные предпочтения и потребности людей в системе городского транспорта. Следовательно, может быть предложена модификация транспортной сети или даже набор возможных альтернатив. Эта модификация отражает реальные потребности людей и сокращает время передвижения и повышает общий уровень удовлетворенности. Чтобы произвести такие модификации необходимо сначала решить задачу кластеризации геораспределенных данных, учитывая при этом препятствия, лежащие между объектами данных, для получения сети предпочтительных остановочных пунктов общественного транспорта. В данной работе предлагаются методы, позволяющие на основе данных о предпочтениях жителей формировать схему остановочных пунктов общественного транспорта.

Список ключевых слов: геораспределенные данные, транспорт, общественный транспорт, остановочные пункты, кластеризация, кластеризация с учетом препятствий, кластеризация пространственных данных, k-means, анализ больших данных, поддержка принятия решений.

Abstract

Urban development is connected with the existing public transportation network modernization. Modern communication technologies make it possible to collect data on the movements of people in the urban environment, based on which is possible to draw conclusions about people's preferences on the movement inside the urban space. In fact, these preferences may be considered as requirements for the public transportation network. Having data about people's everyday movements, we can understand the real people preferences and needs in the urban transport system. Hence, as the results of analysis the modified transport network (or even a bunch of alternatives) can be suggested. This new solutions reflects real people needs and reduce transfer time and increase satisfaction level. To design the modification for transport network, we must first locate the preferable transport terminals. Actually, it's necessary to solve the geospatial clustering problem – clustering data with condition of existence of obstacles between data samples. The proposed methods allows to create a public transportation terminals network based on data about the transportation users preferences.

List of keywords: geospatial data, transport, public transport, transport terminals, clustering, clustering with obstructed distance, clustering geospatial data, big data analytics, decision support.

Содержание

Введение	8
1 Введение в проблему кластеризации геораспределенных данных	11
1.1 Анализ предметной области	11
1.2 Состояние современных исследований	13
1.3 Требования к решаемой задаче	18
1.4 Заключение	19
2 Методы кластеризации геораспределенных данных	20
2.1 Общее описание методов	21
2.1.1 Разделяющие методы	21
2.1.2 Иерархические методы	22
2.1.3 Плотностные методы	24
2.1.4 Сеточные методы	25
2.2 Проанализированные методы	25
2.2.1 Алгоритм Mean Shift	25
2.2.2 Алгоритм DBSCAN	29
2.2.3 Алгоритм BIRCH	32
2.3 Алгоритм k-means	34
2.3.1 Общее описание	35
2.3.2 Объяснение работы	37
2.4 Расчет расстояний между точками	38
2.5 Заключение	41
3 Испытание и обоснование эффективности предлагаемых подходов	42
3.1 Проектирование ПО	42
3.2 Методика проведения эксперимента	42
4 Описание результатов испытания	45
4.1 Проведение эксперимента и описание результатов	45
4.1.1 Последовательная реализация	46
4.1.2 Последовательная с использованием OSRM	46
4.1.3 Параллельная с использованием OSRM	46
4.1.4 Результаты	47
4.2 Обсуждение результатов	48
4.2.1 Выводы	49

Заключение	52
Список использованных источников	53
Приложение А Исходный код	58
Приложение Б Техническое задание	86

Введение

Изменения, происходящие в городской среде вследствие технического прогресса, требуют формирования новых методов планирования и развития инфраструктуры города для создания более комфортной жизни людей. Одной из наиболее важных сторон развития города является организация оптимальной транспортной системы города, в частности — системы общественного транспорта.

Несмотря на кажущуюся хаотичность перемещений жителей, все они подчиняются определенным закономерностям, связанных с масштабом и планировкой городской среды. Для принятия решений по изменению или созданию маршрутной сети города необходимо выявить эти закономерности в поведении населения, и по этим закономерностям сформировать обобщенную модель перемещения жителей внутри города, на основе которой можно будет строить и оценивать различные варианты системы городского общественного транспорта. В связи с этим, можно сформулировать научную проблему, связанную с совершенствованием маршрутной сети пассажирского транспорта на основе методов обработки больших данных о предпочтениях жителей по перемещению.

Актуальность. Изменения в городской среде требуют формирования новых механизмов планирования инфраструктуры города. Для получения эффективных результатов следует осуществлять принятие решений на основе актуальных данных, отражающих предпочтения жителей. В рамках магистерской диссертации следует разработать метод кластеризации предпочтений жителей города по перемещению.

Объект и предмет исследования. Объектом исследования магистерской работы является набор предпочтений жителей города по перемещению, выраженных в виде пары геораспределенных объектов «пункт отправления–пункт назначения», каждый из которых характеризуется двумя параметрами — координатами — широтой и долготой. Предметом исследования является метод кластеризации предпочтений жителей, учитывающий элементы рельефа местности.

Целью данной работы являлась разработка метода кластеризации геораспределенных данных с учетом рельефа местности.

Для достижения поставленной цели решались следующие задачи:

- 1) генерация псевдореалистичных данных о перемещениях жителей;

- 2) разработка метрики расстояний, учитывающей рельеф местности;
- 3) модификация и использование существующих алгоритмов для кластеризации географических данных, полученных из картографического сервиса OpenStreetMap [45], с разработанной метрикой расстояний;
- 4) представление построенных кластеров на карте.

Первая задача заключается в создании псевдореалистичных данных для замены отсутствующих реальных данных о перемещениях жителей на данный момент. Они нужны для работы над последующими задачами как приближенный аналог.

Вторая задача заключается в разработке метрики расстояний, которая учитывала бы рельеф местности — реки, автомобильные и железные дороги, жилые кварталы и др.

Третья задача заключается в анализе и модификации существующих алгоритмов кластеризации данных для работы с метрикой. Используемый алгоритм должен быть оптимален по времени работы и требуемой памяти для обработки большого количества данных.

Четвертая задача заключается в разработке web-приложения для визуализации построенных кластеров.

Научная новизна работы: разработан алгоритм кластеризации на основе k-means, использующий метрику расстояния, вычисленную с использованием карт городской местности, позволяющую учитывать городской рельеф.

В первой главе произведен анализ рассматриваемой предметной области, рассмотрено текущее состояние сложившейся проблемы и существующие методы, описанные в работах других исследователей, для решения данной ситуации.

Во второй главе рассмотрены и проанализированы существующие методы, применяемые для кластеризации. Подробно рассмотрено три алгоритма, которые можно применить для кластеризации геораспределенных данных, если использовать в них предложенную метрику расстояний и подобрать конкретные параметры управления под конкретные выборки. Также описан основной метод, используемый для кластеризации геораспределенных данных в данной работе, и предложенная метрика, учитывающая элементы городского рельефа.

В третьей главе описаны методология проектирования программного обеспечения и методика проведения эксперимента.

В четвертой главе описаны испытания разработанных алгоритмов, а также обсуждение полученных результатов в ходе эксперимента и вывод на их основе.

В заключении работы сформулированы общие выводы по проделанной работе.

1 Введение в проблему кластеризации геораспределенных данных

Развитие транспортной инфраструктуры города основывается на устаревших нормативах, не учитывающих стремительное увеличение личного транспорта и изменений функционального назначения городских пространств [1]. Это ведет к ухудшению транспортной ситуации, увеличению пробок, снижению качества транспортного обслуживания и, как следствие, к усилию неудовлетворенности жителей. Критичным фактором является игнорирование фактических предпочтений жителей по перемещению в городе при проектировании маршрутов общественного транспорта.

Для оптимизации маршрутной сети общественного транспорта необходимо проанализировать большой объем данных, характеризующих численность и мобильность населения, среднее время перемещения, расположение мест приложения труда и жилых массивов. Источниками этих данных выступают статистические сборники, выписки о численности сотрудников крупных предприятий, собираемые муниципальными предприятиями общественного транспорта, информация о количестве проданных билетах на маршрутах общественного транспорта. Для сбора данных о перемещениях жителей организуется целый комплекс мероприятий по натурному подсчету пассажиропотока в подвижном составе общественного транспорта и на остановочных пунктах существующих маршрутов, а также анкетированию жителей [2, 3]. Такие традиционные методы являются достаточно трудоемкими, а полученные данные не в полной мере отражают динамично меняющуюся ситуацию. В связи с этим, необходимо использовать современные технологии и новые ресурсы для получения актуальных данных о предпочтениях жителей по перемещениям в городе и интенсивности пассажиропотоков. Основываясь на современных подходах к анализу данных можно получить ценную информацию для поддержки принятия решений в процессе планирования развития транспортной системы города.

1.1 Анализ предметной области

Развитие городов приводит к модернизации существующей сети общественного транспорта. Управление городской инфраструктурой или городское планирование представляет собой набор сложных процедур, где решения могут оказывать сильное влияние на удовлетворенность жителей, в том числе — не удовлетворить всех. Новой проблемой для быстрорастущих городов ста-

новится создание надежной системы общественного транспорта. В идеале, оптимальная сеть городского общественного транспорта уменьшает время передвижения настолько, как если бы жители использовали личный транспорт.

Часто текущие системы планирования транспортной сети не принимают во внимание реальные предпочтения большинства жителей города. Это приводит к основным проблемам: общественным транспортом становится неудобно пользоваться, жители чаще начинают использовать личный транспорт для перемещения, из-за большего количества машин на дорогах начинают образовываться пробки, что приводит потерю личного времени и, как следствие, к повышению недовольства жителей.

В настоящее время повсеместный сбор данных и технологии обработки данных, которые становятся все более доступными, открывают новые возможности для проектировщиков систем общественного транспорта. Большая часть населения, живущего в городах, имеют мобильные телефоны или смартфоны. Операторы мобильной связи могут получать огромное количество данных о перемещении владельцев телефонов по городу. Исходя из предположения, что каждый человек имеет свои определенные «шаблоны перемещения» — набор часто используемых пунктов отправления–назначения, которые можно рассматривать как личные предпочтения по перемещению в городе, эти данные можно использовать для оценки текущей транспортной сети и ее модификации на основе предпочтений жителей.

Для решения данных проблем необходимо выполнить следующие процедуры: сбор данных, обработка данных, создание набора маршрутных сетей и выбор сети из набора в соответствии с критериями качества.

Сбор данных требует методов для получения и предварительной обработки для дальнейшего эффективного хранения. Основным моментом для сбора данных является обеспечение качества данных, устранения пробелов в них.

Поскольку получение данных происходит от каждого отдельного человека, то для выявления остановочных пунктов общественного транспорта необходимо выявить центры скоплений пунктов отправления и назначения. Эта задача является задачей кластеризации равномерно распределенных данных. Поскольку элементы данных являются географическими объектами, то при кластеризации необходимо учитывать естественные и искусственные препятствия — реки или железные дороги.

Следующим шагом является объединение центров кластеров для создания маршрутов сети общественного транспорта. Эта проблема лежит на пересечении проблемы поиска кратчайшего пути и задачи выбора маршрута транспортного средства со входным параметром пассажиропотока. В отличие от задач поиска кратчайшего пути с функцией затрат, на данном этапе рассматривается функция стоимости, включающая в себя время пешего хода, время, проведенное в пути, и другие изменения. Конечные точки маршрутов и остановки, через которые проложен маршрут, являются центрами кластеров, определенных на предыдущем шаге.

На последнем шаге происходит отбор самой оптимальной сети маршрутов по заданному набору критериев качества.

Формально, данную задачу можно разбить на две подзадачи: сбор данных и их кластеризация и построение оптимальной сети маршрутов по полученным центрам кластеров.

Для кластеризации геораспределенных данных было проанализировано множество алгоритмов, некоторые из которых подробно описаны во второй главе: алгоритмы среднего сдвига (Mean Shift), k-средних (k-means), иерархический метод BIRCH, алгоритмы DBSCAN и OPTICS. В качестве метрик расстояний между точками были рассмотрены евклидова метрика, решение обратной геодезической задачи и расчет расстояний по городским дорогам с при помощи сервиса построения маршрутов Open Source Routing Machine (OSRM) [44].

1.2 Состояние современных исследований

Задача кластеризации заключается в разбиении множества данных объектов на непересекающиеся подмножества — кластеры — таким образом, чтобы каждое подмножество состояло из схожих объектов, а объекты разных кластеров существенно отличались, при этом кластеры изначально неизвестны. Для определения схожести объектов необходимо задавать функцию расстояния на множестве объектов.

Решение задачи кластеризации является неоднозначным. Во-первых, не существует всеми принятого критерия качества кластеризации. Существует множество логически или эмпирически выведенных критериев качества, также существует множество алгоритмов, которые не имеют четко выраженного критерия качества, но осуществляющих достаточно разумную кластеризацию.

Во-вторых, число кластеров, как правило, является неизвестным заранее и устанавливается определенным критерием, отличающегося у различных конкретных алгоритмов. В-третьих, результат кластеризации существенно зависит от выбора метрики ρ , выбор которой также субъективен и определяется экспертом [4, 5].

Поскольку выбор метрики зачастую является абсолютно субъективным, то для решения задачи кластеризации создаются новые, всё более сложные, алгоритмы и улучшаются старые. Одним из широко применяемых алгоритмов для кластеризации является алгоритм k-средних, или k-means [6–12].

Алгоритм k-means производит разбиение входной выборки на k кластеров. Действие алгоритма таково, что он стремится минимизировать среднеквадратичное отклонение точек кластера от его центра. Основная мысль алгоритма заключается в том, что на каждом шаге пересчитываются центр масс каждого кластера, полученного на предыдущем шаге, а затем выборка переразбивается на кластеры вновь в соответствии с тем, какой из новых центров кластеров оказался ближе по выбранной метрике. Алгоритм завершается, когда на каком-либо шаге не происходит изменения кластеров.

Достоинствами метода являются его простота и быстрота использования (вычислительная сложность равна $O(nki)$, где n — число объектов выборки, k — число кластеров, i — число итераций алгоритма), а также его понятность и прозрачность. Недостатками алгоритма являются: (i) чувствительность к выбросам, (ii) необходимость задавать количество кластеров, (iii) необходимость сканировать всю выборку для определения положения каждого кластера.

Из-за медленной работы алгоритма на больших выборках данных, а также простоты идеи и неплохих результатов для большинства выборок, предлагается множество способов ускорить или улучшить алгоритм k-средних.

Так, в работе [6] для ускорения работы алгоритма k-means, предложено использование неравенства треугольника. Ускоренный алгоритм избегает излишних вычислений расстояния с помощью сохранения «верхних» и «нижних» границ для дистанций между точками и центрами, для расчета которых двумя различными методами применяется неравенство треугольника. Ускоренный алгоритм выдает те же результаты, что и обычный, но при этом количество вычислений дистанций сокращается в десятки раз.

Еще один вариант ускорения алгоритма путем сокращения лишних вы-

числений предлагаю авторы статьи [13]: после каждой итерации алгоритма необходимо разделять кластеры на те, которые изменили своё положение, так называемые «активные», и «статические» — оставшиеся на месте; и продолжать дальнейшие расчеты для активных кластеров. При этом для корректировки списков сохраняется минимальное значение расстояний от центров кластера до точек, при достижении которых кластер снова становится «активным».

В работе [14] алгоритм ускоряется при помощи хранения результатов в так называемых «мульти-размерных kd-деревьях». В узлах дерева хранятся центры кластеров, точки ищутся в «листьях» — особых гиперпрямоугольниках, что позволяет сократить количество расчетов.

В этой же статье рассматривается еще один алгоритм — так называемый алгоритм «Черного списка». Идея заключается в идентификации тех центров, которым точно не будут принадлежать точки из листьев, и исключить их из расчетов.

В статье [7] предлагается «алгоритм фильтрации». Он также основан на хранении многомерных данных в структуре kd-деревьев, а отличается тем, что выбор кластера, которому должен принадлежать лист, осуществляется построением гиперплоскости между претендентами.

Авторы работы [11] оценивают отношение расстояний между точкой и двумя центрами кластеров, и приходят к выводу, что большая часть «активных», то есть сдвигающих центр, точек находится в определенном диапазоне значений этого отношения. Используя этот факт на шаге присвоения точек, они создают ускоренную версию алгоритма k-means, однако результат в данном случае оказывается не точным, а приближенным.

В диссертации [15] был описан «быстрый жадный алгоритм k-средних». Предположение, используемое в алгоритме, такое же, как в обычном жадном (глобальном) алгоритме [8]; оно заключается в том, что глобальный оптимум может быть достигнут при запуске алгоритма с $(k - 1)$ центрами, и k -тым центром, достраиваемым автоматически на подходящую позицию. Автор объединил варианты ускорения, предложенные в [14] и [7], с наивным алгоритмом.

В работе [9] предложен усовершенствованный метод k-means, который путем последовательного запуска алгоритма для центров от 1 до k приближает решение к глобальному минимуму искажения. Это достигается за счет процедуры определения векторов-кандидатов на выбор позиции для вставки

нового центра, при этом общее время работы изменяется незначительно.

Кришна и Мёрти в своей работе [12] представляют «генетический k-means» — генетический алгоритм, в котором в качестве операции кроссовера выступает операция k-means. Сам генетический алгоритм был изменен для специфической работы — кластеризации.

Результат работы алгоритма k-средних очень сильно зависит от начального распределения кластеров. В работе [16] рассматривается метод k-means++, в котором начальное распределение центров кластеров задается особым образом, опираясь на вероятности, рассчитанные по кратчайшим расстояниям от точек до выбранных центров.

В работе [10] представлен алгоритм квантования цветов, основанный на методе k-средних и дополненный алгоритмом гравитационного поиска [17]. После стандартного расчета расстояний по метрике, рассчитываются «массы» кластеров и «силы», действующие на них. Таким образом, в работе достигается улучшение значений заданной целевой функции.

Работы [18] и [19] описывают построение словарного дерева с использованием двух различных вариаций алгоритма k-means: иерархического и приближенного соответственно. Иерархический алгоритм строит дерево результатов, на каждом следующем уровне разбивая имеющиеся кластеры на количество, называемое «коэффициентом разветвления», при этом каждый из полученных кластеров рассчитывается независимо от других. В приближенном алгоритме изначально каждое дерево развернуто до листьев, затем итерационно создается очередь приоритетных узлов до достижения конкретного числа открытых путей по дереву. При этом увеличение количества деревьев не приводит к значительному увеличению затрачиваемого времени.

В работе [20] алгоритм k-средних контролируется введением и комбинированием эвристических критериев: критерия оценки ошибки кластеризации и информационного критерия функциональной эффективности.

В работе [21] алгоритм k-means дополнен оценкой количества кластеров с помощью алгоритма X-means, запускаемого после каждой итерации основного алгоритма. X-means основан на двух идеях для определения оптимального количества кластеров: создание нового кластера рядом с одним из существующих и разделение некоторого количества кластеров на равноотстоящие друг от друга кластеры. Решение о принятии созданной структуры принимается на основе Байесовского информационного критерия.

Определение оптимального количества кластеров во входной выборке является одной из сложнейших проблем в кластерном анализе [16, 21–24].

Авторы работы [22] предлагают метод определения количества кластеров, основанный на искажениях: после очередной кластеризации рассчитываются соответствующие искажения, затем искажения трансформируются, рассчитываются значения «скачков» и среди них выбирается значение количества кластеров.

В работах [23] и [24] для определения количества кластеров используется метод «силуэтов». Каждый кластер представляется силуэтом, основанном на его плотности и разреженности. В [23] автор предлагает метод построения и применения силуэтов для оценки результатов кластерного анализа, а в [24] описывается применение различных методов (индекс Данна, индекс Калинского-Харабаза и др.) для определения количества кластеров в зашумленной выборке.

В статье [25] для определения кластерной структуры предварительно используется генетический алгоритм и метод силуэтов, которые в совокупности определяют оптимальные параметры для кластеризации методом k-средних.

Тханг, Пантиухин и Галушкин в работе [26] представляют гибридный алгоритм кластеризации на основе алгоритма k-means и DBSCAN – плотностного алгоритма для кластеризации зашумленных пространственных данных. При существенном сокращении времени кластеризации точность при некоторых параметрах достигает точности алгоритма DBSCAN.

Также существует множество других алгоритмов кластеризации, основанных на различных идеях о поиске внутренней структуры данных. Можно выделить несколько больших групп алгоритмов по способу кластеризации [32]: разделяющие методы, иерархические методы, плотностные методы и сеточные методы.

Для кластеризации при обработке изображений и данных высокой размерности зачастую применяют алгоритм среднего сдвига, или Mean Shift. В статье [27] автор использует этот алгоритм для определения границ регионов и фона на изображениях, а также описывает принцип работы алгоритма и критерии подбора его параметров.

Для кластеризации динамических наборов точек авторы работы [28] разработали модель, названную инкрементной кластеризацией. Она основана

на тщательном анализе требований поисковых систем, целью модели является сохранить кластеры малого диаметра при добавлении новых точек.

Для кластеризации категоричных данных в работе [29] применяется обобщение парадигмы метода k-means — метод k-modes. В нем используется особая метрика, вместо среднего значения набора данных находится их мода и используется частотный метод обновления мод.

В работе [30] описывается алгоритм BIRCH — иерархический алгоритм кластеризации, особенно эффективный при обработке очень больших выборок. Алгоритм является локальным — принятие решений относительно одного кластера производится без перебора всех данных; алгоритм используют на данных с четко выраженным кластерами; алгоритм является иерархическим, благодаря чему может находить скрытые субкластеры, а время работы линейно масштабируемо.

Проблема кластеризации геораспределенных данных с учетом препятствий была описана в работах [31, 32]. В качестве наглядного примера был приведен пример расстановки банкоматов в области, через которую протекает река.

Для решения этой задачи в [31] предлагалось использовать измененный алгоритм CLARANS (более подробно рассмотрен в работе [32]), в [32] перед основной кластеризацией алгоритмом CLARANS предлагается сделать предварительную кластеризацию алгоритмом BIRCH. Сам алгоритм CLARANS был немного изменен для применения к решению задачи кластеризации геораспределенных данных.

В работе [33] так же рекомендуется использовать алгоритм CLARANS, поскольку он эффективно обрабатывает большие пространственные базы данных. В качестве альтернативы рассматривается алгоритм BIRCH из-за более быстрой работы.

В работах [34] и [35] данные о препятствиях рассчитываются из диаграмм Воронова и Делоне с помощью алгоритмов AUTOCLUST и AUTOCLUST+ соответственно.

1.3 Требования к решаемой задаче

Таким образом, если рассматривать задачу создания остановочных пунктов методом кластеризации геораспределенных данных с учетом рельефа, то рассмотренные способы и подходы не решают сформулированные задачи и

имеют следующие недостатки: (а) не учитывают предпочтения жителей по перемещениям в городской среде; (б) не адаптированы для работы с реальными геораспределенными данными, получаемых из картографических сервисов.

Для решения задачи предлагается подход, состоящий из следующих шагов:

- 1) сбор и/или моделирование геораспределенных данных о перемещениях жителей в городе;
- 2) кластеризация полученных данных с целью выявления узлов (остановочных пунктов) для дальнейшего построения маршрутной сети.
 - 2.1) Задание характеристик производимой кластеризации.
 - 2.2) Формирование сети остановочных пунктов.

1.4 Заключение

В данной главе было рассмотрено общее состояние существующей проблемы в кластеризации геораспределенных данных. Были рассмотрены различные методы, предлагаемые для кластеризации, способы ускорения широко используемых алгоритмов, а также способы учета препятствий при кластеризации.

2 Методы кластеризации геораспределенных данных

Геораспределенные данные — это данные о событиях, которые в качестве основных атрибутов и атрибутов временной метки имеют две географические координаты [38].

При автоматизированном сборе информации о предпочтениях жителей по перемещению возникает потребность работы с большими объемами геораспределенных данных. Поскольку в маршрутной сети количество остановочных пунктов ограничено, то возникает проблема создания выборки данных, оптимальным образом представляющая исходный объем геораспределенных данных. Для этого предлагается использовать методы кластеризации данных.

Стоит заметить, что в работе рассматриваются предпочтения жителей города по перемещению, которые представляют собой пару геораспределенных объектов «отправление–назначение». Однако, при кластеризации дифференциация геораспределенных объектов не происходит — все объекты рассматриваются в единой выборке. Это сделано потому, что остановки общественного транспорта, то есть результат кластеризации, не имеют дифференциаций на остановки «отправления» и остановки «назначения».

Сформулируем постановку задачи кластеризации геораспределенных данных.

Пусть X — входная выборка точек, полученная из картографических сервисов, k — ожидаемое число узлов (остановочных пунктов) во входной выборке. Требуется построить C — сеть из k остановочных пунктов, оптимальным образом охватывающие все точки из входной выборки X .

Задача является задачей машинного обучения без учителя, но имеет некоторые специфические особенности. Во-первых, стандартные применяемые метрики для решения данной задачи не подходят: необходимо учитывать особенности местности, такие как реки, овраги и балки, автомобильные и железные дороги, здания и промышленные зоны. Существуют алгоритмы, способные на учет препятствий [32, 34, 35], но они не опираются на карту местности, а хранят информацию о препятствиях в некотором абстрактном виде. Во-вторых, поскольку на выходе необходимо получить сеть остановочных пунктов, то центры рассчитываемых кластеров должны находиться на участках дорожной сети, а не в произвольной географической точке.

В качестве базового алгоритма был выбран алгоритм k-средних ввиду его удобной расширяемости на поставленную задачу и вида входных данных,

которые можно считать равномерно распределенными по рассматриваемой области, то есть в них нет четко выраженных кластерных структур.

Предлагается два способа расчета расстояний между геораспределенными объектами. Первый способ заключается в использовании в качестве расстояний решение обратной геодезической задачи для рассматриваемых точек [36]. Второй способ заключается в использовании в качестве расстояний длину маршрута между двумя рассматриваемыми точками.

Для начала рассмотрим алгоритмы кластеризации, с помощью которых можно решать данную задачу, если внедрить в них описанные выше методы расчета расстояний и подобрать конкретные параметры управления под конкретную входную выборку.

2.1 Общее описание методов

Можно выделить несколько больших групп алгоритмов по способу кластеризации [32]: разделяющие методы, иерархические методы, плотностные методы и сеточные методы.

2.1.1 Разделяющие методы

Разделяющие алгоритмы распределяют объекты заданной d -мерной выборки X на заданное число k кластеров таким образом, что полное отклонение каждого объекта выборки от центра его кластера минимально. Расчет отклонения объекта может производиться по-разному в различных алгоритмах, он более известен как функция схожести объектов.

К разделяющим алгоритмам относят алгоритм максимизации ожидания (англ. Expectation Maximization), или EM-алгоритм, и различные его производные: методы k-means, k-medoids и другие. Алгоритм k-means использует расчет центров масс объектов в кластере, тогда как алгоритм k-medoids в качестве центра кластера использует самый центральный элемент кластера. EM-алгоритм является вероятностным (или нечетким) методом: он не присваивает объекты определенным кластерам, а говорит, что объект принадлежит всем кластерам, но с некоторой вероятностью [4, 32, 40, 41].

Все разделяющие алгоритмы можно представить одним общим алгоритмом 2.1.

Исходные параметры: Число кластеров k , выборка X

Результат: Набор кластеров C из k элементов, минимизирующий некоторую функцию-критерий

1. Каким-либо образом выбрать начальное расположение k элементов;
2. **до тех пор, пока C изменяются выполнять**

1. Рассчитать принадлежность объектов выборки текущему набору кластеров;
2. Обновить центры кластеров согласно рассчитанным принадлежностям объектов;

конец

Рисунок 2.1 – Общий алгоритм разделяющих методов

2.1.2 Иерархические методы

Иерархические методы производят иерархическое разбиение входной выборки, формируя дендрограмму — дерево, которое рекурсивно разделяет выборку на все меньшие группы объектов. Дендрограмма может формироваться двумя методами: «снизу–вверх» и «сверху–вниз».

Построение «снизу–вверх», называемое агломерационным, начинается с разбиения всех объектов на отдельные группы. Далее происходит объединение близлежащих по каким-либо метрикам объектов в большие группы. Алгоритм останавливается, когда все объекты выборки образуют одну группу.

Построение «сверху–вниз», называемое разделяющим, происходит с точностью наоборот: сначала все объекты образуют одну большую группу, которая по каким-либо метрикам отделяет далекие группы друг от друга. Алгоритм останавливается, когда все объекты выборки образуют отдельные кластеры.

При работе с иерархическими методами встает вопрос об определении «расстояния» между группами для объединения или разбиения [4]. Существует формула Ланса–Уильямса (2.1), подбором параметров которой можно добиваться различных результатов:

$$R(W, S) = \alpha_U R(U, S) + \alpha_V R(V, S) + \beta R(U, V) + \gamma |R(U, S) - R(V, S)|, \quad (2.1)$$

где U, V — объединяемые кластеры, $W = U \cup V$ — объединенный кластер, S — другие кластеры, $R(a, b)$ — расстояние между элементами, $\alpha_U, \alpha_V, \beta, \gamma$ — числовые параметры. Есть несколько частных случаев формулы Ланса–

Уильямса:

- 1) расстояние ближнего соседа (рис. 2.2а): $\beta = 0, \gamma = -1/2$,
 $\alpha_U = \alpha_V = 1/2; R_B(W, S) = \min_{w \in W, s \in S} \rho(w, s);$
- 2) расстояние дальнего соседа (рис. 2.2б): $\beta = 0, \gamma = 1/2$,
 $\alpha_U = \alpha_V = 1/2; R_D(W, S) = \max_{w \in W, s \in S} \rho(w, s);$
- 3) групповое среднее расстояние (рис. 2.2в): $\beta = \gamma = 0$,
 $\alpha_U = \frac{|U|}{|W|}, \alpha_V = \frac{|V|}{|W|}; R_\Gamma(W, S) = \frac{1}{|W||S|} \sum_{w \in W} \sum_{s \in S} \rho(w, s);$
- 4) расстояние между центрами (рис. 2.2г): $\beta = -\alpha_U \alpha_V, \gamma = 0$,
 $\alpha_U = \frac{|U|}{|W|}, \alpha_V = \frac{|V|}{|W|}; R_\Pi(W, S) = \rho^2 \left[\sum_{w \in W} \frac{w}{|W|}, \sum_{s \in S} \frac{s}{|S|} \right].$

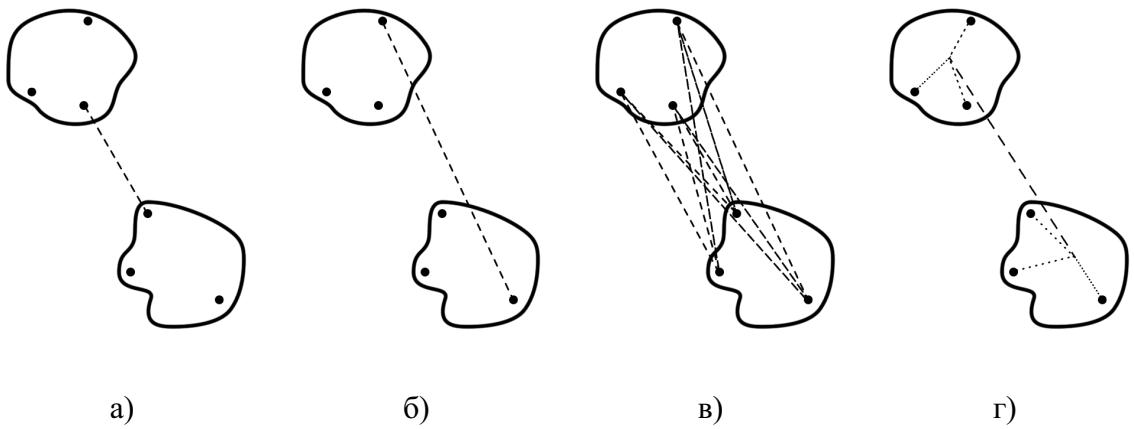


Рисунок 2.2 – Иллюстрации частных случаев расчета расстояния по формуле (2.1)

Не смотря на простую идею, у первых алгоритмов были существенные проблемы с критериями принятия решений о слиянии или разделении групп, одно неверное решение приводило к дальнейшим неправильным результатам и фактически губило правильные результаты, полученные ранее.

К иерархическим алгоритмам относят следующие алгоритмы: алгоритмы AGNES и DIANA, представляющие собой самые первые версии иерархических алгоритмов, в них соответственно реализованы агломерационная и разделяющие построения; алгоритмы CURE и CHAMELEON, разработанные для устранения недостатков первых алгоритмов, и современные алгоритмы, такие как BIRCH [4, 32, 41].

2.1.3 Плотностные методы

Большинство разделяющих методов построено на расчете расстояний между объектами. Такой подход позволяет искать кластеры только сферической формы, а результаты поиска кластеров другой формы оказываются неудовлетворительными. Поскольку типы кластерных структур в данных могут быть различными (рис. 2.3), то необходим иной подход для выделения кластеров в данных.

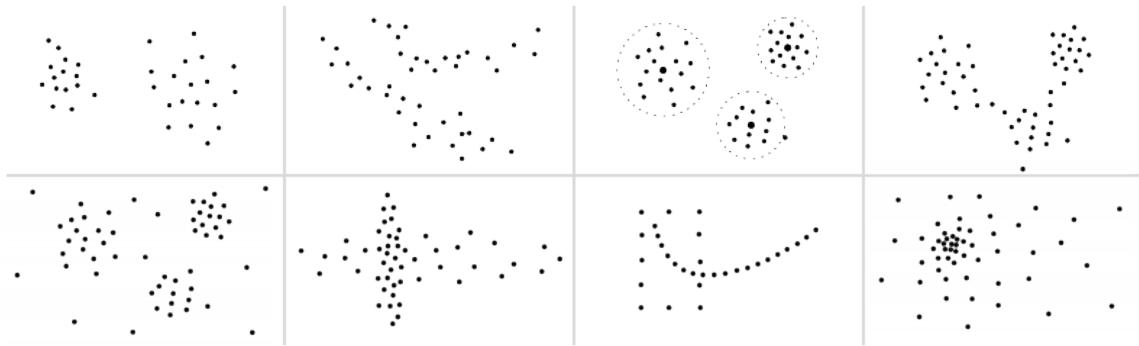


Рисунок 2.3 – Типы кластерных структур (рисунок взят из курса лекций Воронцова К. В. [4])

Этим подходом стал анализ плотности объектов в пространстве. В плотностных методах кластеры рассматриваются как сгустки плотности, между которыми находится разреженное пространство. Такой подход позволяет не только находить кластеры различной формы, но и отсеивать шум в данных.

К плотностным алгоритмам относят алгоритмы DBSCAN, OPTICS, DENCLUE и другие. Такие алгоритмы требуют тщательного подбора входных параметров, с помощью которых определяется плотность в пространстве вокруг объектов выборки: неправильно заданные параметры могут привести либо к получению одного кластера, содержащего всю выборку, либо к множеству мелких кластеров. Подробнее плотностный алгоритм DBSCAN будет рассмотрен в разделе 2.2.2.

Плотностные алгоритмы, подобные DBSCAN и OPTICS, существенно теряют в производительности при высокой размерности данных. Алгоритм DENCLUE разрабатывался для уменьшения потери производительности [32, 41].

2.1.4 Сеточные методы

Чтобы увеличить производительность при обработке высокоразмерных данных в сеточных алгоритмах используется сеточная структура данных. Она представляет собой пространство, поделенное на конечное число ячеек, над которыми производится операция кластеризации. Основное преимущество такого подхода — быстрое время обработки, которое обычно не зависит от числа объектов выборки, а зависит от числа ячеек в каждой плоскости сеточной структуры. К сеточным алгоритмам относят алгоритмы STING, WaveCluster, CLIQUE и другие.

Алгоритм STING делит пространство на квадратные ячейки. Обычно используется несколько уровней таких ячеек, имеющих иерархическую структуру: каждая ячейка верхнего уровня разделяется на несколько ячеек более нижнего уровня. В каждой ячейке хранятся общие данные, рассчитанные по области ячейки. Данные о ячейке состоят из количества элементов в ячейке, минимального, максимального и среднего значений, среднеквадратического отклонения, а также тип распределения данных в ячейке. Для проведения кластеризации пользователь должен задать уровень плотности, используя который алгоритм проведет агломерационное разбиение данных, выбирая подходящие по плотности ячейки.

Алгоритм CLIQUE объединяет плотностный и сеточный подходы. В отличие от других алгоритмов, он способен находить кластеры в подпространствах данных, что удобно при работе с многомерными данными, которые обычно являются разреженными и не формируют кластеров в полном пространстве. С точки зрения обнаружения кластеров используется плотностный подход — пользователь определяет количество объектов, расположенных поблизости, для восприятия их как кластеров. С точки зрения проведения кластеризации используется сеточный подход — пространство разбито на ячейки, и при поиске в d -мерном пространстве алгоритм обобщает информацию, собранную в $(d - 1)$ -мерном пространстве [32].

2.2 Проанализированные методы

2.2.1 Алгоритм Mean Shift

Алгоритм Mean Shift (средний сдвиг) — это процедура определения положения максимумов функции плотности. Часто используется в работах

по кластеризации, обработке изображений и визуальном трекинге [27, 37]. Пример работы можно посмотреть на рисунке 2.4.

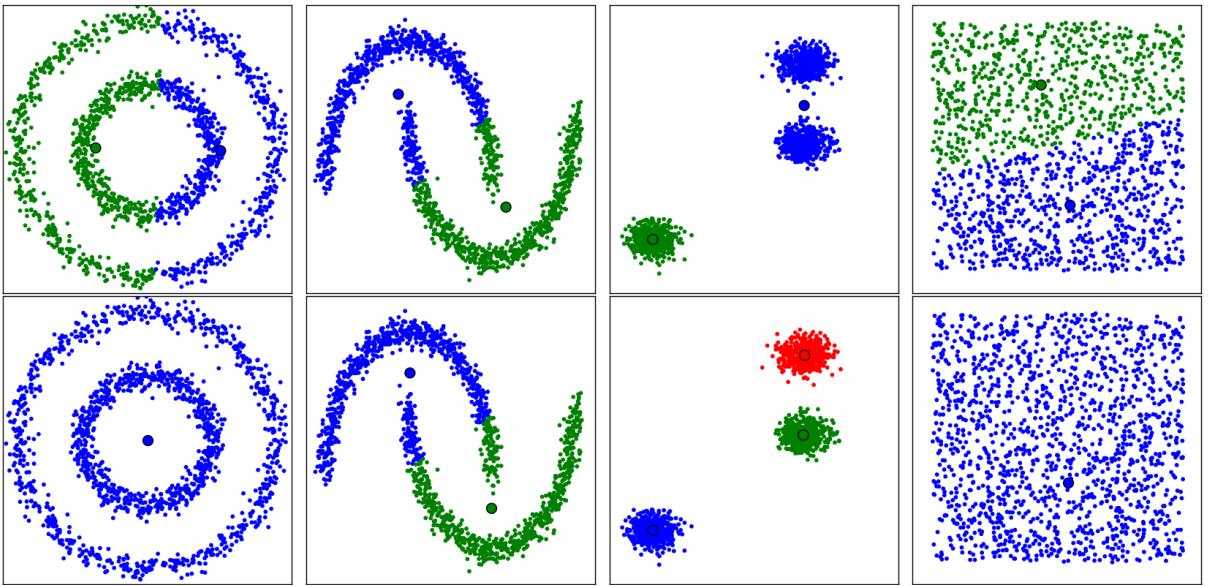


Рисунок 2.4 – Результаты обработки одинаковых наборов данных алгоритмами k-means (верхний ряд) и mean shift (нижний ряд). Рисунок получен с использованием библиотеки scikit-learn [46]

Пусть $S \subset X$ – конечная выборка d -размерного пространства X из n элементов, K – функция ядра, w – весовая функция. «Среднее выборки» с ядром K в точке $x \in X$ определяется функцией:

$$m(x) = \frac{\sum_{s \in S} K(s - x)w(s)s}{\sum_{s \in S} K(s - x)w(s)}. \quad (2.2)$$

Пусть $T \subset X$ – конечное множество центров кластеров. Изменение T итерационным присвоением $T \leftarrow m(T) = \{m(t), t \in T\}$ называется алгоритмом среднего сдвига, или Mean Shift. Алгоритм останавливается, когда достигает фиксированного множества $m(T) = T$ [37].

Алгоритм основан на парзеновской оценке плотности:

$$p_h(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - s_i}{h}\right), \quad (2.3)$$

где h – неотрицательный параметр, называемый шириной окна. Эмпирическая оценка плотности определяется как доля точек выборки $S = (s_i)_{i=1}^n$, лежащих

внутри отрезка $[x - h, x + h]$.

Параметр h в формуле (2.3) называют шириной окна (пропускной способностью, англ. bandwidth) алгоритма.

Ширина окна существенно влияет на качество определения сгустков плотности.

При $h \rightarrow 0$ сгустками плотности считаются все объекты выборки, функция (2.3) начинает претерпевать резкие скачки. При $h \rightarrow \infty$ функция плотности чрезмерно сглаживается и вырождается в константу [4].

При сильно неравномерном распределении объектов выборки в пространстве X одно и то же значение ширины окна h может привести к недостаточному сглаживанию плотности в одних областях пространства X и наоборот, к сильному сглаживанию плотности в других областях. Для решения этой проблемы используют переменную ширину окна, определяемую в каждой точке x пространства X . Тогда при расчетах используется матрица пропускной способности \mathbf{H} — симметричная положительная матрица размером $d \times d$:

$$f(x) = \frac{1}{n} \sum_{i=1}^n K_{\mathbf{H}}(x - s), \quad (2.4)$$

где $f(x)$ — функция оценки плотности, ядро $K_{\mathbf{H}}$ представляется в виде:

$$K_{\mathbf{H}}(z) = |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2} z).$$

Зачастую при расчетах матрицу \mathbf{H} считают диагональной: $\mathbf{H} = \text{diag}(h_1^2, \dots, h_d^2)$. При постоянной ширине окна матрицу можно представить в виде $\mathbf{H} = h^2 \mathbf{E}$, где $\mathbf{E} = \text{diag}(1, 1, \dots)$ — единичная матрица.

Функция ядра $K(z)$ является четной, нормированной и непрерывной, удовлетворяет следующим условиям:

$$\begin{aligned} \int_X K(z) dz &= 1; \quad \lim_{\|z\| \rightarrow \infty} \|z\|^d K(z) = 0; \\ \int_X z K(z) dz &= 0; \quad \int_X z z^\top K(z) dz = c_K \mathbf{E}, \end{aligned}$$

где c_K — константа.

В алгоритме используется профильный способ представления ядра, которым можно описать радиально-симметричные ядра, удовлетворяющие усло-

вию

$$K(z) = c_{k,d} k \left(\|z\|^2 \right). \quad (2.5)$$

Функция $k(z)$ называется профилем ядра, $c_{k,d} > 0$ — константа нормализации.

Подставляя (2.5) в (2.4) и считая ширину окна постоянной и равной h , получим функцию оценивания плотности в следующем виде:

$$f_{h,K}(x) = \frac{c_{k,d}}{nh^d} \sum_{i=1}^n k \left(\left\| \frac{x-s}{h} \right\|^2 \right). \quad (2.6)$$

Хотя функция ядра практически не влияет на качество кластеризации, она определяет степень гладкости функции (2.6), и ее вид может повлиять на эффективность вычислений.

Зачастую для алгоритма Mean Shift используют определенные профили ядерных функций:

- Плоское ядро: $k(z) = \begin{cases} 1 & \text{если } z \leq \lambda, \\ 0 & \text{если } z > \lambda; \end{cases}$
- нормальное (гауссовское) ядро: $k(z) = \exp \left(-\frac{\|z\|^2}{2\sigma^2} \right)$, в котором параметр среднеквадратичного отклонения σ играет роль ширины окна h ;
- ядро Епачечникова: $k(z) = \begin{cases} 1-z & \text{если } 0 \leq z \leq 1, \\ 0 & \text{если } z > 1. \end{cases}$

Формула (2.2) представляет собой упрощенную запись отношения $m(x) = c\nabla f(x)/f(x)$. Выражение ∇f является градиентом функции плотности:

$$\nabla f_{h,K}(x) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n (x-s) k' \left(\left\| \frac{x-s}{h} \right\|^2 \right).$$

Введя замену $g(z) = -k'(z)$, получим следующее:

$$\nabla f_{h,K}(x) = f_{h,G}(x) \frac{2c_{k,d}}{h^2 c_{g,d}} m_{k,G}(x),$$

где $G(z) = c_{g,d} g \left(\|z\|^2 \right)$, $m_{k,G}$ — средний сдвиг. Отсюда получаем:

$$m_{k,G}(x) = \frac{h^2 c_{g,d}}{2c_{k,d}} \frac{\nabla f_{h,K}(x)}{f_{h,G}}. \quad (2.7)$$

Таким образом, алгоритм на каждой итерации производит расчеты функций оценки плотности и градиентов функций оценки плотности в точках, считающихся центрами кластеров; после чего рассчитывает значение среднего сдвига и сдвигает центры кластеров на эту величину

Алгоритм Mean Shift [39, с. 235-236] можно представить алгоритмом 2.5.

Исходные параметры: S, h .

Результат: $C = \{C_j\}$ — список центров кластеров.

Для каждого центра C_j выполнить

1. Для каждой точки данных s выполнить

Рассчитать градиент плотности

$$\nabla f_{h,K}(C_j) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n (s - C_j) g\left(\left\|\frac{C_j - s}{h}\right\|^2\right);$$

конец

2. Сдвинуть позицию согласно (2.7): $C_j = C_j + m_{k,G}(C_j)$;

3. Если центры кластеров C изменились тогда перейти на шаг 1;
иначе закончить выполнение;

конец

Рисунок 2.5 — Алгоритм Mean Shift

2.2.2 Алгоритм DBSCAN

Алгоритм DBSCAN (англ. Density-Based Spatial Clustering of Application with Noise, плотностный алгоритм кластеризации пространственных данных с присутствием шума) был предложен как решение проблемы разбиения данных на кластеры произвольной формы.

Поскольку большинство алгоритмов стараются минимизировать расстояние от элементов до центра кластера, то они создают кластеры, по форме приближенные к сферическим. Авторы DBSCAN экспериментально показали, что разработанный алгоритм способен распознать кластеры различной формы. Пример работы алгоритма можно посмотреть на рисунке 2.6.

Алгоритм управляет двумя параметрами: ε — радиус области, на которой будет проводиться поиск соседних элементов, и μ — минимальное количество соседних элементов в ε -окрестности, при котором элемент считается

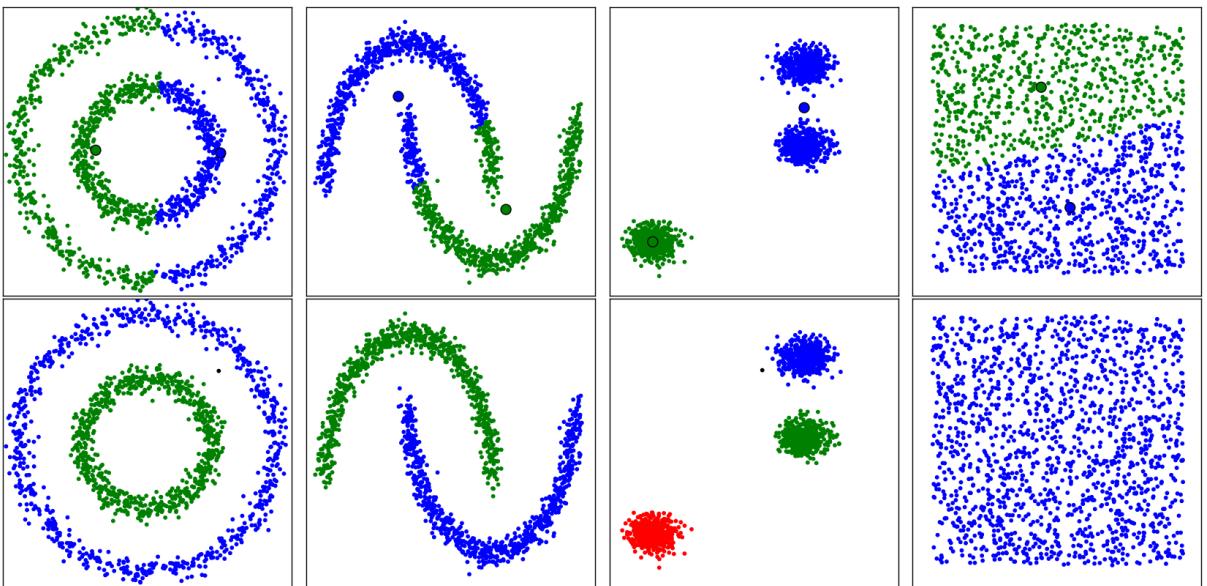


Рисунок 2.6 – Результаты обработки одинаковых наборов данных алгоритмами k-means (верхний ряд) и DBSCAN (нижний ряд). Рисунок получен с использованием библиотеки scikit-learn [46]

ядровым элементом.

Основной концепт, положенный в основу работы алгоритма, заключается в том, что внутри каждого кластера плотность объектов обычно заметно выше, чем плотность объектов снаружи кластера, и плотность объектов в областях с шумом гораздо ниже плотности любого из кластеров [40].

Реализация этого концепта была выражена следующими правилами [32]:

- объект может принадлежать кластеру только в том случае, если он лежит в ε -окрестности одного из ядерных элементов кластера;
- ядерный элемент o , лежащий в ε -окрестности другого ядерного элемента p , должен принадлежать тому же кластеру, что и p ;
- неядерный элемент q , лежащий в ε -окрестности нескольких ядерных элементов p_1, \dots, p_i , должен принадлежать к тому же кластеру, к которому принадлежит хотя бы один из ядерных элементов;
- неядерный элемент r , не лежащий в ε -окрестности ядерных элементов, считается шумовым элементом — элементом, не принадлежащему ни одному кластеру.

Чтобы обнаружить кластеры в данных, алгоритм DBSCAN просматривает ε -окрестности всех элементов входной выборки X . Если в ε -окрестности элемента p находится больше, чем μ элементов, то создается новый кластер с ядерным элементом p . Все элементы в ε -окрестности ядерного элемента p

Исходные параметры: X, ε, μ .

Результат: $C = \{C_j\}$ — список кластеров.

1. Установить всем элементам X флаг = «не посещен»; $j = 0$;
множество шума: $N = \emptyset$;
2. Для каждого $x \in X$ такого, что флаг == «не посещен» выполнить
 1. Флаг(x) = «посещен»
 2. $n_x = \text{Eps}(x) = \{q \in X | \rho(x, q) \leq \varepsilon\}$;
 3. Если $|n_x| < \mu$ тогда $N = N + \{x\}$
иначе
 - номер следующего кластера $j = j + 1$;
 - ExpandCluster($x, n_x, C_j, \varepsilon, \mu$);

конец

конец

ExpandCluster

Исходные параметры: $x, n_x, C_j, \varepsilon, \mu$.

Результат: кластер C_j .

1. $C_j = C_j + \{x\}$;
2. Для каждого $q \in n_x$ выполнить
 1. Если Флаг(q) == «не посещен» тогда
 1. Флаг(q) == «посещен»;
 2. $n_q = \text{Eps}(q)$;
 3. Если $|n_q| \geq \mu$ тогда $n_x = n_x + \{n_q\}$

конец
2. Если элемент q не принадлежит ни одному из кластеров тогда
 $C_j = C_j + \{q\}$

конец

Рисунок 2.7 — Алгоритм DBSCAN

добавляются в созданный кластер. Новые ядерные элементы в ε -окрестности элемента p и элементы из их ε -окрестности также добавляются в созданный кластер; таким образом кластер разрастается. Как только в ε -окрестностях добавленных ядерных объектов больше нет элементов, то из выборки X берется следующий нерассмотренный ядерный элемент и создается новый кластер. Стоит отметить, что во время работы алгоритма ядерные элементы одного кластера могут быть обнаружены при разрастании другого, что приведет к слиянию кластеров. Алгоритм прекращает работу, когда нет новых точек, которые могут быть добавлены к какому-либо из кластеров.

Алгоритм DBSCAN [40, с. 199] можно представить алгоритмом 2.7. Используются следующие обозначения: Eps — множество соседей элемента,

находящихся от точки p на расстоянии не более ε , N — множество элементов шума.

Существует улучшенная версия алгоритма DBSCAN, названная OPTICS (англ. Ordering Points To Identify Clustering Structure, сортировка точек для определения кластерной структуры) [32]. Он был создан для того, чтобы не запускать алгоритм DBSCAN множество раз с различными параметрами для получения оптимальной с точки зрения пользователя кластерной структуры. OPTICS также требует определения входных параметров ε и μ , но, в отличие от DBSCAN, при работе сортирует точки таким образом, чтобы легко визуализировать и рассчитать кластерную структуру для данного ε и некоторых других значений $\varepsilon' < \varepsilon$.

2.2.3 Алгоритм BIRCH

Алгоритм BIRCH (англ. Balanced Iterative Reducing and Clustering using Hierarchies, сбалансированное итерационное сокращение и кластеризация с использованием иерархий) создан для работы с большими наборами данных.

Пример работы алгоритма можно посмотреть на рисунке 2.8.

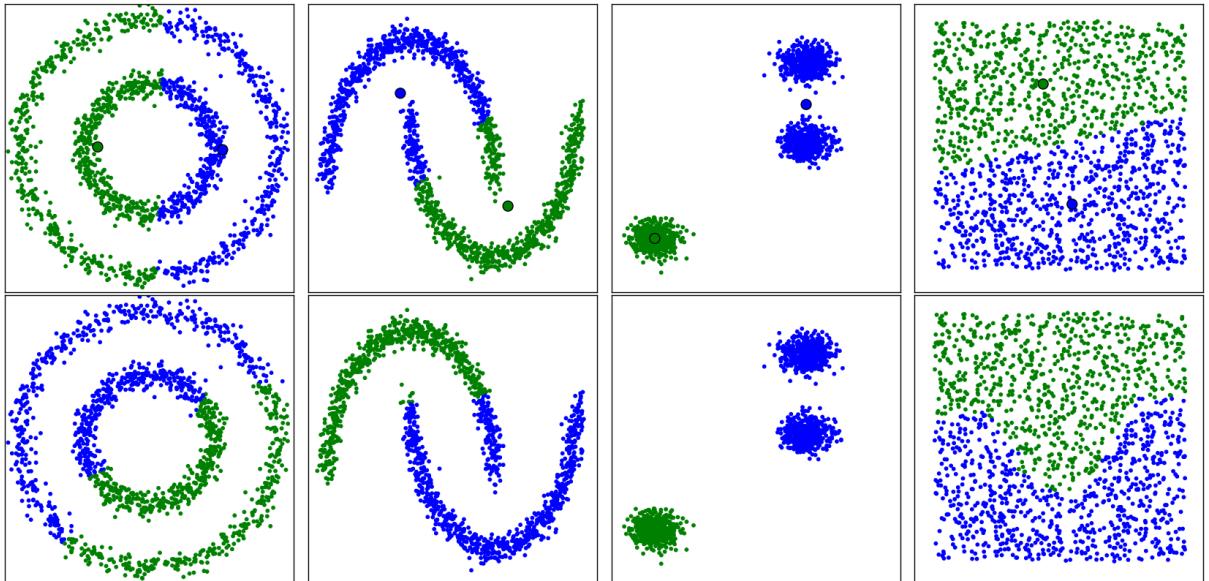


Рисунок 2.8 — Результаты обработки одинаковых наборов данных алгоритмами k-means (верхний ряд) и BIRCH (нижний ряд). Рисунок получен с использованием библиотеки scikit-learn [46]

Основная идея работы алгоритма BIRCH — сократить входную выборку до некоторого количества субкластеров, а затем провести кластеризацию над этими субкластерами [32]. Из-за усечения множества исходных данных коли-

чество субкластеров гораздо меньше, чем число элементов входных данных, следовательно, это позволяет производить кластеризацию с меньшим количеством используемой памяти.

В алгоритме каждый небольшой субкластер представляется кластерным элементом (англ. clustering feature), который является триплетом, обобщающим информацию о группе объектов выборки, которую заменяет субкластер. Если в заданном d -размерном пространстве X субкластер заменяет N объектов выборки, то кластерный элемент определяется как

$$CF = \left(N, \vec{LS}, SS \right),$$

где N — число заменяемых объектов выборки, $\vec{LS} = \sum_{i=1}^N \vec{X}_i$ — линейная сумма заменяемых объектов, $SS = \sum_{i=1}^N \vec{X}_i^2$ — сумма квадратов заменяемых объектов.

Кластерные элементы хранятся в дереве кластерных элементов, которое используется для иерархической кластеризации. Внутренние узлы дерева хранят суммы кластерных элементов их потомков и, следовательно, обобщенную информацию о потомках. Дерево кластерных элементов имеет два параметра: коэффициент ветвления B и пороговое значение T . Коэффициент ветвления определяет максимальное количество потомков у внутренних узлов дерева. Пороговое значение определяет максимальный диаметр субкластеров, хранимых в листовых узлах дерева. Эти два параметра влияют на результирующий размер дерева.

Поскольку алгоритм рассчитан на работу с очень большими данными, то дерево строится сразу по мере считывания входной базы данных. Объекты вставляются в ближайший листовой элемент (субкластер). Если диаметр субкластера после добавления нового объекта становится больше порогового значения, то листовой узел разделяется и становится внутренним узлом. После добавления нового объекта информация о нем передается в корень дерева.

Если результирующее дерево выходит за границы доступной памяти, то изменяется пороговое значение T и дерево перестраивается, начиная с корня. Таким образом, процесс перестройки дерева происходит без повторного считывания всей исходной выборки.

После постройки результирующего дерева для кластеризации может

использоваться любой алгоритм кластеризации, который не превысит границ доступной памяти.

Не смотря на достоинства алгоритма — линейную масштабируемость, хорошее качество кластеризации преобразованных данных, — узел в дереве кластерных элементов может содержать лишь ограниченное количество элементов из-за ограничения по размеру, что влечет за собой тот факт, что кластерное образование, наблюдаемое пользователем, может быть не отражено в дереве. Более того, если кластеры имеют несферическую форму, то алгоритм BIRCH не может корректно отразить этот факт, поскольку контроль за границами кластеров ведется через понятия радиусов и диаметров [30, 32].

Общее описание алгоритма [41, с. 2] приведено на рисунке 2.9.

Фаза 1. Загрузка данных в память.

Построение начального дерева по данным (сканирование выборки).

Фаза 2 (необязательная). Сжатие данных до приемлемых размеров.

Перестройка и уменьшение дерева кластерных элементов с увеличением порогового значения T .

Фаза 3. Глобальная кластеризация.

Применение выбранного алгоритма кластеризации на листовых элементах дерева.

Фаза 4 (необязательная). Улучшение кластеров.

Перераспределение данных между близкими кластерами, используя центры тяжести кластеров, полученные в фазе 3, как основы.

Рисунок 2.9 — Общая схема алгоритма BIRCH

2.3 Алгоритм k-means

Так как алгоритм BIRCH чаще всего используют именно для работы с очень большими данными, о которых можно иметь представление в виде кластерных элементов [30, 32, 41], то для кластеризации геораспределенных данных этот алгоритм не применялся.

Поскольку в работе рассматриваются данные, приближенные (но не являющиеся таковыми) к равномерно распределенным, то результатом работы алгоритма DBSCAN будет являться небольшое количество кластеров, включаяющих в себя большое количество точек.

Точно такое же поведение будет наблюдаться у алгоритма Mean Shift, но он более удобно настраивается, поскольку управляет одним параметром —

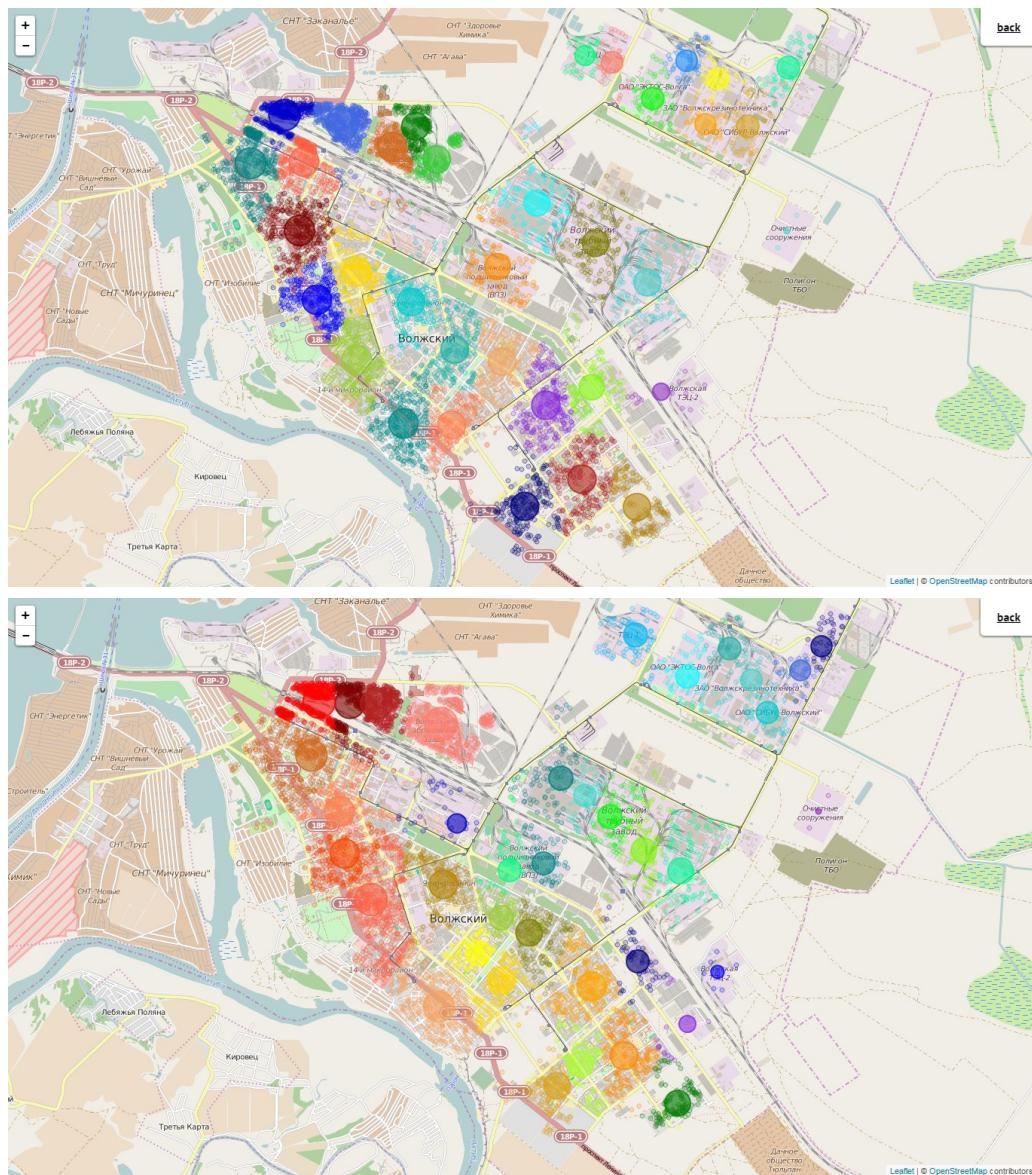


Рисунок 2.10 – Результаты работы алгоритмов k-means (верхний) и Mean Shift (нижний) на сгенерированной входной выборке. Широкими кругами обозначены центры кластеров, радиус круга зависит от количества входящих в него точек

ширины окна h . При уменьшении этого параметра алгоритм проводит более детальную кластеризацию, однако результаты его работы слабо отличаются от результатов работы алгоритма k-means (рис. 2.10).

2.3.1 Общее описание

Алгоритм k-means (k-средних) строит k кластеров, расположенных таким образом, чтобы минимизировать среднеквадратичное отклонение объектов выборки от центров кластеров. Выбор числа k может базироваться на результатах теоретических соображений, предшествующих исследований, ви-

зуального наблюдения или иных источниках информации [41].

Идеальным кластером алгоритм k-means считает сферу с центром кластера в центре сферы [40].

При известном числе кластеров алгоритм k-средних начинает работу с некоторого начального расположения кластеров, к которым присваиваются объекты выборки.

Начальное расположение кластеров очень сильно влияет на работу алгоритма. Из-за неправильного выбора начальных значений алгоритм может попасть в некоторый локальный минимум целевой функции (рис. 2.11).

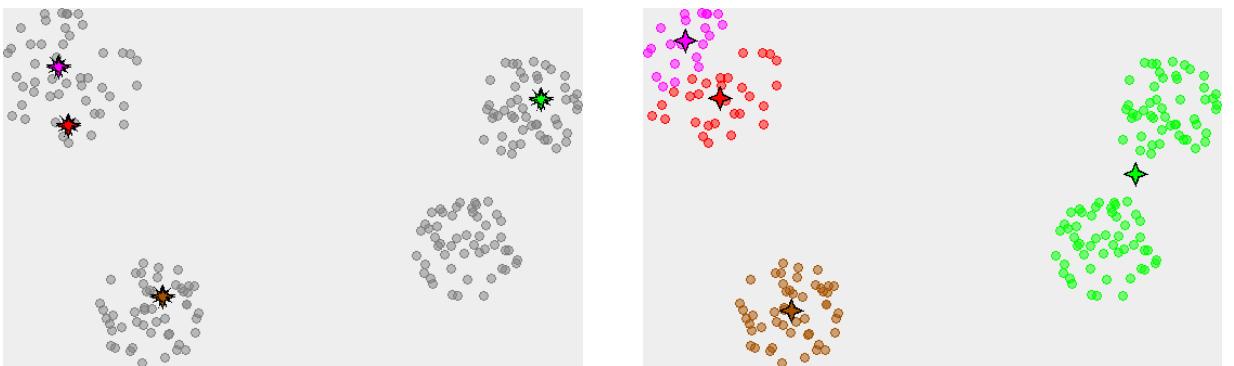


Рисунок 2.11 — Пример сходимости алгоритма к локальному минимуму. Кружками обозначены объекты выборки, звездами — центры кластеров. Рисунок получен с помощью апплета Миркеса [47]

Целевой функцией алгоритма является среднеквадратичное расстояние (при использовании евклидовой метрики) между объектами выборки и центрами их кластеров:

$$f(X, C) = \sum_{j=1}^k \sum_{i=1}^n \|x_i - \mu_j\|^2,$$

где μ_j — центр кластера C_j , вычисляющийся по формуле:

$$\mu_j = \frac{1}{|C_j|} \sum_{i=1}^n x_i. \quad (2.8)$$

Начальное расположение кластеров может задаваться как вручную, так и автоматически. В изначальном варианте алгоритма начальное расположение кластеров было случайным [42]. На практике в качестве начальных центров кластеров зачастую либо выбирают первые k объектов выборки, либо выби-

рают k самых удаленных друг от друга объектов выборки.

Алгоритм k -средних относительно масштабируем и эффективен при обработке не слишком больших баз данных поскольку его вычислительная сложность равна $O(nki)$, где n — полное число объектов выборки, k — число кластеров, i — число итераций алгоритма; обычно $k \ll n$ и $i \ll n$ [32].

Недостатками алгоритма являются:

- чувствительность к выбросам — точки, далеко находящиеся от всех кластеров, сильно искажают среднее;
- необходимость задавать количество кластеров — при работе с алгоритмом выбор числа кластеров является самым сложным вопросом. Если нет предположений о кластерной структуре данных, то рекомендуют постепенно наращивать число k , сравнивая полученные результаты [41];
- необходимость сканировать всю выборку для определения положения каждого кластера — при работе с большими базами данных это сильно замедляет работу.

Достоинствами метода являются его простота и быстрота использования, а также понятность и прозрачность алгоритма.

2.3.2 Объяснение работы

Работу алгоритма можно разбить на три этапа: начальный этап, этап присвоения и этап сдвига.

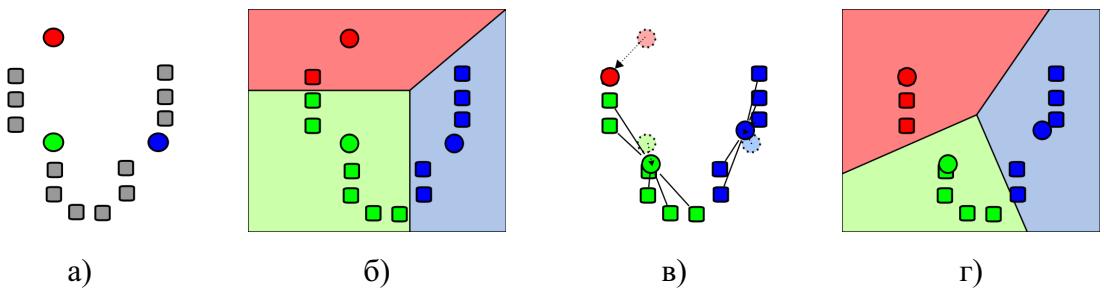


Рисунок 2.12 — Демонстрация работы алгоритма. Иллюстрация взята из статьи о k -means [48]

На начальном этапе происходит выбор числа k , расстановка начальных центров кластеров C_0 (рис. 2.12а).

На этапе присвоения происходит расчет расстояния от каждого объекта выборки X до каждого объекта кластера C_j . После чего считается, что объект принадлежит тому кластеру, до которого у него минимальное расстояние

(рис. 2.12б).

На этапе сдвига происходит расчет нового положения центров кластеров по формуле (2.8) (рис. 2.12в).

Этапы присвоения и сдвига повторяются, пока кластерные центры не стабилизируются, то есть все объекты, принадлежащие кластеру на прошлой итерации, принадлежат и на этой, либо пока не достигается заданное максимальное число итераций I . На выходе алгоритм выдает множество центров кластеров C и метки принадлежности объектов выборки кластерам L (рис. 2.12г).

Алгоритм k-средних можно представить алгоритмом 2.13.

Исходные параметры: X, k, C_0, I .

Результат: C, L .

1. $C = C_0, L_0 = \emptyset, i = 0;$
 2. **до тех пор, пока** $L \neq L_0$ и $i < I$ **выполнять**
 1. $L_0 = L, A = \emptyset, \mu = \emptyset;$
 2. **Для каждого** $x \in X$ **выполнить**
 1. $r = \emptyset;$
 2. **Для каждого** $c \in C$ **выполнить**
 1. Рассчитать $r_{xc} = \rho(x, c);$
 2. $r = r + \{r_{xc}\};$**конец**
 3. $L[x] = C[\min(r)];$
 4. $A[C[\min(r)]] = A[C[\min(r)]] + \{x\};$**конец**
 3. **Для каждого** $c \in C$ **выполнить**
 1. $\mu[c] = \sum(A[c]) / |A[c]|;$
 2. $c = \mu[c];$**конец**
 4. $i = i + 1;$
- конец**

Рисунок 2.13 – Алгоритм k-means

2.4 Расчет расстояний между точками

Особая метрика при работе алгоритма необходима по той причине, что два близких по стандартным метрикам геораспределенных объекта в реальности могут преграждаться препятствиями. Примеры таких объектов приведены

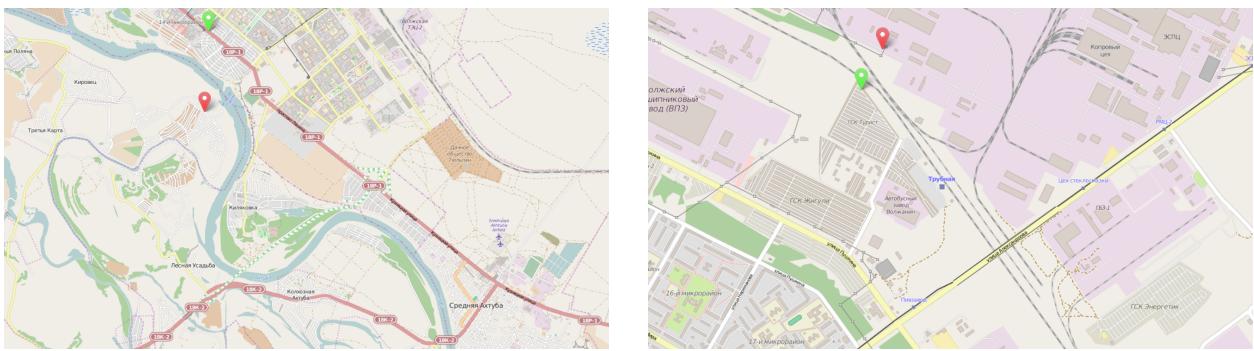


Рисунок 2.14 – Близкие по евклидовой метрике пары объектов

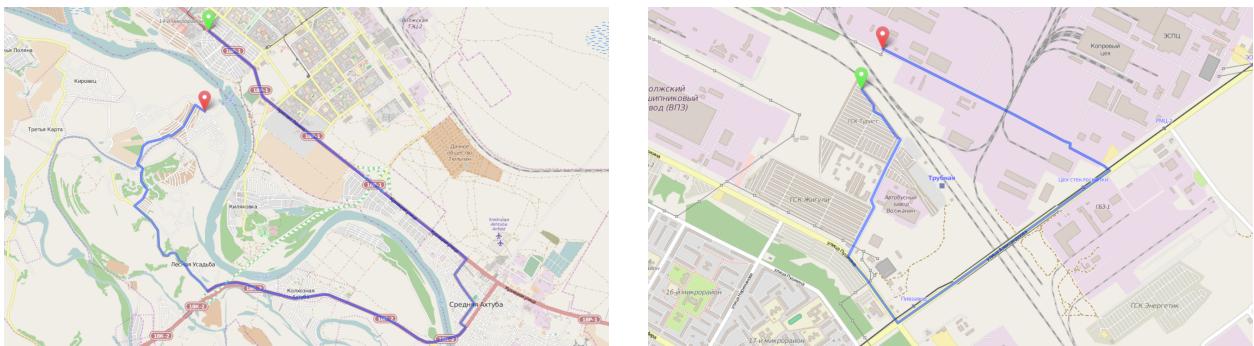


Рисунок 2.15 – Расстояние между объектами, которое должно учитываться при расчете

на рисунке 2.14. На рисунке 2.15 приведено реальное расстояние между объектами, которое должно учитываться при транспортных расчетах.

Для расчета расстояния между точками в работе используются две метрики, названные «Surface» и «Route».

Метрика Surface рассчитывает расстояние путем решения обратной геодезической задачи [36, с. 48-50]. Метрика Route рассчитывает расстояние между объектами по графу городских дорог.

Для решения обратной геодезической задачи в метрике Surface используется библиотека GeographicLib [43]. Метрика Route использует сервис построения маршрутов Open Source Routing Machine (OSRM) [44] для расчета расстояния между точками по городским дорогам.

Сервис OSRM предоставляет открытый код своего движка маршрутизации. Для работы с собранным движком на пользовательском компьютере используются HTTP-запросы определенной структуры, так называемое HTTP-API сервиса (поддерживается две версии API: v4 и v5). Результат обработки запроса выводится в json-формате, содержащем всю информацию по запросу.

В метрике используются два типа запросов: route (viaroute в версии HTTP-API v4) для расчета расстояний и nearest (locate в версии HTTP-API

v4) для сдвига кластеров к дорожной сети. Ответ на запрос route содержит список точек, через которые проложен маршрут, и список маршрутов; каждый маршрут содержит свою длину. Ответ на запрос nearest содержит список точек на графе дорог с указанием расстояний до них.

Для работы движок маршрутизации должен распаковать карту сервиса Open Street Map [45] для выделения на ней графа дорог. Главным параметром при работе с движком маршрутизации является используемый профиль (англ. profile). При сборке OSRM из исходных кодов предоставляется несколько профилей, среди которых профили автомобиля, велосипеда и пешехода. Используемый профиль влияет на способ построения маршрута и выбор городских дорог для его прокладки. При работе с метрикой Route использовался профиль автомобиля. Пример работы сервиса Open Source Routing Machine представлен на рисунке 2.16.

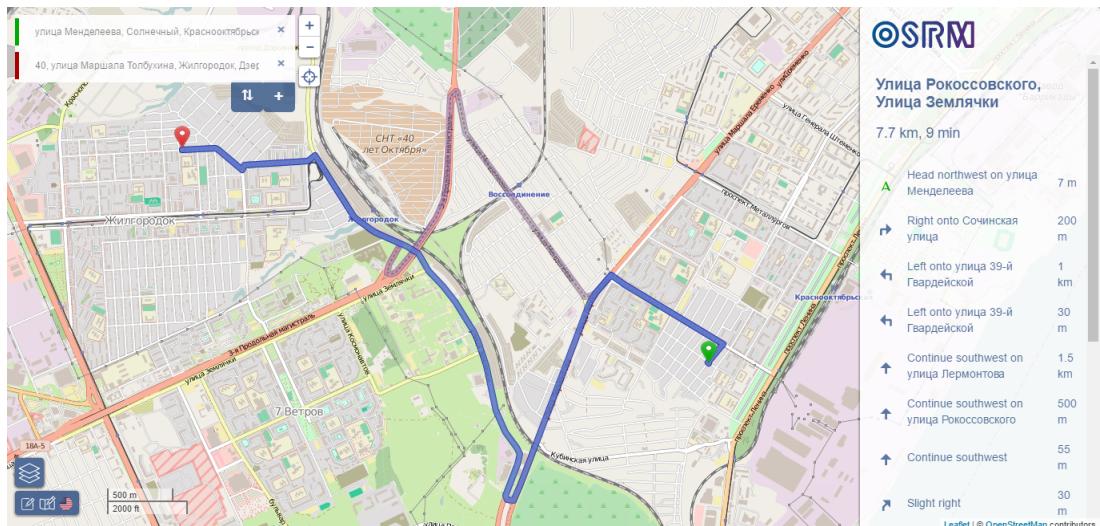


Рисунок 2.16 – Пример работы сервиса Open Source Routing Machine [44]

Работу метрики можно представить алгоритмом 2.17.

Шаги 0 и 4 этого алгоритма выполняются при старте и окончании кластеризации, то есть один раз за работу алгоритма 2.13. Использование шагов 1–3 метрики происходит на шаге 2.2.2.1 алгоритма 2.13. На шаге 2.1 используется запрос route (viaroute в HTTP-API v4) для нахождения расстояния между объектами. Если движок OSRM возвращает ответ с ошибкой, то объекты «локализуются» — к ним ищутся ближайшие участки графа дорог с помощью запросов nearest (locate в HTTP-API v4) (шаг 2.2.1), после чего расстояние ищется уже между ними (шаг 2.2.2). Округление координат на шаге 1

0. Запуск движка OSRM-routed;

Исходные параметры: a, b — объекты, между которыми ищется расстояние.

Результат: r .

1. Округление координат точек до 5 знаков после запятой;

2. **Если** Если координаты точек идентичны **тогда** расстояние $r = 0$;
иначе

1. Находим расстояние функцией $\text{route}(a, b)$;

2. **Если** Если расстояние не найдено **тогда**

1. Найти ближайшие к точкам a и b узлы графа дорог;

2. Рассчет расстояния между ними;

конец

3. Вывод расстояния;

4. Остановка движка OSRM-routed;

Рисунок 2.17 — Алгоритм метрики route

производится для отсеивания очень близко находящихся объектов, положение которых можно принимать в качестве одинакового. Как правило, расстояние между такими объектами не превышает 10 метров.

2.5 Заключение

В данной главе были рассмотрены алгоритмы кластеризации, которые можно использовать при внедрении соответствующих метрик и точной настройке параметров для кластеризации геораспределенных данных: алгоритмы Mean Shift, DBSCAN и BIRCH. Также были описаны основной алгоритм кластеризации, используемый в работе, — алгоритм k-means; и метрики, используемые для расчета расстояния между объектами входной выборки, — Route и Surface.

3 Испытание и обоснование эффективности предлагаемых подходов

3.1 Проектирование ПО

Разработанное программное обеспечение должно соответствовать техническому заданию представленное в Приложении Б и должна обеспечивать возможность выполнения следующих ниже функций:

- 1) Предоставлять возможность сохранять и загружать данные используемые для работы программы:
 - загрузка данных пользователей о перемещении;
 - преобразование загруженных данных во внутренний формат программы;
 - сохранение расчётных данных для последующей обработки;
- 2) Предоставлять функцию кластеризации по заданным параметрам, включающая следующие пункты:
 - использование заданной метрики;
 - ограничение на количество итераций алгоритма;
 - способ расстановки начальных кластеров.
- 3) Предоставлять возможность визуализировать выходные данные.

Предложенные алгоритмы из разделов 2.3 и 2.4 были реализованы с использованием языка программирования Python и сервиса построения маршрутов Open Source Routing Machine для расчета расстояния между узлами графа по городским дорогам. Программный код опубликован на хостинге Github (подробнее в Приложении А).

3.2 Методика проведения эксперимента

Для оценки эффективности работы алгоритма и изучения специфики метрики Route были проведены эксперименты, в ходе которых менялись выборки данных, количество получаемых кластеров и их начальное местоположение, а так же реализация метода.

Для генерации данных о предпочтениях жителей города по перемещению был создан инструмент Scatter [49], позволяющий удобным образом генерировать большие объемы геоданных.

Для оценки работы алгоритма на данных, приближенных к реальным, были сгенерированы данные о предпочтениях по перемещению жителей среднего по размерам города с примерным числом жителей около 350 000. В

качестве результата было получено 6000 пар точек отправления–назначения (рис. 3.1), или 12000 точек в общей сложности (выборка Main). Эту выборку было решено разбить на 125 кластеров, начальное положение которых было случайным. В последствие это случайное расположение кластеров было взято за основу для тестирования различных версий алгоритма.

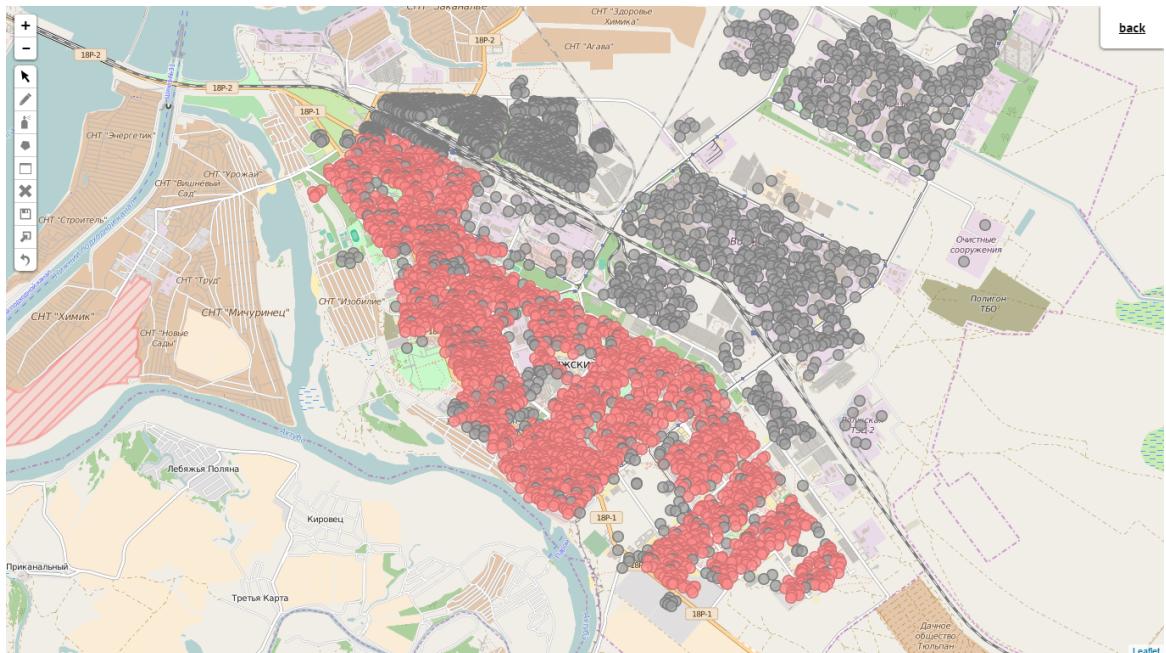


Рисунок 3.1 — Сгенерированная выборка из 12000 точек. Красным отмечены точки отправления, серым — назначения

Так же были сгенерированы другие выборки для проверки специфичных случаев: обхода препятствий в виде железной дороги (выборка Railway, рис. 3.2а) и реки (выборка River, рис. 3.2б). Каждая из них представляет собой набор из шести точек, которые кластеризуются в два кластера. Точки и

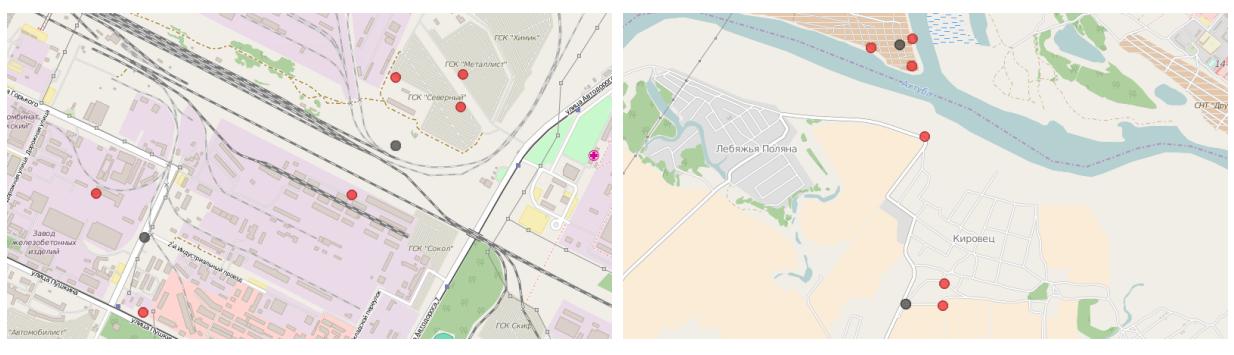


Рисунок 3.2 — Выборки для проверки обхода препятствий. На рисунке а) между объектами выборки находится железная дорога, на рисунке б) — река. Красными кругами отмечены элементы выборки, черными — начальные центры кластеров

начальные центры расположены таким образом, что если алгоритм не учитывает препятствия между объектами выборки, то в одном кластере окажутся точки, разделенные препятствием.

Критериями для оценки эффективности разработанных алгоритмов и метрик являются:

- время, затраченное на расчет расстояния между объектами выборки;
- учет препятствий;
- визуальная оценка результатов кластеризации.

4 Описание результатов испытания

4.1 Проведение эксперимента и описание результатов

Результаты работы алгоритма k-means доступны по следующей ссылке:

<https://vstu-cad-stuff.github.io/clustering/kmeans/>.

Результаты работы алгоритма k-means со сдвигом центров кластеров к дорожной сети доступны по следующей ссылке:

<https://vstu-cad-stuff.github.io/clustering/terminals/>.

Для тестирования работы алгоритма было использовано 4 выборки:

- выборка Main: $|X| = 12000$ точек, $k = 125$ (рис. 3.1); используется для общего тестирования работы алгоритмов;
- выборка Railway: $|X| = 6$ точек, $k = 2$ (рис. 3.2а); используется для проверки обхода препятствий;
- выборка River: $|X| = 6$ точек, $k = 2$ (рис. 3.2б); используется для проверки обхода препятствий;
- выборка Common: $|X| = 180$ точек, $k = 10$ (рис. 4.1); используется для тестов алгоритмов на скорость.

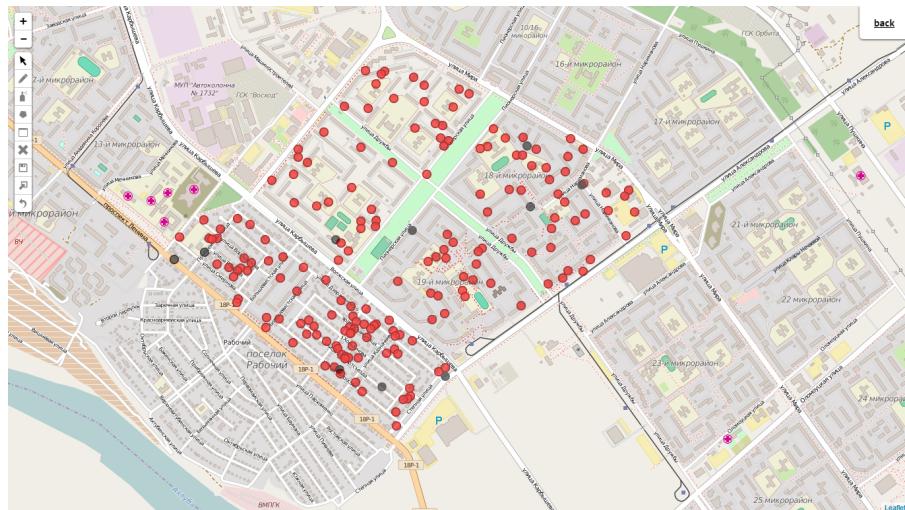


Рисунок 4.1 — Выборка Common. Красными кругами отмечены элементы выборки, черными — начальные центры кластеров

На рисунках, показывающих результаты кластеризации линиями очерчены кластерные области — выпуклые оболочки множества точек, принадлежащих кластеру.

4.1.1 Последовательная реализация

Данная реализация является наиболее простой из всех с точки зрения расчета расстояния. В данном случае применяется метрика Surface. Эта версия может быть интересна в качестве ориентира в teste на производительность. Данный вариант не позволяет учитывать препятствия, результаты кластеризации выборок Railway и River представлены на рис. 4.2.

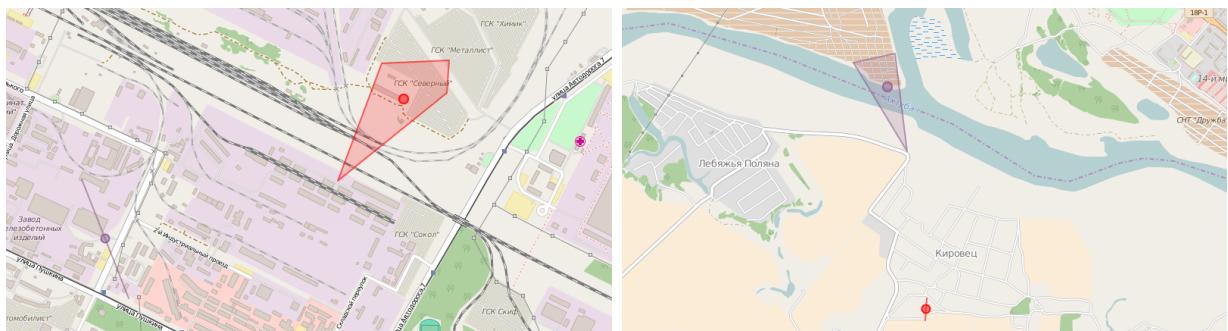


Рисунок 4.2 – Результаты кластеризации выборок Railway и River с метрикой Surface

4.1.2 Последовательная с использованием OSRM

Использование метрики Route существенно увеличивает время расчета расстояния между объектами. Использование OSRM является узким местом в реализации, однако эта метрика позволяет учитывать препятствия, результаты кластеризации выборок Railway и River представлены на рис. 4.3

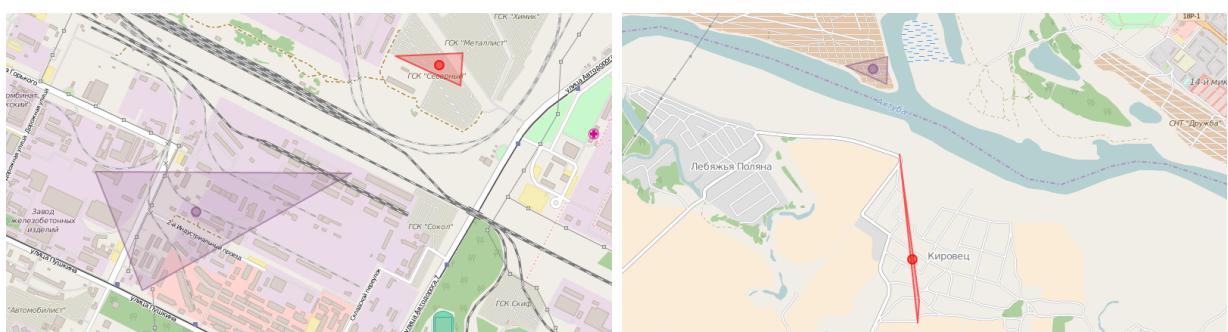


Рисунок 4.3 – Результаты кластеризации выборок Railway и River с метрикой Route

4.1.3 Параллельная с использованием OSRM

Для ускорения расчетов при использовании метрики Route предлагается распараллеливание внутреннего цикла по X (шаги 2.2.1–2.2.4) алгоритма 2.13.

Распараллеливание применимо, так как результаты обработки одного объекта выборки не влияют на обработку других объектов.

4.1.4 Результаты

Результаты обработки выборки Main представлены на рисунках 4.4, 4.5.

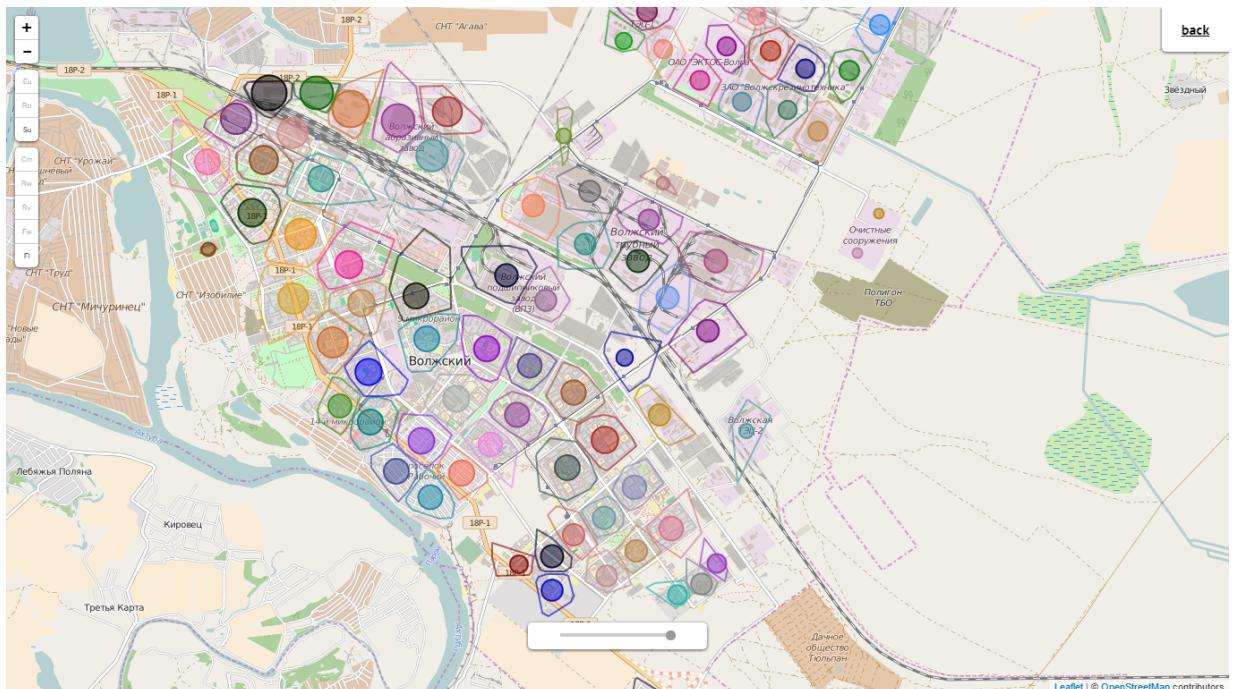


Рисунок 4.4 – Результаты обработки выборки Main метрикой Surface

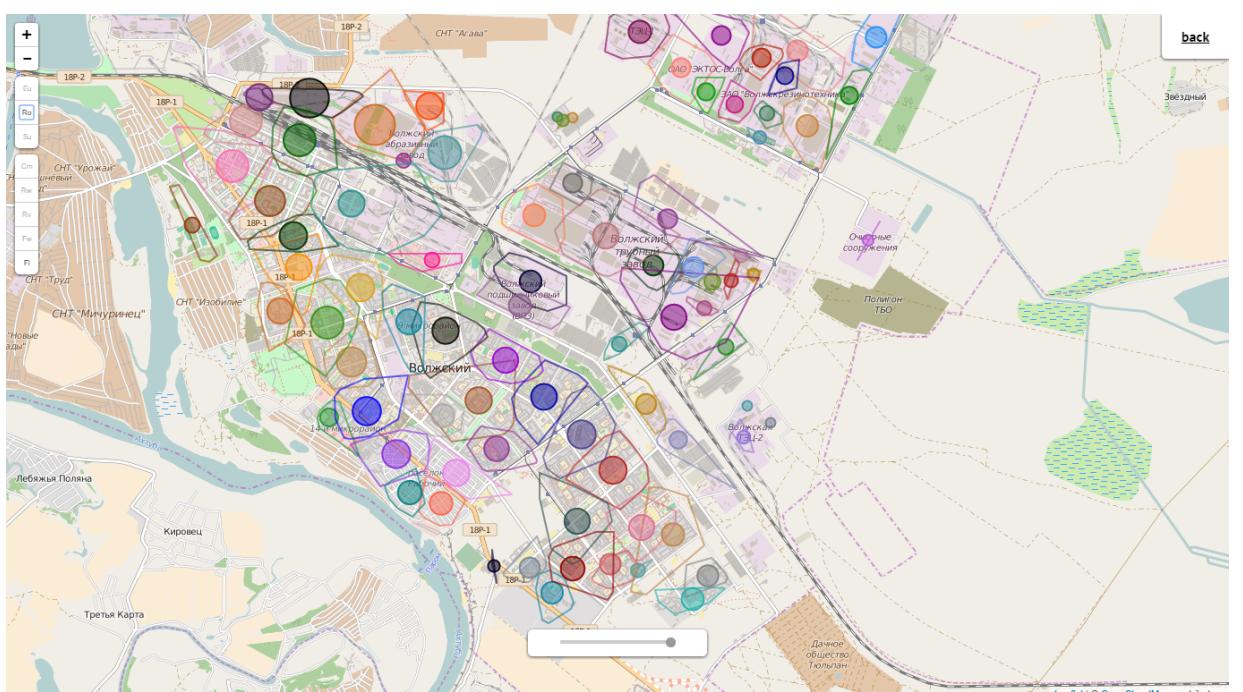


Рисунок 4.5 – Результаты обработки выборки Main метрикой Route

В таблице 4.1 приведены средние длительности выполнения одного рас-

чета дистанции между геопространственными объектами для трех реализаций алгоритма: последовательной, параллельной на два потока и параллельной на 4 потока, и для трех метрик: Euclid, Surface и Route. Метрика Euclid является обычной евклидовой метрикой: $\rho(a, b) = \sqrt{(a.x - b.x)^2 + (a.y - b.y)^2}$ и приводится для сравнения. Величины приведены в миллисекундах.

Таблица 4.1 – Среднее время выполнения одного расчета расстояния при различных метриках и реализациях алгоритма, мс

Реализация	Euclid	Surface	Route
Последовательная	0,0665	0,709	4,785
Параллельная (2)	0,0659	0,692	4,069
Параллельная (4)	0,0654	0,663	3,596

Результат работы алгоритма со сдвигом центров кластеров на ближайший узел графа дорожной сети приведен на рис. 4.6:

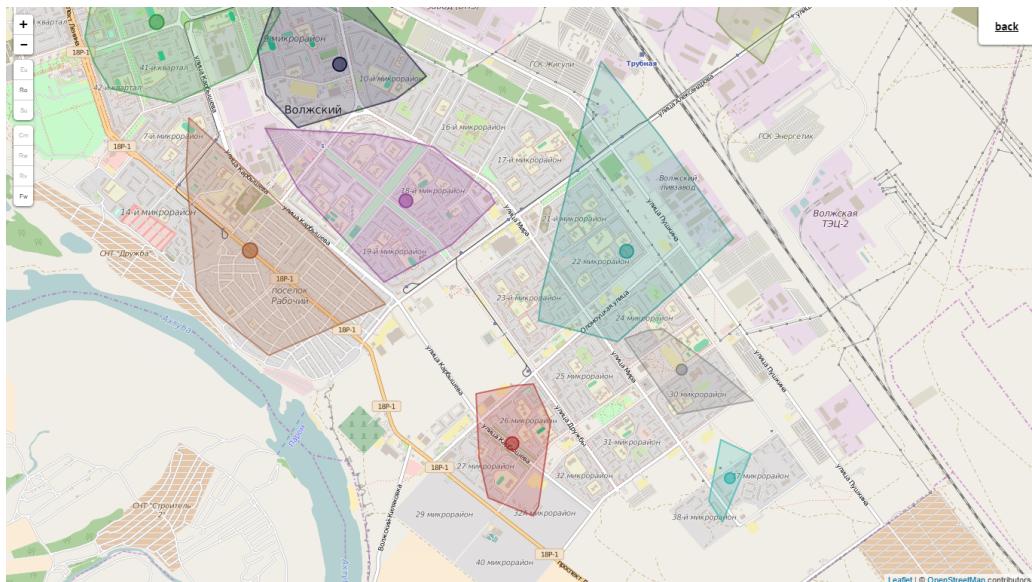


Рисунок 4.6 – Результат работы алгоритма со сдвигом центров кластеров к дорожной сети

4.2 Обсуждение результатов

Из рис. 4.2 видно, что метрика Surface не способна учитывать препятствия при расчетах, в то время как метрика Route учитывает их (рис. 4.3). То, что метрика Surface не учитывает препятствия, видно и на рисунке 4.4.

На рисунке 4.5 видно, что кластерные области пересекаются. Это происходит из-за того, что при расчете расстояния сервис OSRM выбирает бли-

жайшие к элементам выборки участки дорожной сети, а затем строит маршрут по графу дорог между ними. Из-за этого элементы, расположенные, например, по разные стороны домов могут попасть на различные участки дорог, и расстояние между ними и центром кластера получается различным. Это также происходит и из-за обхода метрикой препятствий, из-за чего близкие точки оказываются в разных кластерах и создается наложение областей кластеров. Поскольку элементы, принадлежащие одному кластеру, на рисунках показаны в виде выпуклых оболочек, то из-за нескольких элементов кластера, лежащих в стороне от других, создается видимое наложение областей кластеров.

Одна итерация алгоритма при кластеризации выборки Main требует $12\,000 \times 125 = 1,5$ млн расчетов дистанций между объектами выборки и центрами кластеров. При использовании метрики Surface кластеризация была выполнена за 28 итераций алгоритма, то есть потребовалось 42 млн расчетов. При использовании метрики Route кластеризация была выполнена за 21 итерацию алгоритма, то есть потребовала 31,5 млн расчетов расстояний. Таким образом, соотнеся эти сведения с результатами из таблицы 4.1, на одну итерацию с метрикой Surface понадобилось примерно 16,5 минут, на всю кластеризацию — около 8 часов. На одну итерацию с метрикой Route понадобилось примерно 1,5 часа, а на всю кластеризацию — 31,5 часа или почти полтора дня. Для сравнения, кластеризация с евклидовой метрикой потребовала 19 итераций, на выполнение которых было потрачено чуть больше получаса. Результат кластеризации выборки Main евклидовой метрикой приведен на рис. 4.7.

На рисунке 4.6 видно, что некоторые кластерные центры сдвинулись к магистралям, а некоторые остались в жилых кварталах. Так происходит из-за отсутствия возможности детального рассмотрения кандидатов на расположение центра кластера — участков дорожной сети — средствами OSRM.

4.2.1 Выводы

Подводя итог написанному в предыдущем разделе, можно сказать, что для кластеризации геораспределенных данных предлагаемый подход эффективен по нескольким причинам. Во-первых, ключевой особенностью подхода является использование расстояний между объектами по графу городских дорог позволяет учитывать все препятствия, расположенные между ними. Это позволяет учитывать при кластеризации ситуации, изображенные на рисунке 2.14. Во-вторых, это также позволяет выполнять основную функцию си-

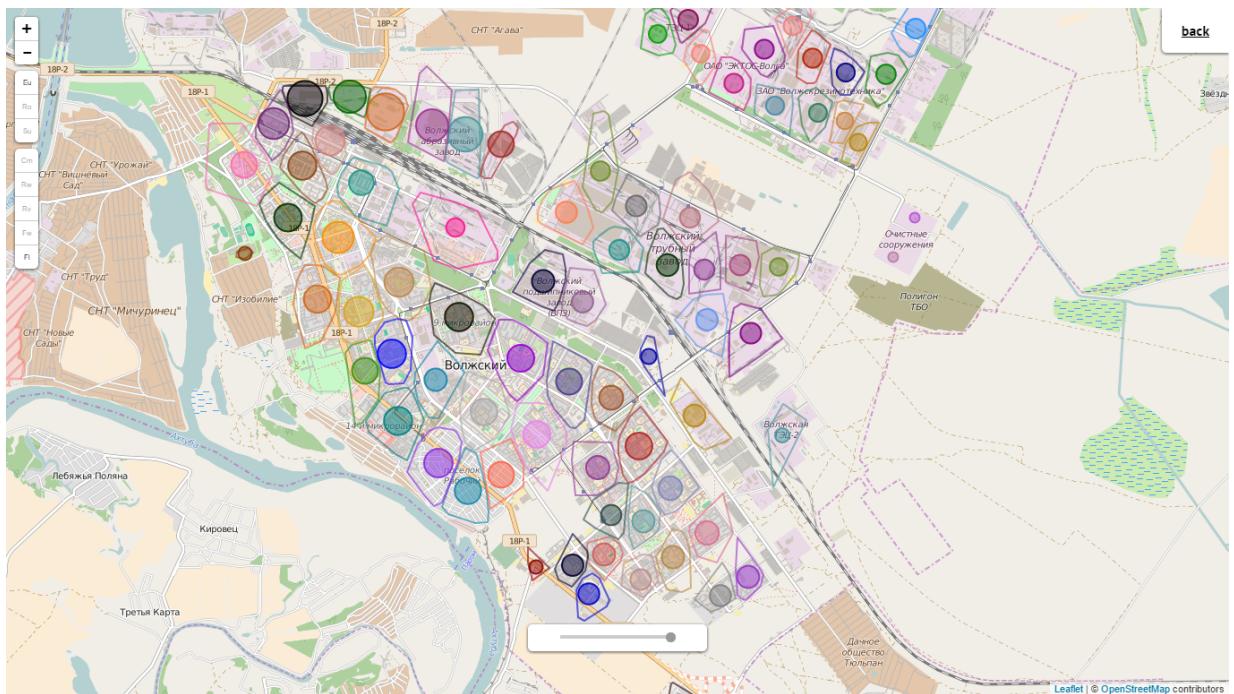


Рисунок 4.7 – Результаты обработки выборки Main евклидовой метрикой

стемы — создание остановочных пунктов, расположенных на участках дорог. В-третьих, используется алгоритм k-средних, обладающий невысокой вычислительной сложностью и отсутствием привязки к плотности данных.

Недостатки данного подхода в большей степени связаны со временем, которое требуется сервису OSRM на расчет дистанции. Этот недостаток, вероятно, можно частично сгладить использованием одного из ускоряющих методов из раздела 1.2. Также недостатком подхода можно считать определение пользователем числа k кластеров, однако такой вариант задавания числа кластеров гораздо нагляднее и удобнее для пользователя, чем определение, например, параметра ширины окна в алгоритме Mean Shift. Способом устранить этот недостаток может быть использование предполагающих методов из раздела 1.2, например, использование метода силуэтов. Недостаток, связанный с привязкой остановочных пунктов к дорожным узлам внутри жилых кварталов, связан с отсутствием описания точек в ответе сервиса OSRM на запрос. Этот недостаток может быть устранен поиском предложенных OSRM узлов дорожной сети в структуре карты OpenStreetMap, и добавления условия по тэгу (англ. tag) с ключом highway.

В целом, данный подход может быть использован для кластеризации геораспределенных данных и для построения сети остановочных пунктов общественного транспорта, особенно, если на рассматриваемой местности на-

ходятся какие-либо естественные или искусственные препятствия, или время кластеризации не столь важно, как получаемый результат.

В рамках магистерской диссертации был разработан метод кластеризации предпочтений жителей по перемещению, выраженных в форме пары геораспределенных объектов «пункт отправления–пункт назначения». Для этого были использованы алгоритм кластеризации k-средних (подробно описан в разделе 2.3) и метрика Route, рассчитывающая расстояния между геораспределенными объектами по графу дорог, для чего использующая сервис маршрутизации Open Source Routing Machine (подробно описана в разделе 2.4). Было сделано четыре реализации метода: последовательная, последовательная с использованием OSRM, параллельная и параллельная с использованием OSRM. В последовательной и параллельной реализациях в качестве метрики используется метрика Surface, представляющая собой решение обратной геодезической задачи.

Для оценки эффективности работы предложенного метода было сгенерировано четыре тестовые выборки, представленные на рисунках 3.1, 3.2, 4.1. Одна выборка используется для тестирования работы алгоритма на псевдореалистичных данных, две выборки используются для проверки метрикой учета препятствий, и последняя — для тестов алгоритма на скорость.

Полученные результаты, приведенные в разделе 4.1.4 показали, что метрика с использованием OSRM позволяет учитывать наличие препятствий между объектами, в то время как вторая метрика этого делать не может. Однако, из таблицы 4.1 видно, что время расчета расстояния метрикой Route существенно (в 5-6 раз) больше времени, затрачиваемого метрикой Surface, и в десятки раз больше времени, затрачиваемого евклидовой метрикой. Отсюда был сделан вывод, что предложенный метод может быть использован для кластеризации геораспределенных данных, если на рассматриваемой местности находятся какие-либо естественные или искусственные препятствия, или если время кластеризации не играет большой роли, а важен именно результат работы алгоритма. В разделах 4.2 и 4.2.1 приведен более подробный анализ результатов работы.

Результаты данной работы используются для построения сети остановочных пунктов общественного транспорта, по которым будет проводится построение маршрутов общественного транспорта.

- 1 Evaluating the sustainability of Volgograd / N. Sadovnikova, D. Parygin, E. Gnedkova, B. Sanzhapov, and N. Gidkova. // In The Sustainable City VIII. WIT Press, 2013.
- 2 Нестерова А. Новая маршрутная сеть г. Томска представлена общественности [Электронный ресурс] // Сетевое издание Центр дорожной информации. — 2015. — Режим доступа: <http://road.perm.ru/index.php?id=1475> (дата обращения: 01.12.2015).
- 3 Nielsen G., Lange T. Network Design for Public Transport Success – theory and examples // Norwegian Ministry of Transport and Communications, Oslo. – 2008.
- 4 Воронцов К.В. Машинное обучение. Курс лекций [Электронный ресурс]. — 2011. — Режим доступа: <http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf> (дата обращения: 25.03.2016).
- 5 Fraley C., Raftery A. E. Model-based clustering, discriminant analysis, and density estimation // Journal of the American statistical Association. — 2002. — V. 97. — N. 458. — P. 611–631.
- 6 Elkan C. Using the triangle inequality to accelerate k-means // ICML. — 2003. — V. 3. — P. 147–153.
- 7 Kanungo T. et al. An efficient k-means clustering algorithm: Analysis and implementation // Pattern Analysis and Machine Intelligence, IEEE Transactions on. — 2002. — V. 24. — N. 7. — P. 881–892.
- 8 Likas A., Vlassis N., Verbeek J. J. The global k-means clustering algorithm // Pattern recognition. — 2003. — V. 36. — N. 2. — P. 451–461.
- 9 Ткаченко О. М. и др. Метод кластеризации на основе последовательного запуска k-средних с усовершенствованным выбором кандидата на новую позицию вставки //Научные труды Винницкого национального технического университета. — 2012. — № 2.

- 10 Лисин А. В., Файзуллин Р. Т. Применение метаэвристических алгоритмов к решению задач кластеризации методом k-средних // Компьютерная оптика. — 2015. — Т. 39. — № 3. — С. 406–412.
- 11 Wang J. et al. Fast approximate k-means via cluster closures //Multimedia Data Mining and Analytics. — Springer International Publishing, 2015. — P. 373–395.
- 12 Krishna K., Murty M. N. Genetic K-means algorithm //Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on. — 1999. — V. 29. — N. 3. — P. 433–439.
- 13 Lai J. Z. C., Huang T. J., Liaw Y. C. A fast k-means clustering algorithm using cluster center displacement // Pattern Recognition. — 2009. — V. 42. — N. 11. — P. 2551–2556.
- 14 Pelleg D., Moore A. Accelerating exact k-means algorithms with geometric reasoning // Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. — ACM, 1999. — P. 277–281.
- 15 Hussein N. A Fast Greedy K-Means Algorithm. Master's Thesis. // University of Amsterdam, Amsterdam. — 2002.
- 16 Arthur D., Vassilvitskii S. k-means++: The advantages of careful seeding // Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. — Society for Industrial and Applied Mathematics, 2007. — P. 1027–1035.
- 17 Duman S., Güvenç U., Yörükeren N. Gravitational Search Algorithm for Economic Dispatch with Valve-Point Effects. // International Review of Electrical Engineering 2010. — V. 5. — P. 2890–2895.
- 18 Nister D., Stewenius H. Scalable recognition with a vocabulary tree // Computer vision and pattern recognition, 2006 IEEE computer society conference on. — IEEE, 2006. — V. 2. — P. 2161–2168.
- 19 Philbin J. et al. Object retrieval with large vocabularies and fast spatial matching // Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on. — IEEE, 2007. — P. 1-8.

- 20 Петров С. А. и др. Гібридний алгоритм кластер-аналізу для формування априорного розбиття простору ознак на класи знань в системах дистанційного навчання. // Вісник Вінницького політехнічного інституту. — № 4. — 2015. — С. 80–87.
- 21 Pelleg D., Moore A. W. X-means: Extending K-means with Efficient Estimation of the Number of Clusters // Proceeding ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning. — 2000. — V. 1. — P. 727–734.
- 22 Sugar C. A., James G. M. Finding the number of clusters in a dataset // Journal of the American Statistical Association. — V. 98. — N. 463. — 2003. — P. 750–763.
- 23 Rousseeuw P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis // Journal of computational and applied mathematics. — 1987. — V. 20. — P. 53–65.
- 24 de Amorim R. C., Hennig C. Recovering the number of clusters in data sets with noise features using feature rescaling factors // Information Sciences. — 2015. — V. 324. — P. 126–145.
- 25 Lletí R. et al. Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes // Analytica Chimica Acta. — 2004. — V. 515. — N. 1. — P. 87–100.
- 26 Тханг В. В., Пантюхин Д. В., Галушкин А. И. Гибридный алгоритм кластеризации FastDBSCAN. // Труды Московского Физико-Технического Института. — 2015. — Т. 7. — № 3. — С. 77–81.
- 27 Comaniciu D., Meer P. Mean shift: A robust approach toward feature space analysis // Pattern Analysis and Machine Intelligence, IEEE Transactions on. — 2002. — V. 24. — N. 5. — P. 603–619.
- 28 Charikar M. et al. Incremental clustering and dynamic information retrieval // SIAM Journal on Computing. — 2004. — V. 33. — N. 6. — P. 1417–1440.
- 29 Huang Z. A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining // DMKD. — 1997. — 8 p.

- 30 Zhang T., Ramakrishnan R., Livny M. BIRCH: an efficient data clustering method for very large databases // ACM Sigmod Record. — ACM, 1996. — V. 25. — N. 2. — P. 103–114.
- 31 Tung A. K. H., Hou J., Han J. Spatial clustering in the presence of obstacles // Data Engineering, 2001. Proceedings 17th International Conference on. — IEEE, 2001. — P. 359–367.
- 32 Han J., Kamber, M., Tung, A. K. H. Spatial Clustering Methods in Data Mining: A Survey. // Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS. — 2001. — P. 201–231.
- 33 Koperski K., Han J., Adhikary J. Mining knowledge in geographical data // Communications of ACM. — 1998. — V. 26.
- 34 Estivill-Castro V., Lee I. Argument free clustering for large spatial point-data sets via boundary extraction from Delaunay Diagram // Computers, Environment and urban systems. — 2002. — V. 26. — N. 4. — P. 315–334.
- 35 Estivill-Castro V., Lee I. Clustering with obstacles for geographical data mining // ISPRS Journal of Photogrammetry and Remote Sensing. — 2004. — V. 59. — N. 1. — P. 21–34.
- 36 Karney C. F. F. Algorithms for geodesics // Journal of Geodesy. — 2013. — V. 87. — N. 1. — P. 43–55.
- 37 Cheng Y. Mean shift, mode seeking, and clustering // Pattern Analysis and Machine Intelligence, IEEE Transactions on. — 1995. — V. 17. — N. 8. — P. 790–799.
- 38 Golubev A. et al. Strategway: web solutions for building public transportation routes using big geodata analysis // Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services. — ACM, 2015. — P. 91.
- 39 Angelov P. Autonomous Learning Systems: From Data Streams to Knowledge in Real-time. — John Wiley & Sons, 2013. — 273 p.
- 40 Автоматическая обработка текстов на естественном языке и компьютерная лингвистика: учеб. пособие / Большакова Е.И., Клышинский Э.С., Ландэ Д.В., Носков А.А., Пескова О.В., Ягунова Е.В. — М.: МИЭМ, 2011. — 272 с.

41 Нейский И. М. Классификация и сравнение методов кластеризации // Интеллектуальные технологии и системы. Сборник учебно-методических работ и статей аспирантов и студентов. — М.: НОК «CLAIM», 2006. — Выпуск 8. — С. 130–142.

42 MacQueen J. et al. Some methods for classification and analysis of multivariate observations // Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. — 1967. — V. 1. — N. 14. — P. 281–297.

43 GeographicLib — a small set of C++ classes for converting between geographic, UTM, UPS, MGRS, and geocentric coordinates [Электронный ресурс]. — 2015. — Режим доступа: <http://geographiclib.sourceforge.net/> (дата обращения: 14.11.2015).

44 Open Source Routing Machine [Электронный ресурс]. — 2014. — Режим доступа: <http://project-osrm.org/> (дата обращения: 25.11.2014).

45 The Free Wiki World Map — An openly licensed map of the world being created by volunteers using local knowledge, GPS tracks and donated sources [Электронный ресурс]. — 2014. — Режим доступа:<https://www.openstreetmap.org> (Дата обращения: 25.11.2014).

46 scikit-learn. Machine Learning in Python. Simple and efficient tools for data mining and data analysis [Электронный ресурс]. — 2015. — Режим доступа: <http://scikit-learn.org/> (дата обращения: 17.03.2015).

47 E.M. Mirkes, K-means and K-medoids applet [Электронный ресурс]. — University of Leicester, 2011. — Режим доступа: http://www.math.le.ac.uk/people/ag153/homepage/KmeansKmedoids/Kmeans_Kmedoids.html (дата обращения: 17.03.2015).

48 k-means clustering — Wikipedia, the free encyclopedia [Электронный ресурс]. — 2014. — Режим доступа: https://en.wikipedia.org/wiki/K-means_clustering (дата обращения: 25.11.2014).

49 Project "Scatter— generate geospatial data [Электронный ресурс]. — 2015. — Режим доступа: <https://vstu-cad-stuff.github.io/scatter/> (Дата обращения: 15.03.2015).

Приложение А
Исходный код

Ссылка на исходный код:

<https://github.com/vstu-cad-stuff/clustering/tree/master>.

Ссылка на демонстрационные примеры:

<https://vstu-cad-stuff.github.io/clustering/>.

Модуль по настройке кластеризации и ее запуску:

```

1 from DataCollector import DataCollector
2 from KMeansMachine import KMeansClusteringMachine as kmeans
3 from KMeansMachineTriangle import KMeansClusteringMachine as kmeans_triangle
4 from argparse import ArgumentParser
5
6 parser = ArgumentParser(description='k-Means clustering.')
7 parser.add_argument('-m', '--metric', help='Used metric', choices=['route', 'euclid', 'surface'], default='route')
8 parser.add_argument('-s', '--sample', help='Used sample.\n          Points are loaded from "data/SAMPLE_pts.txt" if "--points" is not set.\n          If init type is file, clusters are loaded from "data/SAMPLE_cls" if "--clusters" is not set.', default='common')
9
10 parser.add_argument('-i', '--iteration', type=int, help='Iterations count.', default=100)
11 parser.add_argument('-t', '--init', help='Type of init.', choices=['file', 'random', 'grid'], default='file')
12 parser.add_argument('-p', '--threads', type=int, help='Number of threads', default=4)
13 parser.add_argument('-l', '--nolog', help='Do not log iteration result', action='store_true')
14 parser.add_argument('--use_triangle', help='Use triangle inequality to reduce calculations.', action='store_true')
15 parser.add_argument('--stations', help='Locate clusters to roadmap network.', action='store_true')
16 parser.add_argument('--clusters', help='Load clusters from CLUSTERS.')
17 parser.add_argument('--points', help='Load points from POINTS.')
18 parser.add_argument('--n', '--count', type=int, help='Number of clusters if "init" is set to "random"', default=40)
19 parser.add_argument('--g', '--grid', type=int, nargs=2, help='Size of grid if "init" is set to "grid"', default=[7, 7])
20 parser.add_argument('--start', help="Result of previous work, points file.\n          Continue calculations from this step.")
21
22 args = parser.parse_args()
23
24 USE_TRIANGLE_INEQUALITY = args.use_triangle
25 test = args.sample.lower()
26 metric = args.metric.lower() # route, surface, euclid

```

```

28 iterations_count = args.iteration
29 thread_count = args.threads
30 _continue = args.start
31 stations = args.stations
32
33 datafile = 'data/{}_pts.txt'.format(test) if args.points is None else args.
    points
34 init = args.init
35 filename = 'data/{}_cls.txt'.format(test) if args.clusters is None else args.
    clusters
36 if args.nolog:
37     log = False
38 else:
39     if args.init is 'file':
40         if args.clusters is not None:
41             log = args.clusters.split('.')[0] + '_log'
42             if '/' in log:
43                 log = log.split('/')[-1]
44         else:
45             log = test
46     else:
47         if args.points is not None:
48             log = args.points.split('.')[0] + '_log'
49             if '/' in log:
50                 log = log.split('/')[-1]
51         else:
52             log = args.init + '_log'
53
54 random_count = args.count
55 grid_size = args.grid
56
57 # create DataCollector object
58 dc = DataCollector()
59 # upload data from 'data.txt' file
60 dc.uploadFromTextFile(datafile, delimiter=',')
61 # get data from dc object
62 X = dc.getData()
63
64 # create KMeansClusteringMachine object with specified parameters
65 if USE_TRIANGLE_INEQUALITY:
66     km = kmeans_triangle(X, init=init, filename=filename, max_iter=
        iterations_count,
                           log=log, count=random_count, gridSize=grid_size)
67 else:
68     km = kmeans(X, init=init, filename=filename, max_iter=iterations_count,
69                  log=log, count=random_count, grid_size=grid_size,
70                  thread_count=thread_count, start=_continue, stations=
                    stations)
72 # perform clustering

```

```
73 km.fit(metric)
74 # print info
75 print('Fit time: {}, clusters: {}'.format(km.fit_time, km.n_cluster))
76 # export init centers to 'k_init.txt'
77 # km.initM.exportCentersToTextFile('init.txt')
78 if not log:
79     # export centers to 'centers.js'
80     km.exportCentersToTextFile('{}_cls.js'.format(test))
81     # export points to 'points.js'
82     km.exportPointsToTextFile('{}_pts.js'.format(test))
```

Модуль кластеризации:

```

1  from __future__ import division, print_function
2  import geojson as json
3  import time
4  import numpy as np
5  import os
6  from geographiclib.geodesic import Geodesic
7
8  from routelib import route
9  from InitMachine import InitMachine
10 from ClusteringMachine import ClusteringMachine
11
12 from multiprocessing.dummy import Pool as ThreadPool
13 THREADS = 4
14 POOL = ThreadPool(processes=THREADS)
15
16 def async_worker(iterator, func, data=None):
17     thread_list = []
18     result = []
19     for item in iterator:
20         # create job for new_route function
21         thread = POOL.apply_async(func, (item, data))
22         # add thread in list
23         thread_list.append(thread)
24     for thread in thread_list:
25         # get result of job
26         result.append(thread.get())
27     return result
28
29 def dump(data, filename):
30     try:
31         with open(filename, 'w') as file_:
32             json.dump(data, file_)
33     except IOError as e:
34         print('{}'.format(e))
35
36 class KMeans():
37     """ K-Means clustering.
38
39     Attributes
40     -----
41     max_iter_ : int
42         Maximum iteration number. After reaching it, clustering is
43         considered as completed.
44     cluster_centers_ : array, [n_clusters, n_dimensions + 1]
45         Centers of clusters.
46     labels_ : array, [n_points]
47         Labels of points.
48

```

```

49     Parameters
50
51     max_iter : int
52         Set maximum iteration number.
53
54     max_iter_ = None
55     cluster_centers_ = None
56     labels_ = None
57     population_ = None
58     log = False
59     _continue = False
60     route_ = None
61     icntr = 0
62
63     def __init__(self, max_iter, log, start, stations):
64         self.max_iter_ = max_iter
65         self.log = log
66         self._continue = start
67         self.stations = stations
68         self.route_ = route()
69
70     def dist(self, a, b, metric):
71         """ Calculate distance between two points.
72
73         Parameters
74
75             a : array, [n_dimensions]
76                 First point.
77             b : array, [n_dimensions]
78                 Second point.
79
80         Returns
81
82             r : float
83                 Distance between points.
84
85             if metric == 'route':
86                 r = self.route_.route_distance(a, b)
87             elif metric == 'euclid':
88                 r = np.sqrt((b[0] - a[0]) ** 2 + (b[1] - a[1]) ** 2)
89             elif metric == 'surface':
90                 r = Geodesic.WGS84.Inverse(a[0], a[1], b[0], b[1])['s12']
91                 #d = np.arccos(np.sin(a[0]) * np.sin(b[0]) + np.cos(a[0]) * np.cos(b
92                 #    [0]) * np.cos(b[1] - a[1]))
93                 #R = 6371. # Earth radius
94                 #r = d * R
95             else:
96                 raise ValueError('Unknown metric: {}'.format(metric))
97             self.icntr += 1

```

```

97     text = '      progress: {:.2f}k / {:.2f}k'.format(self.icntr / 1000,
98           self.x_len * self.c_len / 1000)
99     digits = len(text)
100    delete = '\r' * digits
101    print('{0}{1}'.format(delete, text), end=' ')
102    return r
103
103  def stop(self, iter, old, new, lold, lnew):
104      """ Check whenever clustering needs to be stopped.
105
106      Parameters
107      -----
108      iter : int
109          Number of current iteration.
110      old : array, [n_clusters, n_dimensions + 1]
111          Centers of clusters on previous iteration.
112      new : array, [n_clusters, n_dimensions + 1]
113          Centers of clusters on current iteration.
114
115      Returns
116      -----
117      stop : boolean
118          If true, clustering is completed.
119      """
120      if iter >= self.max_iter_:
121          return True
122      return np.array_equal(old, new) or np.array_equal(lold, lnew)
123
124  def cloop(self, i, j):
125      return (self.dist(self.X[i], self.C[j], self.metric), j)
126
127  def xloop(self, i, _=None):
128      # calculate distance between point and all the clusters centers
129      D = list(map(lambda j: self.cloop(i, j), range(self.c_len)))
130      # D = list(POOL.map(lambda j: self.cloop(i, j), range(self.c_len)))
131      # sort distances ascending
132      D.sort(key=lambda x: x[0])
133      # pick number of cluster, which center has the smallest
134      # distance to point
135      m = D[0][1]
136      # set label of point
137      self.L[i] = self.C[m][2]
138      self.P[m] += 1
139      self.A[m] = np.append(self.A[m], [self.X[i]], axis=0)
140
141  def fit(self, X, C, metric):
142      """ Perform clustering.
143
144      Parameters

```

```

145
146     X : array, [n_points, n_dimensions]
147         Coordinates of points.
148     C : array, [n_clusters, n_dimensions + 1]
149         Centers of clusters.
150
151     """
152     # set initial parameters
153     iteration = 0
154     c_old = None
155     l_old = None
156     # get length of lists
157     self.c_len = len(C)
158     self.x_len = len(X)
159     print('Now {} points will be clustering to {} clusters'.format(self.
160         x_len, self.c_len))
161     print('Threads count: {}; log: {}'.format(THREADS, self.log))
162
163     self.L = np.empty([self.x_len])
164     self.C = C
165     self.X = X
166     self.P = None
167     self.A = None
168     self.metric = metric
169     self.sleeping = 0
170
171     if self.metric == 'route' or self.stations:
172         self.route_.start()
173         self.sleeping += self.route_.sleep
174     # while clustering isn't completed
175     while not self.stop(iteration, c_old, self.C, l_old, self.L):
176         time_start = time.time()
177         # reset population in clusters
178         self.P = np.zeros([self.c_len])
179         # show iteration number
180         print('Iteration {}'.format(iteration + 1))
181         # create empty python array
182         # each item will contain all the points belongs to specific cluster
183         self.A = [np.empty([0, 2]) for i in range(self.c_len)]
184         # for each point
185         print(' assigning points')
186         l_old = np.array(self.L)
187         if not self._continue:
188             res = async_worker(range(self.x_len), self.xloop)
189             # res = list(POOL.map(self.xloop, range(self.x_len)))
190             # res = list(map(self.xloop, range(self.x_len)))
191             # equate the previous and current centers of clusters
192             c_old = self.C
193             if self.log:
194                 if self.log is not True:

```

```

193     path = '{}'.format(self.log)
194
195     else:
196         path = 'km_{}'.format(self.metric[:2])
197
198         if path == '':
199             path = '.'
200
201         else:
202             if not (os.path.exists(path)):
203                 os.makedirs(path)
204
205             cc = self.C
206
207             cc = list(map(lambda x, y: (np.append(x, y)).tolist(), cc,
208                           self.P))
209
210             filename = '{}/{}_centers_{}.js'.format(path, self.metric
211                                         [0], iteration + 1)
212
213             dump(cc, filename)
214
215
216             xc = self.X
217
218             xc = list(map(lambda x, y: (np.append(x, y)).tolist(), xc,
219                           self.L))
220
221             filename = '{}/{}_points_{}.js'.format(path, self.metric[0],
222                                         iteration + 1)
223
224             dump(xc, filename)
225
226         else:
227             self.L = np.array([])
228
229             for i in json.load(open(self._continue)):
230
231                 self.L = np.append(self.L, i[2])
232
233                 self.A[int(i[2])] = np.append(self.A[int(i[2])], [[i[0], i
234                                         [1]]], axis=0)
235
236             self._continue = False
237
238
239             # array for calculated centers of clusters
240             mu = np.empty([self.c_len, 3], dtype='object')
241
242
243             print('\n  calculating new centers')
244             # for each cluster
245             i = 0
246
247             while i < self.c_len:
248
249                 # if it contains points
250                 if self.P[i] != 0:
251
252                     # calculate center of cluster
253                     k = np.array(np.round(np.mean(self.A[i], axis=0), decimals
254                                         =5), dtype='object')
255
256                     mu[i] = np.append(k, i)
257
258                 else:
259
260                     d = np.round(self.C[i][:2].astype(np.double), decimals=5)
261
262                     mu[i] = np.append(d.astype(np.object), i)
263
264                     i += 1
265
266             if self.stations:
267
268                 print('  locating centers on roadmap')
269
270                 for c in range(self.c_len):
271
272                     if self.C[c][0] > 0:
273
274                         if self.C[c][1] > 0:
275
276                             if self.C[c][2] > 0:
277
278                                 if self.C[c][3] > 0:
279
280                                     if self.C[c][4] > 0:
281
282                                         if self.C[c][5] > 0:
283
284                                             if self.C[c][6] > 0:
285
286                                                 if self.C[c][7] > 0:
287
288                                                     if self.C[c][8] > 0:
289
290                                                         if self.C[c][9] > 0:
291
292                                                             if self.C[c][10] > 0:
293
294                                                               if self.C[c][11] > 0:
295
296                                                               if self.C[c][12] > 0:
297
298                                                               if self.C[c][13] > 0:
299
300                                                               if self.C[c][14] > 0:
301
302                                                               if self.C[c][15] > 0:
303
304                                                               if self.C[c][16] > 0:
305
306                                                               if self.C[c][17] > 0:
307
308                                                               if self.C[c][18] > 0:
309
310                                                               if self.C[c][19] > 0:
311
312                                                               if self.C[c][20] > 0:
313
314                                                               if self.C[c][21] > 0:
315
316                                                               if self.C[c][22] > 0:
317
318                                                               if self.C[c][23] > 0:
319
320                                                               if self.C[c][24] > 0:
321
322                                                               if self.C[c][25] > 0:
323
324                                                               if self.C[c][26] > 0:
325
326                                                               if self.C[c][27] > 0:
327
328                                                               if self.C[c][28] > 0:
329
330                                                               if self.C[c][29] > 0:
331
332                                                               if self.C[c][30] > 0:
333
334                                                               if self.C[c][31] > 0:
335
336                                                               if self.C[c][32] > 0:
337
338                                                               if self.C[c][33] > 0:
339
340                                                               if self.C[c][34] > 0:
341
342                                                               if self.C[c][35] > 0:
343
344                                                               if self.C[c][36] > 0:
345
346                                                               if self.C[c][37] > 0:
347
348                                                               if self.C[c][38] > 0:
349
350                                                               if self.C[c][39] > 0:
351
352                                                               if self.C[c][40] > 0:
353
354                                                               if self.C[c][41] > 0:
355
356                                                               if self.C[c][42] > 0:
357
358                                                               if self.C[c][43] > 0:
359
360                                                               if self.C[c][44] > 0:
361
362                                                               if self.C[c][45] > 0:
363
364                                                               if self.C[c][46] > 0:
365
366                                                               if self.C[c][47] > 0:
367
368                                                               if self.C[c][48] > 0:
369
370                                                               if self.C[c][49] > 0:
371
372                                                               if self.C[c][50] > 0:
373
374                                                               if self.C[c][51] > 0:
375
376                                                               if self.C[c][52] > 0:
377
378                                                               if self.C[c][53] > 0:
379
380                                                               if self.C[c][54] > 0:
381
382                                                               if self.C[c][55] > 0:
383
384                                                               if self.C[c][56] > 0:
385
386                                                               if self.C[c][57] > 0:
387
388                                                               if self.C[c][58] > 0:
389
390                                                               if self.C[c][59] > 0:
391
392                                                               if self.C[c][60] > 0:
393
394                                                               if self.C[c][61] > 0:
395
396                                                               if self.C[c][62] > 0:
397
398                                                               if self.C[c][63] > 0:
399
400                                                               if self.C[c][64] > 0:
401
402                                                               if self.C[c][65] > 0:
403
404                                                               if self.C[c][66] > 0:
405
406                                                               if self.C[c][67] > 0:
407
408                                                               if self.C[c][68] > 0:
409
410                                                               if self.C[c][69] > 0:
411
412                                                               if self.C[c][70] > 0:
413
414                                                               if self.C[c][71] > 0:
415
416                                                               if self.C[c][72] > 0:
417
418                                                               if self.C[c][73] > 0:
419
420                                                               if self.C[c][74] > 0:
421
422                                                               if self.C[c][75] > 0:
423
424                                                               if self.C[c][76] > 0:
425
426                                                               if self.C[c][77] > 0:
427
428                                                               if self.C[c][78] > 0:
429
430                                                               if self.C[c][79] > 0:
431
432                                                               if self.C[c][80] > 0:
433
434                                                               if self.C[c][81] > 0:
435
436                                                               if self.C[c][82] > 0:
437
438                                                               if self.C[c][83] > 0:
439
440                                                               if self.C[c][84] > 0:
441
442                                                               if self.C[c][85] > 0:
443
444                                                               if self.C[c][86] > 0:
445
446                                                               if self.C[c][87] > 0:
447
448                                                               if self.C[c][88] > 0:
449
450                                                               if self.C[c][89] > 0:
451
452                                                               if self.C[c][90] > 0:
453
454                                                               if self.C[c][91] > 0:
455
456                                                               if self.C[c][92] > 0:
457
458                                                               if self.C[c][93] > 0:
459
460                                                               if self.C[c][94] > 0:
461
462                                                               if self.C[c][95] > 0:
463
464                                                               if self.C[c][96] > 0:
465
466                                                               if self.C[c][97] > 0:
467
468                                                               if self.C[c][98] > 0:
469
470                                                               if self.C[c][99] > 0:
471
472                                                               if self.C[c][100] > 0:
473
474                                                               if self.C[c][101] > 0:
475
476                                                               if self.C[c][102] > 0:
477
478                                                               if self.C[c][103] > 0:
479
480                                                               if self.C[c][104] > 0:
481
482                                                               if self.C[c][105] > 0:
483
484                                                               if self.C[c][106] > 0:
485
486                                                               if self.C[c][107] > 0:
487
488                                                               if self.C[c][108] > 0:
489
490                                                               if self.C[c][109] > 0:
491
492                                                               if self.C[c][110] > 0:
493
494                                                               if self.C[c][111] > 0:
495
496                                                               if self.C[c][112] > 0:
497
498                                                               if self.C[c][113] > 0:
499
500                                                               if self.C[c][114] > 0:
501
502                                                               if self.C[c][115] > 0:
503
504                                                               if self.C[c][116] > 0:
505
506                                                               if self.C[c][117] > 0:
507
508                                                               if self.C[c][118] > 0:
509
510                                                               if self.C[c][119] > 0:
511
512                                                               if self.C[c][120] > 0:
513
514                                                               if self.C[c][121] > 0:
515
516                                                               if self.C[c][122] > 0:
517
518                                                               if self.C[c][123] > 0:
519
520                                                               if self.C[c][124] > 0:
521
522                                                               if self.C[c][125] > 0:
523
524                                                               if self.C[c][126] > 0:
525
526                                                               if self.C[c][127] > 0:
527
528                                                               if self.C[c][128] > 0:
529
530                                                               if self.C[c][129] > 0:
531
532                                                               if self.C[c][130] > 0:
533
534                                                               if self.C[c][131] > 0:
535
536                                                               if self.C[c][132] > 0:
537
538                                                               if self.C[c][133] > 0:
539
540                                                               if self.C[c][134] > 0:
541
542                                                               if self.C[c][135] > 0:
543
544                                                               if self.C[c][136] > 0:
545
546                                                               if self.C[c][137] > 0:
547
548                                                               if self.C[c][138] > 0:
549
550                                                               if self.C[c][139] > 0:
551
552                                                               if self.C[c][140] > 0:
553
554                                                               if self.C[c][141] > 0:
555
556                                                               if self.C[c][142] > 0:
557
558                                                               if self.C[c][143] > 0:
559
560                                                               if self.C[c][144] > 0:
561
562                                                               if self.C[c][145] > 0:
563
564                                                               if self.C[c][146] > 0:
565
566                                                               if self.C[c][147] > 0:
567
568                                                               if self.C[c][148] > 0:
569
570                                                               if self.C[c][149] > 0:
571
572                                                               if self.C[c][150] > 0:
573
574                                                               if self.C[c][151] > 0:
575
576                                                               if self.C[c][152] > 0:
577
578                                                               if self.C[c][153] > 0:
579
580                                                               if self.C[c][154] > 0:
581
582                                                               if self.C[c][155] > 0:
583
584                                                               if self.C[c][156] > 0:
585
586                                                               if self.C[c][157] > 0:
587
588                                                               if self.C[c][158] > 0:
589
590                                                               if self.C[c][159] > 0:
591
592                                                               if self.C[c][160] > 0:
593
594                                                               if self.C[c][161] > 0:
595
596                                                               if self.C[c][162] > 0:
597
598                                                               if self.C[c][163] > 0:
599
600                                                               if self.C[c][164] > 0:
601
602                                                               if self.C[c][165] > 0:
603
604                                                               if self.C[c][166] > 0:
605
606                                                               if self.C[c][167] > 0:
607
608                                                               if self.C[c][168] > 0:
609
610                                                               if self.C[c][169] > 0:
611
612                                                               if self.C[c][170] > 0:
613
614                                                               if self.C[c][171] > 0:
615
616                                                               if self.C[c][172] > 0:
617
618                                                               if self.C[c][173] > 0:
619
620                                                               if self.C[c][174] > 0:
621
622                                                               if self.C[c][175] > 0:
623
624                                                               if self.C[c][176] > 0:
625
626                                                               if self.C[c][177] > 0:
627
628                                                               if self.C[c][178] > 0:
629
630                                                               if self.C[c][179] > 0:
631
632                                                               if self.C[c][180] > 0:
633
634                                                               if self.C[c][181] > 0:
635
636                                                               if self.C[c][182] > 0:
637
638                                                               if self.C[c][183] > 0:
639
640                                                               if self.C[c][184] > 0:
641
642                                                               if self.C[c][185] > 0:
643
644                                                               if self.C[c][186] > 0:
645
646                                                               if self.C[c][187] > 0:
647
648                                                               if self.C[c][188] > 0:
649
650                                                               if self.C[c][189] > 0:
651
652                                                               if self.C[c][190] > 0:
653
654                                                               if self.C[c][191] > 0:
655
656                                                               if self.C[c][192] > 0:
657
658                                                               if self.C[c][193] > 0:
659
660                                                               if self.C[c][194] > 0:
661
662                                                               if self.C[c][195] > 0:
663
664                                                               if self.C[c][196] > 0:
665
666                                                               if self.C[c][197] > 0:
667
668                                                               if self.C[c][198] > 0:
669
670                                                               if self.C[c][199] > 0:
671
672                                                               if self.C[c][200] > 0:
673
674                                                               if self.C[c][201] > 0:
675
676                                                               if self.C[c][202] > 0:
677
678                                                               if self.C[c][203] > 0:
679
680                                                               if self.C[c][204] > 0:
681
682                                                               if self.C[c][205] > 0:
683
684                                                               if self.C[c][206] > 0:
685
686                                                               if self.C[c][207] > 0:
687
688                                                               if self.C[c][208] > 0:
689
690                                                               if self.C[c][209] > 0:
691
692                                                               if self.C[c][210] > 0:
693
694                                                               if self.C[c][211] > 0:
695
696                                                               if self.C[c][212] > 0:
697
698                                                               if self.C[c][213] > 0:
699
700                                                               if self.C[c][214] > 0:
701
702                                                               if self.C[c][215] > 0:
703
704                                                               if self.C[c][216] > 0:
705
706                                                               if self.C[c][217] > 0:
707
708                                                               if self.C[c][218] > 0:
709
710                                                               if self.C[c][219] > 0:
711
712                                                               if self.C[c][220] > 0:
713
714                                                               if self.C[c][221] > 0:
715
716                                                               if self.C[c][222] > 0:
717
718                                                               if self.C[c][223] > 0:
719
720                                                               if self.C[c][224] > 0:
721
722                                                               if self.C[c][225] > 0:
723
724                                                               if self.C[c][226] > 0:
725
726                                                               if self.C[c][227] > 0:
727
728                                                               if self.C[c][228] > 0:
729
730                                                               if self.C[c][229] > 0:
731
732                                                               if self.C[c][230] > 0:
733
734                                                               if self.C[c][231] > 0:
735
736                                                               if self.C[c][232] > 0:
737
738                                                               if self.C[c][233] > 0:
739
740                                                               if self.C[c][234] > 0:
741
742                                                               if self.C[c][235] > 0:
743
744                                                               if self.C[c][236] > 0:
745
746                                                               if self.C[c][237] > 0:
747
748                                                               if self.C[c][238] > 0:
749
750                                                               if self.C[c][239] > 0:
751
752                                                               if self.C[c][240] > 0:
753
754                                                               if self.C[c][241] > 0:
755
756                                                               if self.C[c][242] > 0:
757
758                                                               if self.C[c][243] > 0:
759
760                                                               if self.C[c][244] > 0:
761
762                                                               if self.C[c][245] > 0:
763
764                                                               if self.C[c][246] > 0:
765
766                                                               if self.C[c][247] > 0:
767
768                                                               if self.C[c][248] > 0:
769
770                                                               if self.C[c][249] > 0:
771
772                                                               if self.C[c][250] > 0:
773
774                                                               if self.C[c][251] > 0:
775
776                                                               if self.C[c][252] > 0:
777
778                                                               if self.C[c][253] > 0:
779
780                                                               if self.C[c][254] > 0:
781
782                                                               if self.C[c][255] > 0:
783
784                                                               if self.C[c][256] > 0:
785
786                                                               if self.C[c][257] > 0:
787
788                                                               if self.C[c][258] > 0:
789
790                                                               if self.C[c][259] > 0:
791
792                                                               if self.C[c][260] > 0:
793
794                                                               if self.C[c][261] > 0:
795
796                                                               if self.C[c][262] > 0:
797
798                                                               if self.C[c][263] > 0:
799
800                                                               if self.C[c][264] > 0:
801
802                                                               if self.C[c][265] > 0:
803
804                                                               if self.C[c][266] > 0:
805
806                                                               if self.C[c][267] > 0:
807
808                                                               if self.C[c][268] > 0:
809
810                                                               if self.C[c][269] > 0:
811
812                                                               if self.C[c][270] > 0:
813
814                                                               if self.C[c][271] > 0:
815
816                                                               if self.C[c][272] > 0:
817
818                                                               if self.C[c][273] > 0:
819
820                                                               if self.C[c][274] > 0:
821
822                                                               if self.C[c][275] > 0:
823
824                                                               if self.C[c][276] > 0:
825
826                                                               if self.C[c][277] > 0:
827
828                                                               if self.C[c][278] > 0:
829
830                                                               if self.C[c][279] > 0:
831
832                                                               if self.C[c][280] > 0:
833
834                                                               if self.C[c][281] > 0:
835
836                                                               if self.C[c][282] > 0:
837
838                                                               if self.C[c][283] > 0:
839
840                                                               if self.C[c][284] > 0:
841
842                                                               if self.C[c][285] > 0:
843
844                                                               if self.C[c][286] > 0:
845
846                                                               if self.C[c][287] > 0:
847
848                                                               if self.C[c][288] > 0:
849
850                                                               if self.C[c][289] > 0:
851
852                                                               if self.C[c][290] > 0:
853
854                                                               if self.C[c][291] > 0:
855
856                                                               if self.C[c][292] > 0:
857
858                                                               if self.C[c][293] > 0:
859
860                                                               if self.C[c][294] > 0:
861
862                                                               if self.C[c][295] > 0:
863
864                                                               if self.C[c][296] > 0:
865
866                                                               if self.C[c][297] > 0:
867
868                                                               if self.C[c][298] > 0:
869
870                                                               if self.C[c][299] > 0:
871
872                                                               if self.C[c][300] > 0:
873
874                                                               if self.C[c][301] > 0:
875
876                                                               if self.C[c][302] > 0:
877
878                                                               if self.C[c][303] > 0:
879
880                                                               if self.C[c][304] > 0:
881
882                                                               if self.C[c][305] > 0:
883
884                                                               if self.C[c][306] > 0:
885
886                                                               if self.C[c][307] > 0:
887
888                                                               if self.C[c][308] > 0:
889
890                                                               if self.C[c][309] > 0:
891
892                                                               if self.C[c][310] > 0:
893
894                                                               if self.C[c][311] > 0:
895
896                                                               if self.C[c][312] > 0:
897
898                                                               if self.C[c][313] > 0:
899
900                                                               if self.C[c][314] > 0:
901
902                                                               if self.C[c][315] > 0:
903
904                                                               if self.C[c][316] > 0:
905
906                                                               if self.C[c][317] > 0:
907
908                                                               if self.C[c][318] > 0:
909
910                                                               if self.C[c][319] > 0:
911
912                                                               if self.C[c][320] > 0:
913
914                                                               if self.C[c][321] > 0:
915
916                                                               if self.C[c][322] > 0:
917
918                                                               if self.C[c][323] > 0:
919
920                                                               if self.C[c][324] > 0:
921
922                                                               if self.C[c][325] > 0:
923
924                                                               if self.C[c][326] > 0:
925
926                                                               if self.C[c][327] > 0:
927
928                                                               if self.C[c][328] > 0:
929
930                                                               if self.C[c][329] > 0:
931
932                                                               if self.C[c][330] > 0:
933
934                                                               if self.C[c][331] > 0:
935
936                                                               if self.C[c][332] > 0:
937
938                                                               if self.C[c][333] > 0:
939
940                                                               if self.C[c][334] > 0:
941
942                                                               if self.C[c][335] > 0:
943
944                                                               if self.C[c][336] > 0:
945
946                                                               if self.C[c][337] > 0:
947
948                                                               if self.C[c][338] > 0:
949
950                                                               if self.C[c][339] > 0:
951
952                                                               if self.C[c][340] > 0:
953
954                                                               if self.C[c][341] > 0:
955
956                                                               if self.C[c][342] > 0:
957
958                                                               if self.C[c][343] > 0:
959
960                                                               if self.C[c][344] > 0:
961
962                                                               if self.C[c][345] > 0:
963
964                                                               if self.C[c][346] > 0:
965
966                                                               if self.C[c][347] > 0:
967
968                                                               if self.C[c][348] > 0:
969
970                                                               if self.C[c][349] > 0:
971
972                                                               if self.C[c][350] > 0:
973
974                                                               if self.C[c][351] > 0:
975
976                                                               if self.C[c][352] > 0:
977
978                                                               if self.C[c][353] > 0:
979
980                                                               if self.C[c][354] > 0:
981
982                                                               if self.C[c][355] > 0:
983
984                                                               if self.C[c][356] > 0:
985
986                                                               if self.C[c][357] > 0:
987
988                                                               if self.C[c][358] > 0:
989
990                                                               if self.C[c][359] > 0:
991
992                                                               if self.C[c][360] > 0:
993
994                                                               if self.C[c][361] > 0:
995
996                                                               if self.C[c][362] > 0:
997
998                                                               if self.C[c][363] > 0:
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
16
```

```

236             text = '' progress: {} / {} '.format(c + 1, self.c_len)
237             digits = len(text)
238             delete = '\r' * digits
239             print('{0}{1}'.format(delete, text), end='')
240             new = self.route_.locate(mu[c][:2])
241             mu[c][0], mu[c][1] = new[0], new[1]
242             print(' replacing old centers with new')
243             # equate current centroids to calculated
244             self.C = mu
245             # increment iteration counter
246             iteration += 1
247             print(' iteration end: {:.2f}k distance calculations'.format(self.
248                             icntr / 1000))
248             iter_time = time.time() - time_start
249             if iter_time > 86400:
250                 iter_time = '{:.4f} days'.format(iter_time / 86400)
251             elif iter_time > 3600:
252                 iter_time = '{:.4f} hours'.format(iter_time / 3600)
253             elif iter_time > 60:
254                 iter_time = '{:.4f} minutes'.format(iter_time / 60)
255             else:
256                 iter_time = '{:.4f} seconds'.format(iter_time)
257             print('* 17 + {}'.format(iter_time))
258             self.icntr = 0
259             # record results
260             self.cluster_centers_ = self.C
261             self.labels_ = self.L
262             self.population_ = self.P
263             if self.metric == 'route' or self.stations:
264                 self.route_.stop()
265
266 class KMeansClusteringMachine(ClusteringMachine):
267     """ A derived class from ClusteringMachine.
268
269     Performs clustering with using K-Means algorithm.
270
271     Parameters
272     -----
273     X : array, [n_points, n_dimensions]
274         Coordinates of points.
275     init : string, default 'random'
276         Specify the initializing type.
277     count : int, default 40
278         Set clusters count (in random initializing).
279     gridSize : array {size_x, size_y}, default [7, 7]
280         Set size of grid (in grid initializing).
281     filename : string path, default 'init.txt'
282         Specify the name of initialize file (in file initializing).
283     max_iter : int, default 100

```

```

284     Set maximum iteration number.
285
286     Attributes
287
288     initM : InitMachine object
289         Initializes centers of clusters.
290     bounds : array {bottom_border, left_border, top_border, right_border}
291         Bounds of initial centers generation.
292     """
293     initM = None
294     bounds = None
295
296     def __init__(self, X, init='random', count=40, grid_size=[7, 7],
297                  filename='init.txt', max_iter=100, log=True,
298                  thread_cound=4, start=False, stations=False):
299         global POOL
300         global THREADS
301         THREADS = thread_cound
302         POOL = ThreadPool(processes=THREADS)
303
304         self.X = X
305
306         self.initM = InitMachine()
307
308         self.bounds = self.getBounds(points=X)
309         if init == 'random':
310             self.initM.random(count=count, bounds=self.bounds)
311         elif init == 'grid':
312             self.initM.grid(gridSize=grid_size, bounds=self.bounds)
313         elif init == 'file':
314             self.initM.file(filename=filename)
315         else:
316             print('Unrecognized init type: {}'.format(init))
317         self.cluster_centers = self.initM.getCenters()
318         self.cluster_instance = KMeans(max_iter=max_iter, log=log, start=start,
319                                         stations=stations)
320
321     def getBounds(self, points):
322         """ Calculate bounds of initial centers generation.
323
324         Parameters
325
326         points : array [n_points, n_dimensions]
327             Coordinates of points
328
329         Returns
330
331         bounds : array {bottom_border, left_border, top_border, right_border}
332             Bounds of initial centers generation.

```

```

332 """
333 bounds = None
334 if points == None:
335     print('No source to get bounds')
336 else:
337     # if points are setted, choose four coordinates
338     # of different points as bounds:
339     # most bottom, most left, most top, and most right
340     b, l, t, r = [points[0]] * 4
341     for p in points[1:]:
342         if p[0] > t[0]:
343             t = p
344         if p[0] < b[0]:
345             b = p
346         if p[1] > r[1]:
347             r = p
348         if p[1] < l[1]:
349             l = p
350     bounds = [b[0], l[1], t[0], r[1]]
351 return bounds
352
353 def fit(self, metric='route'):
354     """ Perform clustering.
355
356     """
357     t_start = time.time()
358     # perform clustering
359     self.cluster_instance.fit(self.X, self.cluster_centers_, metric)
360     # calculate time
361     self.fit_time = time.time() - t_start - self.cluster_instance.sleeping
362     if self.fit_time > 86400:
363         self.fit_time = '{:.4f} days'.format(self.fit_time / 86400)
364     elif self.fit_time > 3600:
365         self.fit_time = '{:.4f} hours'.format(self.fit_time / 3600)
366     elif self.fit_time > 60:
367         self.fit_time = '{:.4f} minutes'.format(self.fit_time / 60)
368     else:
369         self.fit_time = '{:.4f} seconds'.format(self.fit_time)
370     # get points labels
371     self.labels = self.cluster_instance.labels_
372     # get clusters population
373     self.population = self.cluster_instance.population_
374     # get cluster centers
375     self.cluster_centers = self.cluster_instance.cluster_centers_
376     # get clusters number
377     self.n_cluster = len(np.unique(self.labels))

```

Модуль метрики Route:

```

1 import time
2 import json
3 import subprocess
4 import numpy as np
5
6 from sys import version_info as vinfo
7 PYTHON2 = True
8 if vinfo[0] > 2:
9     PYTHON2 = False
10
11 if PYTHON2:
12     from urllib import urlopen
13 else:
14     from urllib.request import urlopen
15
16 class OSRMError(Exception):
17     def __init__(self, value):
18         self.value = value
19     def __str__(self):
20         return repr(self.value)
21
22 class route():
23     """ A class for getting distance between points by finding a route.
24
25     Uses OSRM-Project API for routing. It sends requests to local OSRM machine
26     and gets distance from json-formatted response.
27
28     Attributes
29     -----
30     osrm: subprocess
31         Local OSRM machine.
32
33     Notes
34     -----
35     See https://github.com/Project-OSRM/osrm-backend/wiki for OSRM-Project API.
36
37     """
38     osrm = None
39     API = 5
40     sleep = 5
41
42     def start(self, loud=False):
43         """ Start local OSRM machine.
44
45         Needs time to start, contains a sleep statement.
46
47         You can have one running machine per instance of class.
48

```

```

49 """
50     if self.osrm is None:
51         if loud:
52             osrm = subprocess.Popen('osrm-routed ~/map/map.osrm', shell=True
53                                     )
54     else:
55         osrm = subprocess.Popen('osrm-routed ~/map/map.osrm > /dev/null',
56                               shell=True)
57     self.osrm = osrm
58     time.sleep(self.sleep)
59     a = [44.27, 48.41]
60     url = 'http://localhost:5000/nearest/v1/car/{} , {}' .format(*a[:-1])
61     try:
62         responce = urlopen(url)
63         if not PYTHON2:
64             responce = responce.readall().decode('utf-8')
65             data = json.loads(responce)
66         else:
67             data = json.load(responce)
68         try:
69             code = data['code']
70             self.API = 5
71         except KeyError:
72             self.API = 4
73     except:
74         self.API = 4
75
76     def viaroute(self, a, b):
77         if self.osrm is None:
78             raise OSRMError('OSRM not started!')
79
80         # API v4 uses [lat, lon] coordinates
81         # API v5 uses [lon, lat] coordinates
82
83         # send request to find route between points
84         if self.API == 4:
85             url = "http://localhost:5000/viaroute?loc={},{}&loc={},{}" \
86                   "&geometry=false&alt=false" .format(*np.append(a, b))
87         elif self.API == 5:
88             url = 'http://localhost:5000/route/v1/car/{} , {} ; {} , {} ?overview=false
89                           ' \
90                           '&alternatives=false&steps=false' .format(*np.append(a[:-1], b
91                                             [-1]))
92
93         # get responce
94         responce = urlopen(url)
95         # parse json
96         if not PYTHON2:
97             responce = responce.readall().decode('utf-8')

```

```

94     data = json.loads(responce)
95
96 else:
97     data = json.load(responce)
98
99 # if route isn't found
100 if self.API == 4:
101     if data['status'] is not 0:
102         raise OSRMError('Error routing: {}' .format(data['status_message']))
103     else:
104         return data['route_summary']['total_distance']
105 elif self.API == 5:
106     if data['code'] != 'Ok':
107         raise OSRMError('Error routing: {}' .format(data['message']))
108     else:
109         return data['routes'][0]['distance']
110
111 def locate(self, a):
112     if self.osrm is None:
113         raise OSRMError('OSRM not started!')
114
115     if self.API == 4:
116         loc = 'http://localhost:5000/locate?loc={},{}' .format(*a)
117     elif self.API == 5:
118         loc = 'http://localhost:5000/nearest/v1/car/{},{}' .format(*a[:-1])
119     l_resp = urlopen(loc)
120     if not PYTHON2:
121         l_resp = l_resp.readall().decode('utf-8')
122         l_data = json.loads(l_resp)
123     else:
124         l_data = json.load(l_resp)
125     # if can't locate
126     if self.API == 4:
127         if l_data['status'] != 0:
128             # stop osrm machine
129             self.stop()
130             # throw error
131             raise OSRMError('Error locating: {}' .format(l_data['status_message']))
132         else:
133             return l_data['mapped_coordinate']
134     elif self.API == 5:
135         if l_data['code'] != 'Ok':
136             # stop osrm machine
137             self.stop()
138             # throw error
139             raise OSRMError('Error locating: {}' .format(l_data['message']))
140         else:
141             return l_data['waypoints'][0]['location'][:-1]

```

```

141
142     def route_distance(self, a, b):
143         """ Get distance between points.
144
145         Get distance between points a and b by finding route on OSM map.
146
147         Parameters
148
149             a: array-like, shape=[2]
150                 Coordinates of first point.
151             b: array-like, shape=[2]
152                 Coordinates of second point.
153
154         Returns
155
156             dist: int
157                 Route distance between given points.
158
159         """
160
161         if self.osrm is None:
162             raise OSRMError('OSRM not started!')
163
164         dist = None
165         # if shape of given arrays isn't (0:2)
166         if a.shape[0] == 1:
167             a = np.array([a[0][0], a[0][1]])
168         if b.shape[0] == 1:
169             b = np.array([b[0][0], b[0][1]])
170
171         a, b = a[:2], b[:2]
172
173         c = list(map(lambda i: np.round(i, decimals=5), a))
174         d = list(map(lambda i: np.round(i, decimals=5), b))
175
176         if np.array_equal(c, d):
177             # if points are the same return zero
178             dist = 0.0
179         else:
180             # send request to find route between points
181             try:
182                 dist = self.viaroute(a, b)
183             except OSRMError:
184                 # locate first point
185                 c = self.locate(a)
186                 c = list(map(lambda i: np.round(i, decimals=5), c))
187                 # locate second point
188                 d = self.locate(b)
189                 d = list(map(lambda i: np.round(i, decimals=5), d))

```

```
190
191         if np.array_equal(c, d):
192             dist = 0.0
193         else:
194             dist = self.viaroute(c, d)
195     return dist
196
197     def stop(self):
198         """ Stop local OSRM machine.
199
200         """
201         if self.osrm is not None:
202             subprocess.Popen(['pkill', 'osrm-routed'])
203             self.osrm = None
```

Модуль вспомогательного класса:

```

1 import geojson as json
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5
6 class ClusteringMachine():
7     """ General class for clustering machine.
8
9     Parameters
10    -----
11     X : array, [n_points, n_dimensions]
12         Coordinates of points.
13
14     Attributes
15    -----
16     X : array, [n_points, n_dimensions]
17         Coordinates of points.
18     labels : array, [n_points]
19         Labels of each point.
20     cluster_centers : array, [n_clusters, n_dimensions]
21         Coordinates of cluster centers.
22     n_cluster : int
23         Number of clusters.
24     cluster_instance : class
25         Class used for clustering.
26     fit_time : float
27         Time of clustering.
28 """
29
30     X = None
31     labels = None
32     population = None
33     cluster_centers = None
34     n_cluster = None
35     cluster_instance = None
36     fit_time = 0
37
38     def __init__(self, X):
39         self.X = X
40
41     def fit(self):
42         """ Perform clustering.
43
44         """
45         self.cluster_instance.fit(self.X)
46
47     def plotClusters(self, plotCenters = True):
48         """ Plot the results of clustering.

```

```

49     Parameters
50
51     plotCenters : boolean, default True
52         If true, mark centers of clusters on plot.
53
54     Notes
55
56     Uses matplotlib.pyplot for plotting data.
57     """
58
59     # set colors of clusters
60     colors = 10 * [ 'r.', 'g.', 'm.', 'c.', 'k.', 'y.', 'b.' ]
61     # create a new figure
62     plt.figure()
63     # for each point in X array
64     for i in range(len(self.X)):
65         # plot it on figure with specified color
66         plt.plot(self.X[i][0], self.X[i][1],
67                  colors[self.labels[i]], markersize = 5)
68     # if plotCenters set to True, plot cluster centers as "X" marks
69     if plotCenters:
70         plt.scatter(self.cluster_centers[:, 0],
71                     self.cluster_centers[:, 1], marker = "x",
72                     s = 150, linewidths = 2.5, zorder = 10)
73     # showing result
74     plt.show()
75
76     def exportCentersToTextFile(self, filename):
77         """ Record cluster centers to text file.
78
79         Parameters
80
81             filename : string path
82                 Recording file name.
83
84         Notes
85
86             Uses JSON data format.
87             """
88             cc = list(map(lambda x, y: (np.append(x, y)).tolist(), self.
89                         cluster_centers, self.population))
90             try:
91                 path = os.path.dirname(filename)
92                 if path == '':
93                     pass
94                 else:
95                     if not (os.path.exists(path)):
96                         os.makedirs(os.path.dirname(filename))
97                     with open(filename, 'w') as file_:
98                         json.dump(cc, file_)

```

```

97     except IOError as e:
98         print('{}'.format(e))
99
100    def exportPointsToTextFile(self, filename):
101        """ Record points with labels to text file.
102
103        Parameters
104        -----
105        filename : string path
106            Recording file name.
107
108        Notes
109        -----
110        Uses JSON data format.
111        """
112        # create new array with one more dimension for points
113        X_ = np.empty((len(self.X), 3))
114        # for each point in X array
115        for i in range(len(X_)):
116            # set X_[i][0] and X_[i][1] to point coordinates
117            X_[i][0], X_[i][1] = self.X[i][0], self.X[i][1]
118            # set X_[i][2] to point label
119            X_[i][2] = self.labels[i]
120
121        try:
122            path = os.path.dirname(filename)
123            if path == '':
124                pass
125            else:
126                if not (os.path.exists(path)):
127                    os.makedirs(os.path.dirname(filename))
128                with open(filename, 'w') as file_:
129                    json.dump(X_.tolist(), file_)
130        except IOError as e:
131            print('{}'.format(e))

```

Модуль по загрузке данных из файлов:

```

1 import geojson as json
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 class DataCollector():
6     """ Collects points from various sources.
7
8     Attributes
9     -----
10    data : array, [n_points, n_dimensions]
11        Coordinates of points.
12    """
13
14    data = np.array([])
15
16    def uploadFromTextFile(self, filename, delimiter = ','):
17        """ Upload data from text file.
18
19        Parameters
20        -----
21        filename : string path
22            Name of data source file.
23        delimiter : string, default ','
24            Sets source file data delimiter.
25        """
26
27        try:
28            with open(filename) as file_:
29                # create an empty array for points
30                data_ = np.empty((0, 2), float)
31                # for each line in file read latitude and longitude of point
32                # and record them to data array
33                for line in file_:
34                    lat, lon = [float(n) for n in line.split(delimiter)[1:]]
35                    data_ = np.append(data_, [[lat, lon]], axis=0)
36                self.data = data_
37
38        # if error while reading file, print error message and clear data array
39        except IOError as e:
40            print('{}'.format(e))
41            self.data = np.array([])
42
43    def getData(self):
44        """ Get collected data.
45
46        Returns
47        -----
48        data : array, [n_points, n_dimensions]
49            Coordinates of points
50        """

```

```

49         return self.data
50
51     def plotData(self):
52         """ Plot collected data.
53
54     Notes
55     -----
56
57     Uses matplotlib.pyplot for showing data.
58     """
59
60     plt.figure()
61     plt.scatter(self.data[:, 0], self.data[:, 1])
62     plt.show()
63
64
65     def exportToTextFile(self, filename):
66         """ Record data to text file.
67
68     Parameters
69     -----
70
71     filename : string path
72         Recording file name.
73
74     Notes
75     -----
76
77     Uses JSON data format.
78     """
79
80     try:
81         with open(filename, 'w') as file_:
82             json.dump(self.data.tolist(), file_)
83     except IOError as e:
84         print('{}'.format(e))

```

Модуль инициализации начального положения центров кластеров:

```

1 import random
2 import geojson as json
3 import numpy as np
4 import os
5
6 class InitMachine():
7     """ Initializes centers of clusters.
8
9     Attributes
10    -----
11     centers : array, [n_clusters, n_params]
12         n_params = 3: lat, lon, id
13     """
14     centers = np.empty([0, 3], dtype='object')
15
16     def grid(self, gridSize, bounds):
17         """ Initialize centers by grid.
18
19         Parameters
20         -----
21         gridSize : array {size_x, size_y}
22             Set grid size.
23         bounds : array {bottom_border, left_border, top_border, right_border}
24             Specify borders.
25         """
26         delta_lt = (bounds[2] - bounds[0]) / gridSize[0]
27         delta_ln = (bounds[3] - bounds[1]) / gridSize[1]
28         curr_lt = bounds[0] + delta_lt / 2
29         curr_ln = bounds[1] + delta_ln / 2
30
31         self.centers = np.append(self.centers,
32             [[curr_lt, curr_ln, 0]], axis=0)
33         for i in range(gridSize[0] * gridSize[1]):
34             if curr_ln + delta_ln > bounds[3]:
35                 if curr_lt + delta_lt > bounds[2]:
36                     break
37             else:
38                 curr_lt += delta_lt
39                 curr_ln -= delta_ln * (gridSize[0] - 1)
40         else:
41             curr_ln += delta_ln
42         self.centers = np.append(self.centers,
43             [[curr_lt, curr_ln, i + 1]], axis=0)
44
45     def random(self, count, bounds):
46         """ Initialize centers by random.
47
48         Parameters

```

```

49
50     count : int
51         Set clusters count.
52     bounds : array {bottom_border, left_border, top_border, right_border}
53         Specify borders.
54 """
55     for i in range(count):
56         self.centers = np.append(self.centers,
57             [[random.uniform(bounds[0], bounds[2]),
58              random.uniform(bounds[1], bounds[3]), i]], axis=0)
59
60 def file(self, filename):
61     """ Initialize centers by random.
62
63     Parameters
64 """
65     filename : int
66         Specify source file name.
67 """
68     with open(filename, 'r') as file_:
69         centers = json.load(file_)
70     for i in centers:
71         centers[int(i[2])] = np.array(i[:3], dtype='object')
72         centers[int(i[2])][0] = float(centers[int(i[2])][0])
73         centers[int(i[2])][1] = float(centers[int(i[2])][1])
74         centers[int(i[2])][2] = int(centers[int(i[2])][2])
75     self.centers = centers
76
77 def getCenters(self):
78     """ Get centers of clusters.
79 """
80
81     return self.centers
82
83 def exportCentersToTextFile(self, filename):
84     """ Record cluster centers to text file.
85
86     Parameters
87 """
88     filename : string path
89         Recording file name.
90
91     Notes
92 """
93     Uses JSON data format.
94 """
95     try:
96         path = os.path.dirname(filename)
97         if path == '':

```

```
98     pass
99
100    else:
101        if not (os.path.exists(path)):
102            os.makedirs(os.path.dirname(filename))
103        with open(filename, 'w') as file_:
104            json.dump(self.centers.tolist(), file_)
105
except IOError as e:
    print('{}'.format(e))
```

Модуль для преобразования данных в формат для визуализации. Для расчета выпуклых оболочек используется алгоритм Джарвиса:

```

1 import json
2 import numpy as np
3 from argparse import ArgumentParser
4
5 def rotate(a, b, c):
6     return (b[1] - a[1]) * (c[0] - b[0]) - (b[0] - a[0]) * (c[1] - b[1])
7
8 def jenkins(a):
9     n = len(a)
10    p = list(range(n))
11    for i in range(1, n):
12        if a[p[i]][1] < a[p[0]][1]:
13            p[i], p[0] = p[0], p[i]
14    h = [p[0]]
15    del p[0]
16    p.append(h[0])
17    while True:
18        right = 0
19        for i in range(1, len(p)):
20            if rotate(a[h[-1]], a[p[right]], a[p[i]]) < 0:
21                right = i
22            if p[right] == h[0]:
23                break
24            else:
25                h.append(p[right])
26                del p[right]
27    return h
28
29 def hull(a, b):
30     h = []
31     for i in a:
32         h.append(b[i])
33     return h
34
35 def action(param_list):
36     last = param_list['last']
37     log = param_list['log']
38     metric = param_list['metric']
39     filecen = '{}_{}.js'.format(log, metric[0])
40     filepoi = '{}_{}.js'.format(log, metric[0])
41
42     centers = []
43     hulls = []
44
45     print('Processing {} {} {}...'.format(log, metric, last))
46
47     File = open(filecen, 'w')

```

```

48     File.write('{}_{}c = '.format(log[:3], metric[0]))
49     for i in range(1, last + 1):
50         filec = '{}_{}_centers_{}.js'.format(log, metric[0], i)
51         with open(filec, 'r') as file_:
52             center = json.load(file_)
53             # print('readed c#' + str(i))
54             centers.append(center)
55     json.dump(centers, File)
56     File.close()
57
58     File = open(filepoi, 'w')
59     File.write('{}_{}p = '.format(log[:3], metric[0]))
60     for i in range(1, last + 1):
61         filep = '{}_{}_points_{}.js'.format(log, metric[0], i)
62         with open(filep, 'r') as file_:
63             points = json.load(file_)
64             # print('readed p#' + str(i))
65             mx = int(max(map(lambda k: k[2], points))) + 1
66             ps = [np.empty([0, 3]) for each in range(0, mx)]
67             for j in points:
68                 b = int(j[2])
69                 ps[b] = np.append(ps[b], [j], axis=0)
70             this_hulls = []
71             for j in ps:
72                 try:
73                     # print('Calculating hull of {} points for cluster {}'.format(
74                     # len(j), j[0][2]))
75                     h = hull(jarvis(j), j.tolist())
76                     # print('Found {} vertices'.format(len(h)))
77                     this_hulls.append(h)
78                 except IndexError:
79                     pass
80             hulls.append(this_hulls)
81     json.dump(hulls, File)
82     File.close()
83
84 def actions(lists):
85     for params in lists:
86         action(params)
87
88 parser = ArgumentParser()
89 parser.add_argument('-m', '--metric', choices=['route', 'euclid', 'surface'],
90                     help='Used metric')
91 parser.add_argument('-l', '--last', type=int, help='Last iteration numer')
92 parser.add_argument('-f', '--folder', help='Folder with data')
93 parser.add_argument('-n', '--many', nargs='*', help='Multiple converts. Args
order: folder1 metric1 last1 folder2 metric2 ... ')
94 args = parser.parse_args()

```

```

94 if args.many:
95     folders = args.many[::3]
96     metrics = args.many[1::3]
97     lasts = args.many[2::3]
98     length = len(lasts)
99
100    acts = []
101    for i in range(length):
102        last = int(lasts[i])
103        fold = folders[i]
104        metr = metrics[i]
105        acts.append({'last': last, 'log': fold, 'metric': metr})
106
107    actions(acts)
108    if len(folders) > length:
109        print('[warn] Number of args was not n*3')
110 else:
111    last = args.last
112    metr = args.metric
113    fold = args.folder
114
115    action({'last': last, 'log': fold, 'metric': metr})

```

Приложение Б
Техническое задание