ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему «Разработка системы предсказания успешного завершения учебной дисциплины»

(промежуточный, этап 2)

Выполнил:
студент группы БПМИ185 _____ Наумова А. К.
Подпись
10.04.20

Принял:
руководитель проекта Андрей Андреевич Паринов
м.н.с. МНУЛ ИССА ФКН НИУ ВШЭ

Дата 10.04.2020 _____ _____
Оценка (по 10-тибалльной шкале) Подпись

Москва 2020

Содержание

1 Задачи КТ-1

Изучение и сравнение алгоритмов кластеризации. Описание алгоритмов BIRCH и OPTICS.

2 Алгоритм BIRCH

BIRCH — Balanced Iterative Reducing and Clustering using Hierarchies or BIRCH(Сбалансированн итеративное сокращение и кластеризация с использованием иерархий).

2.1 Применение алгоритма

- большой объем данных
- самостоятельный алгоритм кластеризацииhttps://www.overleaf.com/project/5e332fb75986
- вспомогательный алгоритм для сокращения входных данных

2.2 Шаги алгоритма

- Создание дерева, получение из дерева новых данных
- Применение другого алгоритма кластеризации для новых данных

2.3 Используемые термины

- CF: BIRCH сокращает входные данные, объединяя плотные участки точек в структуры для компактного хранения подкластеров—Clustering Feature (CF). CF = (N, $\overrightarrow{LS}$, SS)
  - N - число точек в CF
  - $\overrightarrow{LS}$—линейная сумма точек
  - $\overrightarrow{SS}$— квадратичная сумма точек

    Применение другого алгоритма кластеризации для новых данных

  CF может строиться и из одной точки.

  Объединение CF: $CF = CF_1 + CF_2 = (N_1 + N_2, \overrightarrow{LS_1} + \overrightarrow{LS_2}, SS_1 + SS_2)$
- Центроид: среднее значение координат точек, объединённых в CF (или просто центр точек в CF).
- Радиус: среднее расстояние от точек в CF до центроида.

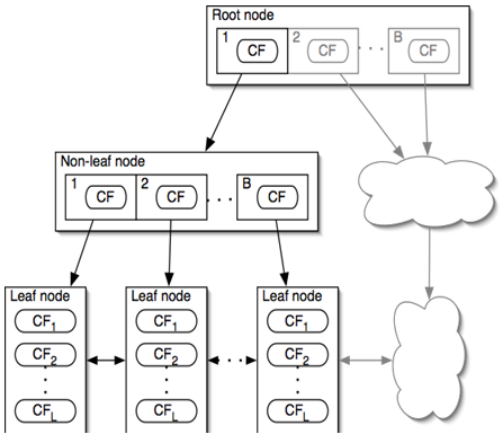
- Threshold: параметр, по которому объединяются CF в листьях- радиус от точки до центроида меньше threshold.
- Outliers: CF низкой плотности (с небольшим количеством точек) относительно остальных подкластеров.

2.4 Параметры

- Threshold
- Branching factor: максимальное число CF в вершине дерева (необходим для оптимизации)

2.5 Создание CF-tree

Описание дерева: Каждая вершина хранится массивом из CF, которые в них входят. Максимальное число CF в каждой вершине, не являющейся листом—B. Каждый CF является суммой CF в вершине, родителем которой является, и содержит ссылку на эту вершину. Максимальное число CF в листьях это L. Между листами есть ссылки на следующий и предыдущий листы.



Добавление в дерево нового CF: Добавление происходит рекурсивно, начиная с корня. Его шаги:

1. Нахождение подходящего листа: находим среди массива CF вершины, в которой стоим (изначально root), ближайший CF в евклидовой метрике. Ближайший CF это тот, для которого верно, что расстояние от его центроида до центроида добавляемого CF наименьшее.
2. Изменение листа: когда дошли до листа, выбираем ближайший CF.

- Если элемент подходит по threshold, объединим CF.
- Иначе добавим наш элемент в лист, если число элементов в листе позволяет добавить ещё один элемент (удовлетворяет ограничению L).
- Если нет, то разделим выбранный лист на два следующим образом:

  Найдём среди центроидов CF листа два наиболее удалённых друг от друга, они будут находиться в разных листьях. Для всех остальных будем проверять, к какому центроиду из двух они ближе, к такому листу и будем добавлять их.

3. Изменение пути до листа: для каждой вершины, которая не является листом, на пути к изменённому листу необходимо изменить путь до листа. Если произошло разделение листа на два, то родителя листа необходимо изменить (изменить CF, отвечающий за изменённый leaf, добавить новый CF):

   - Если не произошло переполнение в родителе листа, то изменяем его. Иначе поднимаемся выше и разделяем его родителя.
   - Если таким образом дошли до корня, то и корень делим на два и в дереве увеличивается число уровней.

2.6 Оптимизация памяти

Можно ввести ограничение на число вершин в дереве:

- Тогда если дерево будет переполняться, будем увеличивать threshold.
- После этого (при достаточном увеличении threshold) заново строим дерево до того момента, на котором остановились, добавляя уже выделенные CF.
- При этом или некоторые из CF (которые заново добавляем) сольются, или новый CF, который попытаемся добавить, войдёт по ограничению радиуса к кому-то из добавленных CF.

Таким образом, дерево может только уменьшиться.

Также, применяя предыдущую оптимизацию, при перестраивании дерева можно убирать из него outlier-ы, чтобы уменьшить объём данных, но сохранять их отдельно, выделяя для них определённый объем памяти.

Можно проверять, остаётся ли каждый outlier не включённым в дерево, или его можно добавить в какой-то лист:

- При следующем перестраивании дерева (при очередном увеличении threshold)
- Когда заканчивается память, выделенная на хранение outlier-ов

2.7 Получение из дерева новых данных

Полученные центроиды в CF-ах каждого листа и есть улучшенные данные (уменьшенные в объеме входные данные).
Эти же данные можно использовать, как уже готовый ответ, то есть точка, для которой i-й центроид ближайший, относится к i-му кластеру.

2.8 Применение другого алгоритма кластеризации для новых данных

1. Для центроидов всех CF в листах применить алгоритм кластеризации (например AgglomerativeClustering) для нахождения центров итоговых кластеров. Возвращает массив: i-е число— это номер кластера, к которому относится i-й центроид.
2. Находим для каждой точки ближайший центроид и относим точку к тому кластеру, к которому отнесён центроид. Это и будет ответ.

3 Алгоритм OPTICS

OPTICS — Ordering points to identify the clustering structure (Упорядочение точек для обнаружения кластерной структуры).

3.1 Приминение алгоритма

- большой объём данных
- большие кластеры
- кластеры с различным расстоянием от центра (разная плотность кластеров)

3.2 Параметры

- ξ: максимальный радиус кластера
- MinPts: минимальное число точек в кластере

3.3 Термины

- $N_\xi(p)$ множество точек, находящихся в ξ-окрестности точки p («соседи точки p»)
- core point — основная точка, у которой есть хотя бы MinPts соседей

3.4 Функции

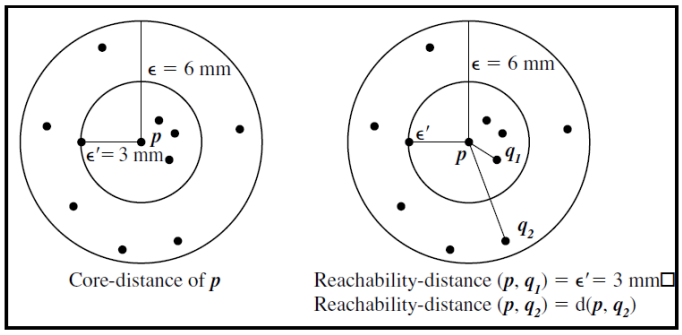
- core-dist — MinPts-е расстояние в порядке возрастания до $N_\xi(p)$

$$core-dist_{MinPts,\xi}(p) = \begin{cases} UNDEFINED & |N_\xi(p)| < MinPts \\ MinPts-thN_\xi(p) & |N_\xi(p)| \geq MinPts \end{cases}$$

- reachability-dist — максимум из расстояния от точки a до core-point p и core-dist(p)

$$\text{reachability-dist}_{MinPts,\xi}(a,p) = \begin{cases} UNDEFINED & |N_\xi(p)| < MinPts \\ max(core-dist(p), dist(p,o) & |N_\xi(p)| \geq MinPts \end{cases}$$

Как основное, так и достижимое расстояния не определены, если нет достаточно плотного кластера (применительно к ξ)
Ниже приведён пример,где $core-dist(p) = \xi'$



3.5 Шаги алгоритма

1. Вычисление core-dist для всех точек
2. Проход по всем точкам по одному разу, обновление reachability-dist у всех соседей выбранной точки

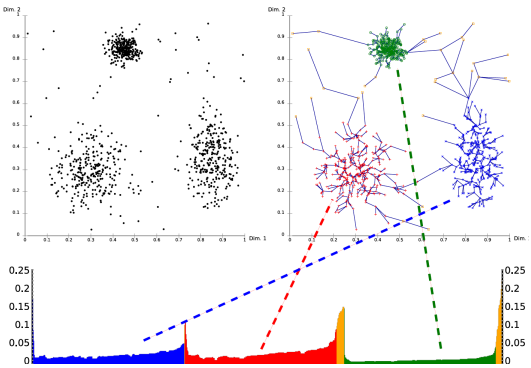
   Следующая точка—ближайшая по минимальному reachability-dist
3. Построение графика для извлечения кластеров

3.6 Построение графика

График достижимости—двумерный график:

- по оси x откладываются точки в порядке их обработки
- по оси y откладывается минимальное достижимое расстояние на момент добавления в список

Поскольку точки, принадлежащие кластеру, имеют небольшое достижимое расстояние до ближайшего соседа, кластеры выглядят как долины на графике достижимости. Чем глубже долина, тем плотнее кластер.



3.7 Извлечение кластеров

Извлечение кластеров из такого графика может быть осуществлено:

- вручную путём выбора интервалов по оси x после просмотра графика
- путём выбора порога по оси y (тогда результат подобен DBSCAN-кластеризации)
- с помощью различных алгоритмов, которые пытаются определить долины по крутизне графика, по изгибу или по локальным максимумам.

4 Задачи КТ-2

Реализация алгоритма кластеризации OPTICS, изучение и создание REST API сервера для доступа к данным.

5 Реализация клиент-серверной архитектуры

В проекте использовалась клиент-серверная архитектура JSON API, отвечающая требованиям RESTful.

5.1 REST

REST— согласованный набор архитектурных принципов для создания более масштабируемой и гибкой сети:

1. Сеть состоит из клиентов и серверов.
2. Существует понятие «без состояния». Оно не означает, что серверы и клиенты его не имеют, у них просто нет необходимости отслеживать состояние друг друга. Сервер также не ведёт учет прошлых запросов. Каждый запрос рассматривается как самостоятельный.
3. Унифицированный интерфейс.
    - Каждый ресурс должен быть уникально обозначен постоянным идентификатором
    - Клиент управляет ресурсами, направляя серверу представления, обычно в виде JSON-объекта, содержащего контент, который он хотел бы добавить, удалить или изменить. В REST у сервера полный контроль над ресурсами, и он отвечает за любые изменения
    - Каждое сообщение между клиентом и сервером самодостаточное, содержит всю информацию, которая необходима для понимания его получателем
    - Гипермедиа — это понятие для обозначения данных, которые содержат информацию о том, какие еще запросы клиент может сделать. В REST серверы должны посылать клиентам только гипермедиа.
4. Кэширование: ответы сервера должны помечаться как кэшируемые или некэшируемые. Когда эти данные нужны снова, кэширование может избавить от полного прохода данных по сети. Возможность кэшировать сущесвует благодаря самодостаточным сообщениям.

5. Уровневая система. Клиент может взаимодействовать не напрямую с сервером, а с произвольным количеством промежуточных узлов. Например, прокси действует как сервер для начального клиента, который посылает запрос, а затем как клиент, когда ретранслирует эту просьбу. Шлюз — это еще один дополнительный компонент, он переводит HTTP-запрос в другой протокол, распространяет этот запрос, а затем переводит полученный ответ обратно в HTTP.

5.2 JSON API

JSON API— это удобный способ реализации клиент-серверного взаимодействия. Он разработан для минимизации числа запросов и объема данных, необходимых для отправки между клиентом и сервером. Кроме того, на нём возможна удобная реализация взаимодействия клиента с базой данных сервера.
Flask— это библиотека на языке Python, предоставляющая "микро-фреймворк" для веб-разработки. Flask-REST-JSONAPI и Flask-SQLAlchemy— это дополнения Flask, которые использовались в реализации. Использование SQLAlchemy позволяет легко взаимодействовать с базой данных.
Центральное место в концепции RESTful имеет понятие ресурсов. Они представлены URI. Клиенты отправляют запросы к этим URI используя методы представленные протоколом HTTP. Метод GET позволяет получить информацию о ресурсе, POST создаёт новый ресурс, а DELETE удаляет его.
Реализация имеет два класса. Один из них отвечает за запросы на получение результатов всех проведённых экспериментов (GET-http запрос) и создание нового пустого эксперимента (POST-http запрос), другой начинает новый эксперимент с полученными параметрами (POST-http запрос) и возвращает результат введённого эксперимента (GET-http запрос). Каждый эксперимент проводится один раз.

5.3 Результаты

Вышеописанная архитектура была реализована для проведения экспериментов с алгоритмом OPTICS. Были созданы ссылки для получения результатов как эксперимента индивидуально, так и всех проведённых экспериментов вместе, и создания новых экспериментов. Полученные результаты были загружены в репозиторий (вместе с реализацией OPTICS):
https://github.com/AnastasiiaNaum/Study-Project-2020

6 Список информационных источников

BIRCH:

- Описание алгоритма

  https://towardsdatascience.com/machine-learning-birch-clustering-algorithm-clearly-explain

- Полная книга про алгоритм

  https://www2.cs.sfu.ca/CourseCentral/459/han/papers/zhang96.pdf

- Документация алгоритма

  https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html

OPTICS:

- Описание алгоритма

  https://towardsdatascience.com/clustering-using-optics-cac1d10ed7a7

- Описание алгоритма

  https://en.wikipedia.org/wiki/OPTICS_algorithm

- Сравнение различных алгоритмов кластеризации

  https://scikit-learn.org/stable/modules/clustering.html#birch

REST:

- Что такое REST

  https://habr.com/ru/company/dataart/blog/277419/

- Подходы к проектированию RESTful API

  https://habr.com/ru/company/dataart/blog/277419/

- Проектирование RESTful API с помощью Python и Flask

  https://habr.com/ru/post/246699/

- Flask Tutorial

  https://www.freecodecamp.org/news/build-a-simple-json-api-in-python/