Санкт-Петербургский государственный университет

Матвеева Татьяна Павловна

Выпускная квалификационная работа

Разработка быстрых алгоритмов расчета дельта-гиперболичности данных в задачах коллаборативной фильтрации на языке Python

Уровень образования: бакалавриат

Направление *02.03.03 "Математическое обеспечение и администрирование информационных систем"*

Основная образовательная программа *СВ.5162.2020 "Технологии программирования"*

Научный руководитель:
старший преподаватель кафедры СП, к.т.н. Ю. В. Литвинов

Консультант:
Ph.D., доцент департамента больших данных и информационного поиска факультета компьютерных наук ВШЭ Е. П. Фролов

Рецензент:
доцент, к.ф.-м.н., Сколковский институт науки и технологий А. М. Катруца

Санкт-Петербург
2024

Saint Petersburg State University

Tatiana Matveeva

Bachelor's Thesis

Development of fast algorithms for calculating delta-hyperbolicity of data in collaborative filtering problems in Python

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2020 "Programming Technologies"*

Scientific supervisor:
C.Sc., senior lecturer Y.V. Litvinov

Consultant:
Ph.D., docent of Big Data and Information Retrieval School, Higher School of Economics
E. P. Frolov

Reviewer:
C.Sc, docent, Skolkovo Institute of Science and Technology (Skoltech) A. M. Katrutsa

Saint Petersburg
2024

Оглавление

1. Введение

Успешная деятельность большинства крупных современных компаний, таких как Google и Яндекс, Amazon и Walmart, была бы невозможна без использования рекомендательных систем. Речь идет о системах, которые делают предположения относительно оценки каждым конкретным пользователем товаров и услуг, предлагаемых компанией, и рекомендуют наиболее подходящие.

Одним из основных методов, использующихся при создании рекомендательных систем, является коллаборативная фильтрация, что подтверждается большим количеством исследований на эту тему, например [25], [28] и [14]. Метод позволяет предсказать выбор конкретного пользователя на основании уже имеющихся данных о поведении других пользователей. Основная идея метода заключается в том, что пользователям, демонстрирующим схожее поведение, можно рекомендовать похожие товары.

Хорошим примером использования коллаборативой фильтрации являются такие сервисы как Netflix, Ozon, Okko. Кроме того, существуют наборы данных, такие как Amazon [26] и MovieLens [13], которые находятся в открытом доступе, предназначаясь для тренировки различных рекомендательных систем в исследовательских целях, и будут использованы в этой работе для валидации результатов.

В последние годы было предложено множество различных техник для применения коллаборативной фильтрации к реальным задачам. Некоторые обобщают подходы матричной факторизации [1], другие же используют нейросетевые архитектуры, такие как вариационные автокодировщики [8] или системы с обучением на последовательностях [24]. Из-за того что модели машинного обучения способны интерпретировать лишь математические объекты, а реальные данные не всегда являются таковыми (это могут быть строки, или id-номера, состоящие из цифр и букв), необходимо предварительно кодировать объекты и субъекты информационного процесса в числовые пространства. Правильный подход к проведению этого этапа может привести к значительному повыше-

нию эффективности работы рекомендательных систем, таких как [16] и [24]. Компании, заинтересованные в повышении качества рекомендаций, мотивируют дальнейшие исследования в этой области. Одним из перспективных направлений для изучения является внедрение гиперболической геометрии в архитектуру нейронных сетей.

Применение гиперболической геометрии — один из способов качественно отобразить реальные данные на числовое пространство для использования в машинном обучении [33]. За счет некоторых полезных свойств гиперболическая геометрия обеспечивает интерпретируемое вложение данных с иерархической структурой в пространство скрытых факторов, что было показано в работе [21].

Важной характеристикой гиперболического пространства является кривизна. В некоторых исследованиях кривизна фиксируется как константа — например, в работах [44], [11] авторы фиксируют кривизну как единицу. Однако при дальнейшем изучении было установлено, что модели с точно выбранной кривизной, соответствующей специфике конкретного набора данных, показывают лучшие результаты [17].

Вычислить кривизну можно с помощью такой характеристики данных как дельта-гиперболичность, которая была введена Михаилом Леонидовичем Громовым в работе [12]. Основная сложность состоит в том, что асимптотика базового подхода к вычислению этой величины составляет $O(n^4)$ относительно количества элементов в наборе данных, что делает работу в реальных случаях очень ресурсоемкой. Несмотря на это, алгоритм точно вычисляет δ-гиперболичность, поэтому может быть использован для подтверждения корректности других алгоритмов. Для приблизительной оценки этой величины, а также для исследования ее поведения при изменении размера выборки и размерности пространства, исследователями лаборатории «Вычислительного Интеллекта» [40] Сколтеха был предложен алгоритм, вычисляющий эту величину параллельно на небольших порциях данных. Экспериментальным путем было выяснено, что даже этот алгоритм не подходит для работы на наборах реальных данных также из-за высокой ресурсоемкости и значительной ошибки относительно базового алгоритма. Таким

образом, в рамках продолжения исследований по применению гиперболической геометрии в рекомендательных системах был сформулирован запрос на разработку алгоритма подсчета δ-гиперболичности, который более точно аппроксимирует базовый алгоритм и менее ресурсоемок.

2. Постановка задачи

Целью данной работы является разработка эффективного подхода к подсчету δ-гиперболичности. Для достижения этой цели были поставлены следующие задачи.

1. Провести обзор существующих подходов и эвристик для ускорения подсчета δ-гиперболичности. Выбрать наиболее эффективные алгоритмы.

2. Реализовать выбранные алгоритмы в форме библиотеки с Python-интерфейсом.

3. Провести сравнительный анализ точности реализованных алгоритмов с наивным алгоритмом расчета δ-гиперболичности на синтетических данных.

4. Провести сравнительный анализ точности реализованных алгоритмов с наивным алгоритмом расчета δ-гиперболичности, реализованном на GPU на датасетах MovieLens и Amazon.

5. Провести сравнительный анализ производительности реализованных алгоритмов с эвристикой, разработанной в лаборатории «Вычислительного интеллекта», на датасетах MovieLens и Amazon.

3. Обзор

В данном разделе будут рассмотрены основные понятия, связанные с определением гиперболического пространства, а также существующие алгоритмы и эвристики для подсчета δ-гиперболичности.

3.1. δ-гиперболичность в коллаборативной фильтрации

Гиперболическая геометрия была формализована Николаем Лобачевским в 1829 году. С точки зрения задачи коллаборативной фильтрации гиперболическая геометрия интересна своим репрезентативным представлением иерархических структур в скрытом пространстве, которые нередко возникают в наборах данных, использующихся в предметной области. Наборы данных, которые используются в этой работе представляют собой разреженную матрицу, где на пересечении i-го столбца и j-ой строки стоит некое число, отображающее взаимодействие соответствующего пользователя и товара. Иерархичность может возникать, например, из популярности, то есть ближе к корню (середине диска) находятся наиболее популярные товары, на периферии — более нишевые или новые. Эффект проиллюстрирован на Рис. 1, представленном в работе [18], где справа изображена древовидная структура, а слева диск Пуанкаре.

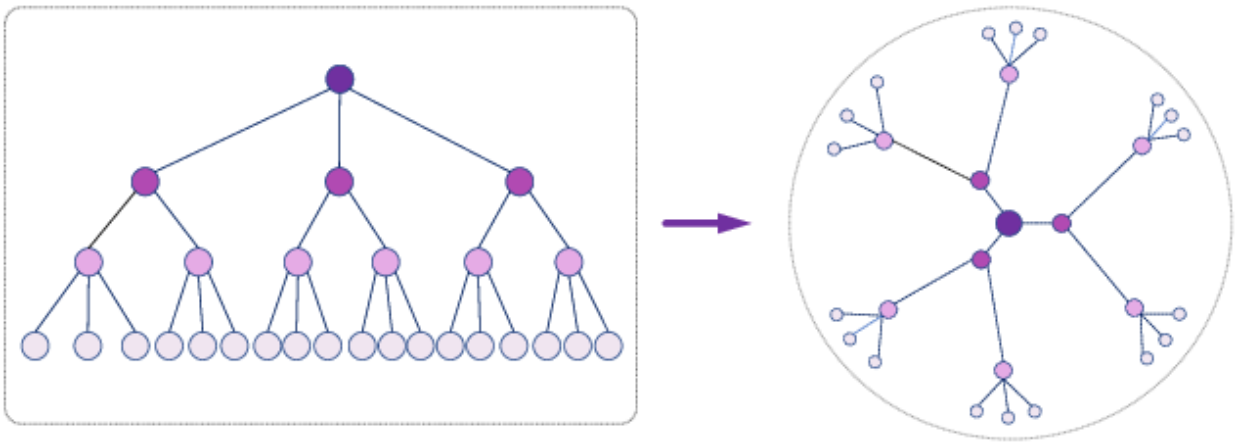

Рис. 1: Вложение иерархических структур в гиперболическое пространство

3.1.1. Формализация гиперболического пространства: диск Пуанкаре

Существует несколько формализаций гиперболического пространства, самые известные из них: псевдосфера, модель Бертрами-Кляйна, гиперболоид, полуплоскость Пуанкаре и шар Пуанкаре. Каждая модель имеет свои достоинства и недостатки, выбор в пользу того или иного формального обоснования зависит лишь от прикладной задачи, в которой предполагается использование гиперболического пространства.

В данной работе в качестве основной модели рассматривается диск Пуанкаре, так как именно эта интерпретация позволяет переопределить большинство алгебраических операций, необходимых для реализации моделей машинного обучения. Кроме того эффективность данной модели уже была продемонстрирована в применении к рекомендательным системам [15].

Модель шара Пуанкаре — это гиперболическое пространство размерности n, представленное открытым единичным шаром:

$$\mathcal{B}^n = \{x \in \mathbb{R}^n : \|x\| < 1\}$$

с постоянной кривизной $k = -1$.

Реальные данные не всегда могут хорошо описываться моделью с кривизной $k = -1$, поэтому в общем случае она может быть параметризована как

$$k^{-1} = -c^2. \tag{1}$$

Тогда модель диска Пуанкаре примет вид открытого шара с радиусом $\frac{1}{\sqrt{c}}$ и будет скорректирована следующим образом:

$$\mathcal{B}_c^n = \{x \in \mathbb{R}^n : c\|x\|^2 < 1, c \geq 0\}$$

Вычисление кривизны может показать, насколько искривлено скрытое пространство — то есть насколько успешным будет применение гиперболической геометрии для представления конкретных данных. Кро-

ме того, не последнюю роль играет точность, с которой вычислена кривизна [32].

3.1.2. δ-гиперболичность и кривизна пространства

Во многих работах, например [43], [19] и [30] для измерения кривизны используется понятие δ-гиперболичности — величины, введенной Михаилом Леонидовичем Громовым в работе [12]. Параметр c, использующийся в общем определении кривизны конкретного многообразия в (1), в этом случае определяется как отношение:

$$c(X) = \left(\frac{\delta_P}{\delta_X}\right)^2$$

где δ_P — δ-гиперболичность диска Пуанкаре единичного радиуса.

δ-гиперболичность для метрического пространства X, в свою очередь, определяется следующим образом.

Определение (δ-гиперболичность) Метрическое пространство X с метрикой d δ-**гиперболично**, если $\forall\ x, y, z, \omega \in X$ выполняется неравенство:

$$\delta \geq \min((x, y)_\omega, (y, z)_\omega) - (x, z)_\omega \tag{2}$$

где $(x, y)_\omega$ — **произведение Громова**, которое определяется как:

$$(x, y)_\omega = \frac{1}{2}(\mathrm{d}(\omega, x) + \mathrm{d}(\omega, y) - \mathrm{d}(x, y)) \tag{3}$$

δ-**гиперболичность** пространства X задается нижней границей возможных δ, при которых пространство (X, d) δ-гиперболично:

$$\delta_X = \min \delta \tag{4}$$

Неформально δ-гиперболичность можно описать как условие того, что любая точка на стороне любого треугольника, образованного геодезическими линиями, лежит в объединении δ-окрестностей других сторон.

Стоит отметить, что для диска Пуанкаре δ-гиперболичность может быть рассчитана аналитически и равна $\ln(1+\sqrt{2})$ [41]. Таким образом, для расчета кривизны необходимо вычислить только параметр δ_X, эффективный алгоритм для подсчёта которого разрабатывается в данной работе.

Помимо δ-гиперболичности существуют другие способы для определения и подсчета кривизны пространства. В работах последних лет для этой цели также используется отображение Вейнгартена [4], [9], однако работа с этим понятием предполагает выполнение некоторых условий, например ориентированность многообразия, гарантировать которые невозможно для произвольного набора данных.

3.1.3. δ-гиперболичность: эквивалентные определения

δ-гиперболичность является важной характеристикой для анализа графов, поэтому нередко в литературе можно встретить эквивалентное определение [5, 30].

Пусть $G = (V, E)$ — граф, S_i — суммы соответствующих расстояний для вершин $a, b, c, d \in V$

$$\begin{aligned} S_1 &= \mathrm{d}(a,b) + \mathrm{d}(c,d) \\ S_2 &= \mathrm{d}(a,c) + \mathrm{d}(b,d) \\ S_3 &= \mathrm{d}(a,d) + \mathrm{d}(b,c) \end{aligned} \tag{5}$$

Предположим, что $S_3 < S_2 < S_1$, тогда гиперболичность графа определяется выражением:

$$\delta = \frac{\min(S_1 - S_2)}{2} \ \forall \ a, b, c, d \in V \tag{6}$$

.

3.2. Алгоритмы подсчета δ-гиперболичности

В данном разделе описаны существующие алгоритмы и эвристики подсчета δ-гиперболичности. В первую очередь, они включают в себя

наивный алгоритм и эвристику для ускорения подсчета, предложенную в лаборатории «Вычислительного интеллекта». Несмотря на то, что данная эвристика ускоряет алгоритм, полученная аппроксимация не гарантирует высокую точность относительно базового наивного алгоритма. Поэтому в рамках исследования более оптимальных стратегий для подсчета δ-гиперболичности, приводится обзор других алгоритмов и эвристик, в том числе из смежных областей.

3.2.1. Наивный алгоритм

Использование для вычислений неравенств из определения (4) является самым прямолинейным подходом. Согласно этому подходу необходимо перебрать все возможные четвёрки элементов множества X, которые определяют нижнюю границу δ, для которой X δ-гиперболично. Псевдокод для алгоритма представлен в Листинге 1. Асимптотика данного алгоритма равна $O(n^4)$. Такой способ дает точный результат и подходит для работы с небольшими объемами данных, так как с ростом количества точек, принадлежащих множеству X, время расчета растет очень быстро.

Algorithm 1 Наивный алгоритм

Require: S – матрица расстояний между точками
Ensure: δ — дельта-гиперболичность

```
δ ← 0
for p ← 1 to N do
    for h ← 1 to N do
        for c ← 1 to N do
            for k ← 1 to N do
                S_1 = S[p, h] + S[k, c]
                S_2 = S[p, k] + S[h, c]
                S_3 = S[p, c] + S[k, h]
                M = 0.5 · (S_1 − max(S_2, S_3))
                δ ← max(δ, M)
            end for
        end for
    end for
end for
```

3.2.2. Алгоритм Condensed

Чтобы сократить время расчета δ-гиперболичности, в лаборатории «Вычислительного интеллекта» Сколтеха [40] был разработан алгоритм, вычисляющий δ-гиперболичность несколько раз для конкретного размера выборки и усредняющий полученные результаты. Методы, использующие такой подход для оценки величины называются методами Монте-Карло. Для каждого эксперимента одна из четырех необходимых для подсчета точек фиксируется, что позволяет получить асимптотику $O(mn^3)$, где m — количество сэмплирований выборки заданного размера. В оригинальной реализации данный параметр по умолчанию равен 25.

3.2.3. Альтернативные алгоритмы и эвристики

Алгоритмы оценки δ-гиперболичности можно разбить на два множества методов: те, что работают с представлением данных в виде набора точек в непрерывном n-мерном метрическом пространстве, и те, что работают с дискретным представлением, например, вершинами и ребрами графа. Первое представление данных активно используется на числовых векторах, представляющих данные в латентных числовых пространствах, например, для работы с изображениями [23, 22] или аудио [20]. Графовые представления, в свою очередь, активно применяются для анализа сложных сетей [42].

Основные теоретические подходы для эффективной оценки δ в дискретных метрических пространствах представлены в работе [10], которая содержит оптимизации для оригинального алгоритма, а также различные его аппроксимации. Так, в рамках неё предлагается эквивалентная формулировка, основанная на быстром $min - max$ перемножении матриц. Это позволяет сократить асимптотику алгоритма до $O(n^{3.69})$. Также в рамках данной работы аналитически показано, что выбор одной опорной точки даёт оценку дельты, не более чем в два раза отличающуюся от оригинальной при вычислительной сложности $O(n^{2.69})$. Помимо этого, авторы предлагают $(2log_2 n)$-аппроксимацию оценки δ-

гиперболичности, которую можно получить за $O(n^2)$, вложив метрическое пространство во взвешенное дерево определенного вида. Однако, стоит отметить, что использование данных подходов осложняется отсутствием поддержки эффективных алгоритмов в известных математических библиотеках, в отличие от матриц, определенных над полем вещественных чисел. Кроме того, данный прямолинейный подход не предполагает оценки δ-гиперболичности с учетом заданного вычислительного бюджета. В то же время, предложенные порядки аппроксимации алгоритма допускают значительную ошибку при оценке δ-гиперболичности, что может негативно сказаться на дальнейшем использовании полученного значения.

В случае подходов к оценке δ-гиперболичности для графов существует значительное количество вариаций алгоритмов и эвристик [31, 2, 7, 39]. Это связано с тем, что в случае графов некоторые вершины не соединены между собой, что позволяет хранить матрицы расстояний в разреженном формате и не учитывать некоторые пары в алгоритме, а также использовать информацию о встречающихся свойствах компонент в графе для ускорения вычислений. Наиболее практический и масштабируемый на большие данные алгоритм под названием CCL (Cohen, Coudert, Lancin) был предложен в работе [5]. Авторы описывают эвристику, в рамках которой пары точек сортируются в порядке убывания расстояния между точками. Также авторами предлагается критерий останова, который позволяет избежать полного перебора. В худшем случае данный подход дает асимптотику равную асимптотике наивного алгоритма, однако экспериментальное сравнение на различных наборах данных продемонстрировало его практическую эффективность. Отдельно стоит отметить, что в алгоритме предлагается возможность установки вычислительного бюджета фиксированием количества рассматриваемых пар удаленных друг от друга точек. Несмотря на то что алгоритм разработан для исследования кривизны графов, решение может быть применено к случаю непрерывного многообразия. Асимптотика такого решения — $O(|P|^2 + P\log(P))$, где $|P|$ — количество самых удаленных друг от друга пар точек. Псевдокод алгоритма представлен

в Листинге 2.

Algorithm 2 Алгоритм CCL

Require: S – матрица расстояний между точками
Ensure: δ — дельта-гиперболичность
$P = \{(x_1, y_1,), \ldots, (x_N, y_N)\} \leftarrow$ индексы пар точек, расстояние между которыми отсортировано по убыванию
$\delta \leftarrow 0$
for $i \leftarrow 1$ to N **do**
  if $S(x_i, y_i) \leq 2\delta$ **then**
    return δ
  end if
  for $j \leftarrow 1$ to $i - 1$ **do**
    $p = x_i$, $h = y_i$, $k = x_j$, $c = y_j$
    $S_1 = S[p, h] + S[k, c]$
    $S_2 = S[p, k] + S[h, c]$
    $S_3 = S[p, c] + S[k, h]$
    $M = 0.5 \cdot (S_1 - \max(S_2, S_3))$
    $\delta \leftarrow \max(\delta, M)$
  end for
end for

3.3. Выводы

В ходе данного обзора были рассмотрены основные понятия для определения δ-гиперболичности, а также некоторые идеи для ее подсчета. Значительная их часть представлена теоретическими результатами и аппроксимациями алгоритма. Однако аппроксимация предложенных алгоритмов грубее существующего алгоритма для подсчета параметра, также большинство работ не учитывают специфику обработки большого объема данных. Для дальнейшей реализации в рамках данной работы был выбран алгоритм CCL, который обладает рядом преимуществ — используемая в нем эвристика показывает высокую точность, также алгоритм предоставляет возможность задавать вычислительный бюджет, что важно для обработки больших объемов данных.

4. Реализация

В данном разделе приводится обзор и выбор конкретных технологий для реализации библиотеки с алгоритмом, детали реализации алгоритма CCL для подсчета δ-гиперболичности для различных вычислительных архитектур. Также, поскольку иногда требуется многократный подсчет δ-гиперболичности в конечном пайплайне для рекомендательных систем, в данной работе было предложено решение для параллельного подсчета параметра за счет многопоточности.

4.1. Выбор инструментария

Базовым шагом при выборе инструментария является формулирование требований к конечному решению. Совместно с сотрудниками лаборатории «Вычислительного интеллекта» были выделены следующие необходимые свойства:

- скорость алгоритма;
- масштабируемость на большие данные;
- точность вычислений;
- поддержка различных вычислительных архитектур (CPU и GPU);
- распространение языка разработки в ML-сообществе.

Было установлено, что реализация алгоритмов на языке Python может удовлетворить всем поставленным требованиям. Язык Python в стандартной реализации (интерпретатор CPython [3]) ввиду некоторых особенностей, таких как ограничение многопоточности и интерпретируемый код — нестандартный выбор для улучшения производительности вычислительных алгоритмов. Однако реализация кода на Python значительно сокращает время разработки новых решений, также ускоряет

проверку гипотез. Ниже рассмотрены популярные решения для оптимизации кода на выбранном языке разработки для различных вычислительных архитектур.

4.1.1. Инструменты для CPU

Так как проблему скорости языка пытались решить со дня его создания, в настоящее время существуют некоторые пути обхода ограничений Python, которые позволяют минимизировать как время разработки, так и время работы реализованных алгоритмов. В настоящее время наиболее популярны такие решения как PyPy [35], Numba [45] и Cython [6].

PyPy — альтернативный интерпретатор кода Python, написанный на RPython (restricted Python), специальном расширении языка Python, подразумевающем статическую типизацию. Основные различия с базовым интерпретатором CPython заключаются в следующих аспектах:

- Механизм сборки мусора — в отличие от CPython [6], PyPy [35] не использует счетчик ссылок, что снимает проблему блокировки многопоточности.

- Процесс интерпретации кода — судя по документации [34], PyPy использует tracing JIT-компиляцию, что означает, что самые часто использующиеся операции оптимизируются и переводятся сразу в машинный код. CPython в свою очередь расходует дополнительное время на трансляцию кода.

- Совместимость с библиотеками — основной проблемой при работе с PyPy может стать несовместимость с некоторыми библиотеками, зависящими от функций языка C. Однако, высокоэффективные пакеты для математических вычислений, такие как Scipy, Sklearn и Numpy, поддерживаются.

Numba является JIT-компилятором кода Python, разработанным Anaconda. Он ускоряет отдельные участки кода, отвечающие за математическое вычисления, переводя их в LLVM-код. Numba устанавлива-

ется как обычная библиотека, а ускорение функций реализуется через применение специальных декораторов к обычным функциям, реализованным на Python с некоторыми ограничениями.

Cython — расширение Python, позволяющее напрямую вызывать функции C и C++. Проект нацелен на упрощение интеграции между кодом на Python и C, и в некоторых случаях обеспечивает многократное ускорение вычислений. Большим минусом этого решения является сильное отличие синтаксиса, что дает относительно высокий порог входа, увеличение времени на разработку.

Стоит заметить, что разработка реализации для центрального процессора важна, поскольку обычно объем памяти на CPU значительно больше, чем доступное количество памяти на GPU. Таким образом, в некоторых случаях более оптимальным решением может оказаться использование центрального процессора из-за возможности разместить большее количество данных и отсутствия дополнительных расходов на перенос данных с CPU на GPU и обратно.

4.1.2. Инструменты для GPU

В настоящее время наиболее популярным и эффективным способом ускорения кода является использование вычислений на графическом процессоре. При корректном использовании векторизации типа SIMD[1] и расчете оптимальных значений для числа блоков и потоков код может быть ускорен в десятки раз. Сегодня для языка Python существуют следующие инструменты, позволяющие перенести вычисления с CPU на GPU.

CuPy [29] — это адаптация функций библиотеки NumPy для вычислений на графическом процессоре. В своей основе библиотека CuPy поддерживает работу с многомерными массивами, хранящимися в памяти GPU, и содержит оптимизации популярных векторизованных функций, реализованных с помощью CUDA. CuPy является хорошим решением для оптимизации функций, в основном полагающихся на векторизован-

[1]Single Insturction Multiple Data — технология векторизации, при которой одна инструкция применяется к множеству данных

ные функции Numpy. Так как в рассматриваемых алгоритмах ключевую роль играет поиск максимума и последовательная обработка пар, функциональность Numpy не может быть применена явным образом.

Numba. Написание ядер (функций для исполнения на графическом процессоре) в библиотеке Numba производится за счет применения соответствующих декораторов и преобразования кода таким образом, чтобы ядро содержало последовательность команд, которые будут исполняться каждым потоком независимо. Предоставленная функциональность позволяет оптимизировать код с минимальными изменениями в синтаксисе и является лучшим решением для реализации алгоритмов, для логики которых не хватает функций, предложенных в CuPy. Кроме того, использование оптимизаций Numba для CPU и GPU помогает выдержать код в едином стиле.

4.2. Реализация алгоритма CCL для различных архитектур

В данном разделе рассматриваются детали реализации алгоритма CCL для CPU и GPU. Поскольку начиная с некоторого объема данных, для которых нужно рассчитать δ-гиперболичность, невозможна загрузка всех данных, а именно матрицы расстояний, в память GPU в рамках работы предложены две реализации алгоритма на GPU: для малых объемов данных (когда матрица расстояний помещается на GPU) и для больших объемов данных (когда данные специальным образом обрабатываются по частям).

Расчет δ-гиперболичности с использованием алгоритма CCL на центральном и графическом процессоре включает в себя следующие основные шаги:

- расчет матрицы расстояний;
- расчет массива отсортированных по расстоянию пар;
- основной алгоритм расчета, приведенный в Листинге 2.

Первые два шага выполняются на центральном процессоре. Выбор наиболее оптимального подхода для расчета матрицы расстояний представлен в разделе 4.3. Для последнего шага в рамках данной работы реализованы целевые модификации алгоритма.

4.2.1. Реализации на CPU и GPU

Для реализации ускоренных версий алгоритма на CPU и GPU был реализован базовый алгоритм на Python согласно Листингу 2.

Если выбрана ускоренная модификация для CPU, основной алгоритм выполняется на центральном процессоре с использованием JIT-компиляции, реализованной с помощью декоратора `njit` и при помощи итератора `prange` библиотеки Numba [45]. Если же требуется рассчитать параметр на графическом процессоре, исполняется ядро, написанное для архитектуры CUDA[2], функциональность для которой предоставлена в библиотеке Numba [45] через соответствующий декоратор.

Также были использованы флаги JIT-компилятора `parallel`, `fastmath` и `nopython`, которые позволяют оптимизировать циклы, исполняя их параллельно, математические вычисления, а также значительно улучшить производительность за счет трансляции в машинный код.

4.2.2. Реализация на GPU с обработкой данных по частям

Одна из сложностей при использовании графического процессора для вычислений — ограничение по памяти, с которым приходится сталкиваться при обработке объемных наборов данных. В частности, в базовой версии алгоритма требуется доступ ко всей матрице расстояний, которая, начиная с некоторого объема, может не помещаться в память GPU.

Поэтому в рамках данной работы был предложен подход, состоящий в том, чтобы разделить обрабатываемые данные на части, соответствующие максимально возможному объему, и применить реализацию на

[2]Compute Unified Device Architecture — программно-аппаратная архитектура параллельных вычислений

GPU отдельно к каждой из них. Такой подход требует предварительной обработки на центральном процессоре следующим образом.

В алгоритме CCL сначала формируется массив наиболее удаленных друг от друга пар. Затем, на каждой итерации для обработки новой пары рассматривается ее комбинация с каждой парой, предстоящей ей в отсортированном по расстоянию массиве, и для каждой четверки индексов рассчитывается параметр δ-гиперболичности. Таким образом, рассматривается декартово произведение всех возможных пар в специальном порядке. Так как алгоритм предполагает максимальную нагрузку GPU, построение батча там же невозможно, поскольку в процессе обработки также предполагается хранение массива пар. В рамках данной работы предлагается делить декартово произведение на части (батчи) на CPU, затем подсчитывать δ-гиперболичность на GPU для каждого батча последовательно, после чего считать максимум всех батчей на CPU.

Структура массива, обрабатываемого алгоритмом, в случае такого подхода должна быть изменена так, чтобы все шесть расстояний, необходимых для одной итерации алгоритма, находились на соседних позициях в массиве. Такое преобразование данных требует отображения одномерных индексов массива пар в двумерные, как показано на Рис. 2.

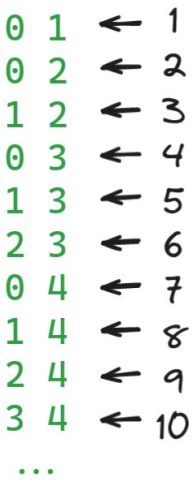

Рис. 2: Отображение в индексы декартова произведения

Для того чтобы оценить, какие пары входят в конкретный батч,

была использована формула арифметической прогрессии:

$$S_n = \frac{a_1 + a_n}{2} \cdot n,$$

которая в этом случае примет вид

$$S_n = \frac{1 + n}{2} \cdot n.$$

Подставим одномерный индекс indx вместо S_n, и решим квадратное уравнение относительно n:

$$n^2 + n - 2 \cdot \text{indx} = 0.$$

Из двух полученных корней необходим положительный

$$n = \frac{-1 + \sqrt{(1 + 8 \cdot \text{indx})}}{2}$$

Так как indx представляет собой число между S_{n-1} и S_n, полученное значение нужно округлить в большую сторону с помощью функции ceil.

Индекс i должен перебирать все числа от 0 до n - 1, поэтому представляется как

$$i = n - 1 - (S_n - \text{indx})$$

то есть равен 0 , когда $S_n = \text{indx} + n - 1$, равен $n - 1$, когда $S_n = \text{indx}$ и принимает все значения между ними. Итоговая формула выглядит как:

$$\begin{cases} i = n - (S_n - \text{indx}) - 1, \\ j = \text{ceil}(\frac{-1+\sqrt{(1+8 \cdot \text{indx})}}{2}) \end{cases}$$

Схематично разработанный алгоритм представлен на Рис. 3. Для оптимальной загрузки графического процессора был реализован механизм, позволяющий вычислять максимальный размер батча, который возможно поместить в память GPU.

Рис. 3: Схема работы алгоритма с обработкой данных на GPU по частям

4.3. Подсчет матрицы расстояний

Важным этапом при подсчете δ-гиперболичности является расчет матрицы расстояний между точками, который по производительности напрямую коррелирует с количеством точек и может быть ресурсоемок. Так как задача крайне распространена, свои варианты реализации предлагают многие математические библиотеки, такие как scipy [37], sklearn [38] и cupy [29]. Рассмотрим каждое из указанных решений подробнее.

1. `pdist` — реализация функции расстояний в библиотеке scipy. Код использует двойной цикл, но дополнительное ускорение происходит за счет реализации на языке C. Дополнительным достоинством является особый способ хранения матрицы в памяти: в отличие от аналогичной функции (`cdist`), также представленной в библиотеке, и подсчитывающей матрицу расстояний в стандартном виде, функция `pdist` возвращает одномерный массив, который может быть отображен в верхнюю треугольную часть матрицы расстояний.

2. `pairwise_distances` — функция, реализованная в математическом пакете sklearn. Имеет многопоточную реализацию: сначала высчитывается число потоков, которое будет максимально эффективным, затем вся матрица разбивается на n блоков, которые об-

рабатываются параллельно с использованием векторизации. Если эффективнее всего будет работать с матрицей целиком, то используется функция `cdist` пакета scipy.

3. `distance_matrix` — пакет CuPy, предоставляющий функции для работы на платформах NVIDIA и AMD, тоже имеет несколько реализаций функций для подсчета попарных расстояний. Одной из них является `distance_matrix`. Стоит отметить, что раздел библиотеки cupyx.scipy.spatial был добавлен недавно и, возможно, еще не оптимизирован. В коде используется `cdist` пакета scipy.

На рисунке 4 представлено сравнение скорости работы каждого из подходов. Сравнение проводилось на датасете MovieLens, на вычислительном устройстве с характеристиками, представленными в Таблице 1. Из-за очевидного преимущества по времени работы в рамках данной работы для вычислений матрицы расстояний используется функция `pairwise_distances`.

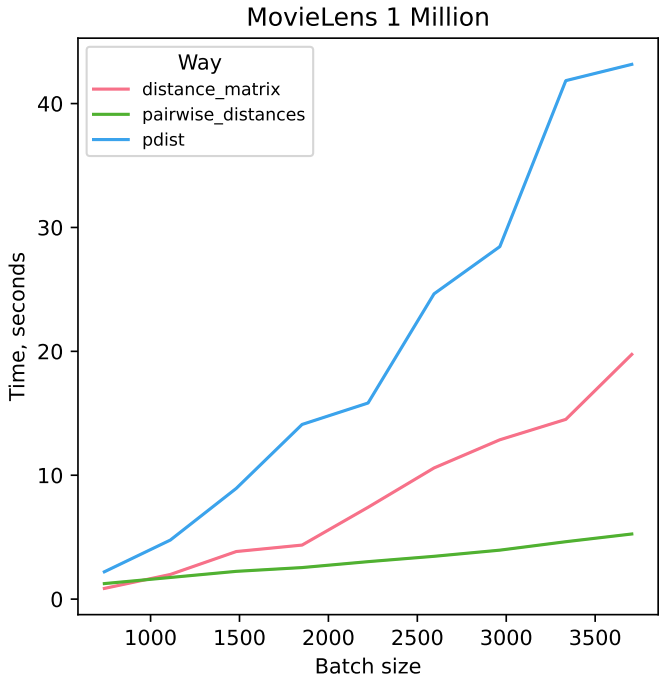

Рис. 4: Сравнение функций подсчета попарных расстояний

4.4. Параллельный подсчет δ-гиперболичности для нескольких выборок

Для использования δ-гиперболичности в рекомендательных системах иногда требуется процесс многократного подсчета этого параметра для случайно выбранных поднаборов точек. Чтобы ускорить данный процесс, в работе предлагается решение с использованием многопоточности. Для этого были использованы инструменты ThreadPool [27] и библиотека Numba. Декораторы библиотеки были использованы преимущественно для реализации параллелизма внутри алгоритма, тогда как методы ThreadPool помогли оптимизировать процесс многократного подсчета дельты для случайно выбранных поднаборов точек.

Для каждого из трех реализованных алгоритмов (CCL CPU, CCL GPU, CCL GPU-batch) была реализована своя оптимизированная стратегия.

Для алгоритма CCL CPU необходимо лишь посчитать максимальное число потоков, которое может быть обработано на центральном процессоре. Расчет производится исходя из доступного объема памяти и объектов, которые необходимо туда поместить.

Необходимость в следующей стратегии (для алгоритма CCL на GPU) возникла из-за различного вычислительного бюджета для CPU и GPU. Таким образом, первым этапом строятся массивы пар и матрицы расстояний, в том количестве, которое может быть размещено в памяти центрального процессора. Далее происходит подсчет максимально возможного количества потоков для GPU и полученные раннее на CPU объекты (матрицы расстояний и пары) обрабатываются в цикле, каждая итерация которого рассчитывает параметр в доступных потоках.

Стратегия для CCL GPU-batch обрабатывает случай, когда даже один набор необходимых объектов (матрица и массив пар) не может быть обработан на графическом процессоре. Матрицы и пары при этом варианте расчета также рассчитываются на центральном процессоре, но на GPU использование параллелизма уже невозможно, так как обработка одного объекта занимает все ресурсы процессора.

5. Эксперименты

В данном разделе представлены эксперименты, проведенные для оценки точности и производительности реализованных алгоритмов, а также для оценки наиболее подходящих гиперпараметров алгоритмов.

5.1. Тестовый стенд

Вычислительное устройство. Все эксперименты проводились на удаленном сервере с характеристиками, представленными в Таблице 1. При запуске алгоритмов количество ядер CPU ограничивалось 25.

Архитектура CPU	x86_64
Количество ядер CPU	80
Модель CPU	Intel®Xeon® CPU E5-2698 v4 @ 2.20GHz
Модель GPU	Tesla V100-SXM2-16GB

Таблица 1: Характеристики удаленного сервера

Наборы данных. Для экспериментов использовались наборы данных, предложенные в датасетах Amazon [26] и MovieLens [13]. Детальные характеристики конкретных наборов данных указаны в Таблице 2.

Набор данных	Users	Items	Interactions	Density
Software	1826	802	11884	0.00811
Luxury_Beauty	3819	1581	28074	0.00465
Movielens 1 million	6040	3706	1000209	0.04468

Таблица 2: Характеристики наборов данных

Плотность набора данных (density) оценивалась как отношение ненулевых элементов матрицы взаимодействий ко всем. Она характеризует количество полезной информации, содержащейся в датасете. Рассматриваемые наборы данных покрывают различные области применения рекомендательных систем, а также имеют отличающиеся характеристики по таким параметрам как количество товаров и плотность набора

данных. Для предобработки данных и вложении их в числовые пространства использовался метод `randomized SVD`, предложенный в статье [36]. Основная идея подхода в том, что умножив исходную матрицу X на случайную матрицу малого ранга P, с большой вероятностью мы сохраним основные векторы, которые хотели получить в малоранговом представлении. Далее, несколько раз применяя QR разложение и используя свойства случайных матриц, а также структуру матриц, участвующих в разложении, мы можем вычислить для исходной матрицы X размерности (m, n) представление вида

$$X = U_r \Sigma_r V_r^T,$$

где r — необходимый малый ранг.

Рассматриваемые алгоритмы. В экспериментальных сравнениях использовались следующие алгоритмы: базовый алгоритм: оригинальная реализация на CPU (`naive CPU`) и ускоренная реализация на GPU с помощью библиотеки Cuda, реализованной как часть Numba (`naive GPU`), `condensed` — алгоритм, предложенный в лаборатории «Вычислительного интеллекта», а также три реализованные в данной работе модификации алгоритма CCL — реализация на CPU (`CCL CPU`), реализация на GPU для случая, когда матрица расстояний помещается в память GPU (`CCL GPU`), реализация на GPU с обработкой данных по частям (`CCL GPU—batch`).

5.2. Исследовательские вопросы

Целью эксперимента является установление корректности реализованных алгоритмов (`CCL CPU`, `CCL GPU`, `CCL GPU—batch`), сравнение времени их работы относительно существующих алгоритмов, а также поиск оптимальных гиперпараметров. Для достижения поставленной цели были сформулированы следующие исследовательские вопросы:

- *Каковы оптимальные гиперпараметры алгоритма CCL?* Вопрос нацелен на сравнение качества оценки δ-гиперболичности алгорит-

мом CCL для различных значений основного гиперпараметра алгоритма (количества рассматриваемых пар) и оценки производительности при этих значениях.

- *Какова относительная ошибка между реализованными алгоритмами и наивным подходом?* Вопрос нацелен на сравнение качества оценки дельты каждым из решений.

- *Какая реализация алгоритма наиболее эффективна?* Вопрос нацелен на сравнительный анализ времени работы алгоритмов.

5.2.1. Эквивалентность реализаций CCL

Эквивалентность различных реализованных модификаций алгоритма CCL, а именно `CCL CPU`, `CCL GPU`, `CCL GPU—batch`, была проверена на синтетических данных для наборов из 150, 500 и 1000 точек в пространствах размерности 100, 200, 300, 400 и 500. Ошибка между алгоритмами при всех параметрах не превышает машинной точности.

5.2.2. Сравнение производительности реализаций CCL

Для сравнения производительности был использован набор данных Movielens 1 million, на котором алгоритмы `CCL GPU`, `CCL CPU`, `CCL GPU—batch` запускались 25 раз для каждого размера выборки с 10 до 100% с шагом 10%. На Рис. 5 представлена производительность различных реализаций алгоритма CCL. Как видно, реализация на GPU значительно превосходит остальные алгоритмы по эффективности. Далее для экспериментов по умолчанию будет использоваться модификация `CCL GPU` как наиболее эффективная. Несмотря на то, что при большом количестве попыток `CCL GPU—batch` уступает по производительности версии `CCL CPU`, при расчетах используется именно она, так как на 100% батче дисперсия оценки дельты алгоритмом равна нулю, следовательно, число испытаний может быть сокращено до 1. То, что `CCL GPU—batch` более эффективен, чем CPU версия при 1 испытании показано на Рис. 6

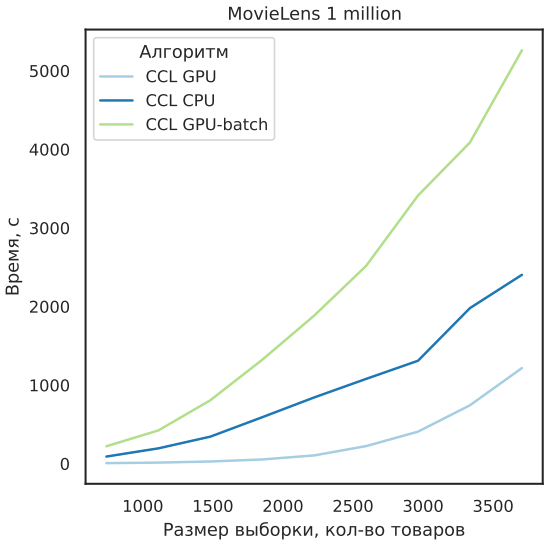

Рис. 5: CCL при 25 испытаниях

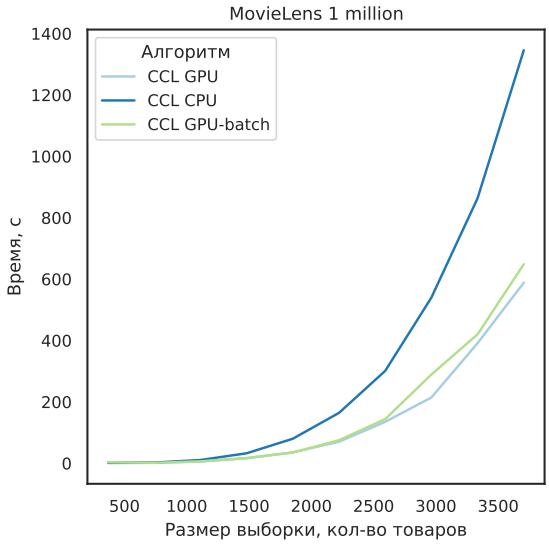

Рис. 6: CCL при 1 испытании

5.2.3. Оптимальные гиперпараметры алгоритма CCL

Алгоритм CCL позволяет рассчитывать δ-гиперболичность, не прибегая к полному перебору, а используя для подсчета лишь ограниченное гиперпараметром l количество самых далеких пар (в процентах). В данном эксперименте предлагается найти количество пар l, оптимальное с точки зрения точности и производительности. Для этого было проанализировано поведение алгоритмов на различных размерах выборки, составляющих от 10 до 100% от каждого датасета, для каждого размера выборки набор точек обрабатывался 25 раз. В качестве возможных значений гиперпараметра l были рассмотрены 1, 2 и 5% от общего количества. На Рис. 7 приведены значения относительной ошибки по размеру выборки, а также доверительные интервалы с 95% уровнем доверия. На Рис. 8 продемонстрирована производительность каждого подхода. Как можно видеть, реализованный алгоритм CCL быстрее существующих реализаций, а также уже начиная с параметра $l = 0.01$ он дает результаты лучше ранее предложенной эвристики в алгоритме Condensed. На датасете `MovieLens 1 million` можно наблюдать уменьшение доверительного интервала с увеличением размера выборки в отличие от алгоритма Condensed. Это обусловлено более детерминированным подходом к подсчету величины. Таким образом можно сделать вывод, что

реализованный алгоритм дает более надежную оценку.

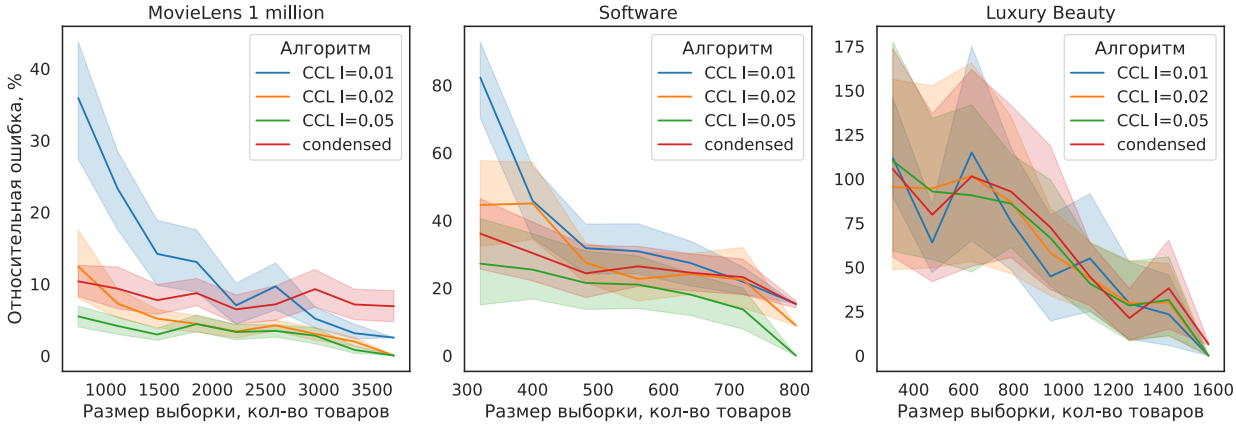

Рис. 7: Относительная ошибка алгоритмов на различных датасетах

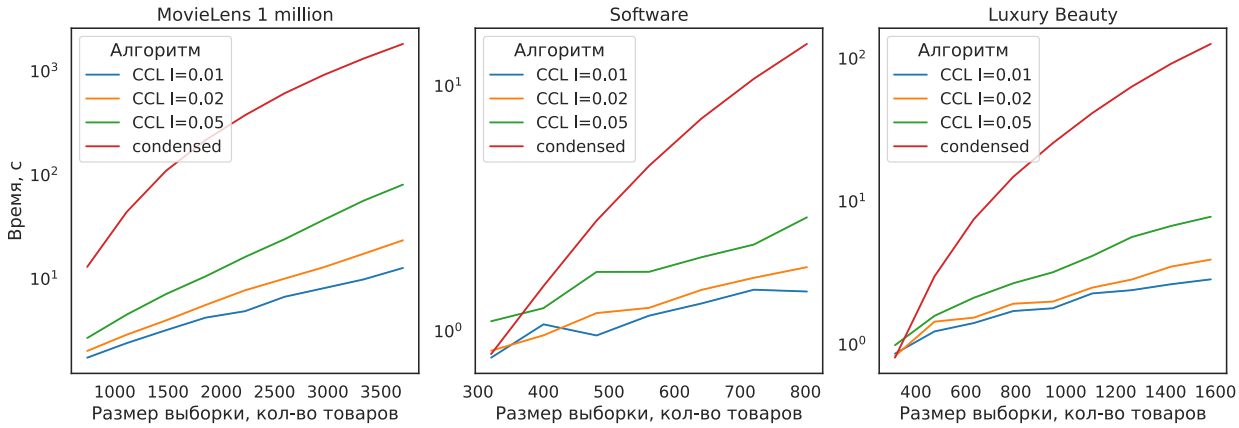

Рис. 8: Время работы алгоритмов на различных датасетах

5.3. Выводы

Подведем итоги экспериментального сравнения, ответив на поставленные вопросы.

Какое минимальное количество пар можно взять для корректной работы алгоритма CCL? Исходя из полученных результатов, можно наблюдать, что алгоритму CCL достаточно от 2 до 5 процентов пар для корректной оценки δ-гиперболичности при рассмотрении всего датасет.

Какова относительная ошибка между реализованными алгоритмами и наивным подходом? Посмотрев на визуализацию полученных результатов, представленную на Рис. 7 и Рис. 8, можно убедиться, что для верно подобранного гиперпараметра l ошибка алгоритма CCL при расчете дельты для всего датасета близка к машинному нулю, тогда как ошибка алгоритма Condensed имеет значительный разброс и варьируется от 7 до 20 процентов на различных наборах данных.

Какая реализация алгоритма наиболее эффективна? Время рассчета δ-гиперболичности сильно зависит от выбранной реализации, каждая из которых может быть полезна в определенной ситуации. Наиболее быстрым решением будет использование алгоритма CCL на GPU, однако, если набор данных объемный, расчет производится с помощью версии `CCL GPU—batch`. `CCL GPU—batch` показывает лучшую производительность, чем аналог CCL на CPU при небольшом количестве сэмплов.

6. Заключение

В ходе выполнения дипломной работы получены следующие результаты.

- Проведен обзор существующих подходов для ускорения подсчета δ-гиперболичности (наивный подход, алгоритм Condensed), набора модификаций алгоритма CCL (базовый алгоритм, алгоритм с использованием ограниченного вычислительного бюджета). С учетом специфики данных для реализации на различных целевых архитектурах (CPU, GPU) были выбраны модификации алгоритма CCL.

- Написана Python-библиотека, реализующая алгоритмы Condenced, CCL и naive и поддерживающая версии для центрального и графического процессоров. Также библиотека содержит версию алгоритма CCL для расчета δ-гиперболичности на наборах данных произвольного размера.

- Выполнено экспериментальное исследование:

  – проверена корректность реализованных алгоритмов посредством сравнения с эталонным решением (наивный подход), реализованным для CPU и GPU на синтетических данных и на наборах данных Amazon и MovieLens;

  – сделаны замеры производительности относительно эвристики, разработанной в лаборатории «Вычислительного интеллекта», наилучшие результаты показал алгоритм CCL, реализованный на GPU.

  С кодом библиотеки можно ознакомится по ссылке – https://github.com/tnmtvv/hypdelta/tree/main.

  Эксперименты проводились с помощью скриптов, представленных в репозитории – https://github.com/tnmtvv/Gromov-delta-estimation/tree/main.

Список литературы

[1] Aghdam Mehdi Hosseinzadeh, Analoui Morteza, Kabiri Peyman. Collaborative filtering using non-negative matrix factorisation // Journal of Information Science. — 2017. — Vol. 43, no. 4. — P. 567–579.

[2] Applying clique-decomposition for computing Gromov hyperbolicity / Nathann Cohen, David Coudert, Guillaume Ducoffe, Aurélien Lancin // Theoretical computer science. — 2017. — Vol. 690. — P. 114–139.

[3] CPython. CPython is the reference implementation of the Python programming language. // Repository. — URL: https://github.com/python/cpython (дата обращения: 2023-11-10).

[4] Cao Yueqi. Efficient Weingarten map and curvature estimation on manifolds // Mach. Learn. — 2021. — Vol. 110, no. 6. — P. 1319–1344. — URL: https://doi.org/10.1007/s10994-021-05953-4 (дата обращения: 2023-11-05).

[5] Cohen Nathann, Coudert David, Lancin Aurélien. On Computing the Gromov Hyperbolicity // ACM J. Exp. Algorithmics. — 2015. — aug. — Vol. 20. — 18 p. — URL: https://doi.org/10.1145/2780652 (дата обращения: 2023-10-02).

[6] Cython. Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language // Repository. — URL: https://github.com/cython/cython (дата обращения: 2023-11-10).

[7] Diameters, centers, and approximating trees of delta-hyperbolicgeodesic spaces and graphs / Victor Chepoi, Feodor Dragan, Bertrand Estellon et al. // Proceedings of the twenty-fourth annual symposium on Computational geometry. — 2008. — P. 59–68.

[8] Doersch Carl. Tutorial on Variational Autoencoders. — 2021. — 1606.05908.

[9] Efficient Curvature Estimation for Oriented Point Clouds / Yueqi Cao, Didong Li, Huafei Sun et al. // ArXiv. — 2019. — Vol. abs/1905.10725. — URL: https://api.semanticscholar.org/CorpusID:166228634 (дата обращения: 2023-11-05).

[10] Fournier Hervé, Ismail Anas, Vigneron Antoine. Computing the Gromov hyperbolicity of a discrete metric space // Information Processing Letters. — 2015. — Vol. 115, no. 6-8. — P. 576–579.

[11] Ganea Octavian, Becigneul Gary, Hofmann Thomas. Hyperbolic Entailment Cones for Learning Hierarchical Embeddings. — 2018. — 04.

[12] Gromov M. Essays in group theory. — London : MSRI Publications, 1987. — P. 75–263.

[13] Harper F. Maxwell, Konstan. Joseph A. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS). — 2015. — URL: https://grouplens.org/datasets/movielens/1m/ (дата обращения: 2023-09-15).

[14] Herlocker Jonathan L, Konstan Joseph A, Riedl John. Explaining collaborative filtering recommendations // Proceedings of the 2000 ACM conference on Computer supported cooperative work. — 2000. — P. 241–250.

[15] Hyperbolic Embeddings in Sequential Self-Attention for Improved Next-Item Recommendations / Evgeny Frolov, Lina Bashaeva, Leyla Mirvakhabova, Ivan Oseledets. — 2023.

[16] Li Yicong, Chen Hongxu, Sun Xianguo et al. Hyperbolic Hypergraphs for Sequential Recommendation. — 2021. — 08.

[17] Hyperbolic Image Embeddings / Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova et al. // 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). — 2020. — P. 6417–6427.

[18] Hyperbolic Representation Learning: Revisiting and Advancing / Menglin Yang, Min Zhou, Rex Ying et al. — 2023. — 06.

[19] Hyperbolic Representation Learning: Revisiting and Advancing / Menglin Yang, Min Zhou, Rex Ying et al. // Proceedings of the 40th International Conference on Machine Learning. — ICML'23. — JMLR.org, 2023. — 21 p.

[20] Hyperbolic audio-visual zero-shot learning / Jie Hong, Zeeshan Hayder, Junlin Han et al. // Proceedings of the IEEE/CVF International Conference on Computer Vision. — 2023. — P. 7873–7883.

[21] Hyperbolic geometry of complex networks / Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak et al. // Physical Review E. — 2010. — sep. — Vol. 82, no. 3. — URL: https://doi.org/10.1103 (дата обращения: 2023-10-30).

[22] Hyperbolic image embeddings / Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova et al. // Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. — 2020. — P. 6418–6428.

[23] Hyperbolic vision transformers: Combining improvements in metric learning / Aleksandr Ermolov, Leyla Mirvakhabova, Valentin Khrulkov et al. // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. — 2022. — P. 7409–7419.

[24] Kang Wang-Cheng, McAuley Julian. Self-Attentive Sequential Recommendation. — 2018. — 1808.09781.

[25] Koren Yehuda, Rendle Steffen, Bell Robert. Advances in collaborative filtering // Recommender systems handbook. — 2021. — P. 91–142.

[26] Learning a Hierarchical Embedding Model for Personalized Product-Search. / Qingyao Ai, Yongfeng Zhang, Keping Bi et al. // In Proceedings of SIGIR '17. — 2017. — URL: https://jmcauley.ucsd.edu/data/amazon/ (дата обращения: 2023-09-15).

[27] Malakhov Anton. Composable Multi-Threading for Python Libraries. // SciPy. — 2016. — P. 15–19.

[28] Neural collaborative filtering / Xiangnan He, Lizi Liao, Hanwang Zhang et al. // Proceedings of the 26th international conference on world wide web. — 2017. — P. 173–182.

[29] Nishino ROYUD, Loomis Shohei Hido Crissman. Cupy: A numpy-compatible library for nvidia gpu calculations // 31st confernce on neural information processing systems. — 2017. — Vol. 151, no. 7.

[30] On Computing the Hyperbolicity of Real-World Graphs / Michele Borassi, David Coudert, Pierluigi Crescenzi, Andrea Marino. — 2015. — 09. — Vol. 9294.

[31] On computing the hyperbolicity of real-world graphs / Michele Borassi, David Coudert, Pierluigi Crescenzi, Andrea Marino // Algorithms-ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings / Springer. — 2015. — P. 215–226.

[32] Performance of hyperbolic geometry models on top-N recommendation tasks / Leyla Mirvakhabova, Evgeny Frolov, Valentin Khrulkov et al. // Proceedings of the 14th ACM Conference on Recommender Systems. — 2020. — P. 527–532.

[33] Popov Andrey, Iacob Andrei. Lobachevsky geometry and modern nonlinear problems. — Springer, 2014.

[34] Project The PyPy. PyPy documentation. — URL: https://doc.pypy.org/en/latest/architecture.html (дата обращения: 2023-11-10).

[35] Pypy. Python programming language interpreter. // Repository. — URL: https://www.pypy.org/ (дата обращения: 2023-11-10).

[36] Rokhlin Vladimir, Szlam Arthur, Tygert Mark. A Randomized Algorithm for Principal Component Analysis // SIAM Journal on Matrix Analysis and Applications. — 2008. — 09. — Vol. 31.

[37] SciPy 1.0: fundamental algorithms for scientific computing in Python / Pauli Virtanen, Ralf Gommers, Travis E Oliphant et al. // Nature methods. — 2020. — Vol. 17, no. 3. — P. 261–272.

[38] Scikit-learn: Machine learning in Python / Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort et al. // the Journal of machine Learning research. — 2011. — Vol. 12. — P. 2825–2830.

[39] Sigarreta Jose M. Hyperbolicity in median graphs // Proceedings-Mathematical Sciences. — 2013. — Vol. 123. — P. 455–467.

[40] Skolkovo Institute of Science and Technology, Computational Intelligence Group. — URL: https://new.skoltech.ru/en/laboratories/computational-intelligence (дата обращения: 2023-11-30).

[41] Tifrea Alexandru, Bécigneul Gary, Ganea Octavian-Eugen. Poincaré GloVe: Hyperbolic Word Embeddings // ArXiv. — 2018. — Vol. abs/1810.06546. — URL: https://api.semanticscholar.org/CorpusID:53116133 (дата обращения: 2023-09-10).

[42] Verbeek Kevin, Suri Subhash. Metric embedding, hyperbolic space, and social networks // Proceedings of the thirtieth annual symposium on Computational geometry. — 2014. — P. 501–510.

[43] Verbeek Kevin, Suri Subhash. Metric embedding, hyperbolic space, and social networks // Computational Geometry. — 2016. — Vol. 59. — P. 1–12. — URL: https://www.sciencedirect.com/science/article/pii/S0925772116300712 (дата обращения: 2023-10-01).

[44] Yuki Ryo, Ike Yuichi, Yamanishi Kenji. Dimensionality Selection of Hyperbolic Graph Embeddings using Decomposed Normalized Maximum Likelihood Code-Length // 2022 IEEE International Conference on Data Mining (ICDM). — 2022. — P. 666–675.

[45] numba. A Just-In-Time Compiler for Numerical Functions in Python // Repository. — URL: https://github.com/numba/numba (дата обращения: 2023-11-10).