Отчет по практическому заданию «Параллельная реализация алгоритма многомерного шкалирования»

Сенин Александр, 417

1 декабря 2020 г.

Введение

Данное введение, как и несколько следующих параграфов, во многом повторяет и значительно дополняет соответствующую статью на АлгоВики. Задача этого введения: объяснить математическую и алгоритмическую суть задачи, а также последовательность ее решения. Для неподготовленного читателя вкратце опишем, как обычно ставятся задачи машинном обучении. В классическом машинном обучении имеется множество объектов, для каждого объекта предполагается некоторый ответ (целевая переменная). Считаем, что существует некоторая зависимость между объектами и ответами, в общем случае она неизвестна. Общая задача машинного обучения и состоит в том, чтобы восстановить эту зависимость: для каждого объекта предсказать соответствующую ему целевую переменную. Обычно, у нас уже есть некоторые знания об этой зависимости, чаще всего они выражаются в совокупности прецедентов: пар (объект, целевая переменная). Такая совокупность называется обучающей выборкой. Предполагается, что в конечном счете мы для любого объекта будем уметь возвращать целевую переменную. Существенная часть алгоритмов машинного обучения сужает понятие объекта до конечного вектора - признакового описания, получает матрицу объекты-признаки, каждая строка такой матрицы соответствует объекту, и

каждой строке соответствует целевая переменная.

Задача многомерного шкалирования относится скорее к задаче анализа данных. В случае задачи многомерного шкалирования ситуация иная: считаем, что у нас нет никакой информации на конкретном объекте, но у нас есть информация о всевозможных парах объектов - обычно, эта информация несет смысл сходства или различия. Вместо входной матрицы объекты-признаки в терминах машинного обучения, у нас есть входная матрица попарных сходств или различий. Наша задача по этой матрице визуализировать исходную совокупность объектов, по которой эта матрица и была посчитана. Визуализировать будем следующим образом: найдем конфигурацию точек в двух или трехмерном пространстве, которая будет наиболее близко описывать исходную, нам неизвестную выборку объектов (каждая точка взаимно однозначно соответствует объекту).

Постановка получилась сильно общей, вполне естественно существование более узких постановок и разных подходов к решению. Резюмируем общую задачу: по вещественной матрице попарных различий найти для каждого объекта такое положение в пространстве размерности p (чаще всего $p = 2, 3$), что попарные различия будут лучше всего сохранены. Тогда общее описание алгоритма: получаем на вход квадратную вещественную матрицу D размера NxN, возвращаем N векторов размерности p — координаты точек, которые лучше всего описывают наши объекты.

Математическое решение задачи

Конкретизируем подход к решению задачи многомерного шкалирования. Будем рассматривать задачу классического многомерного шкалирования (Classical Multidimensional Scaling, cMDS).

Пусть входная матрица различий - матрица попарных расстояний для евклидовой метрики $D = (d_{ij})$. Здесь предполагаем, что данные нам различия во входной матрице — расстояния, с точки зрения математики, причем посчитанные евклидовой метрикой. Это допущение позволит найти конфигурацию, которая идеально точно воспроизводит расстояния на парах, но про размерность полученных координат мы

не сможем ничего сказать. Однако, даже если метрика была не евклидовой, или даже не была формально математическим расстоянием, а также в случае, если требуются координаты фиксированной небольшой размерности, мы все равно получим результат.

Задача классического MDS (cMDS) — найти $X = (x_1, ..., x_N)^T$, т.ч. $d_{ij} = ||x_i - x_j||_2$.

Решение не единственно, потому что $X^* = X + c^T$ тоже решение, так как $d_{ij} = ||x_i - x_j|| = ||(x_i + c) - (x_j + c)||$. Поэтому будем искать центрированную конфигурацию $\overline{x} = 0$. Матрица D евклидова, т.е. $\exists \{x_i\}_{i=1}^N \in \ R^p$, т.ч. $d_{ij}^2 = (x_i - x_j)^T (x_i - x_j)$

Тогда приходим к задаче восстановления $\{x_i\}_{i=1}^N \in R^p$, при условии $\overline{x} = 0$.

Попробуем восстановить матрицу Грама $B = (b_{ij})$, где $b_{ij} = x_i^T x_j$. Обозначим $X = (x_1, ..., x_N)^T$, тогда $B = XX^T$. Предположим, мы найдем спектральное разложение матрицы Грама $B = \Gamma \Lambda \Gamma^T$, тогда, если положим $X = \Gamma \Lambda^{\frac{1}{2}}$, то мы решим поставленную задачу и восстановим искомые координаты. Таким образом, если получим матрицу Грама, то сможем по ее спектральному разложению получить X.

Восстановление матрицы Грама в cMDS:

$d_{ij}^2 = (x_i - x_j)^T (x_i - x_j) = x_i^T x_i + x_j^T x_j - 2x_i^T x_j = b_{ii} + b_{jj} - 2b_{ij}$

$\overline{x} = 0 \Rightarrow \sum_{i=1}^N b_{ij} = 0$

$\frac{1}{N} \sum\limits_{i=1}^N d_{ij}^2 = \frac{1}{N} \sum\limits_{i=1}^N b_{ii} + b_{jj} \ \frac{1}{N} \sum\limits_{j=1}^N d_{ij}^2 = b_{ii} + \frac{1}{N} \sum\limits_{j=1}^N b_{jj}$

$\frac{1}{N^2} \sum\limits_{i,j=1}^N d_{ij}^2 = \frac{2}{N} \sum\limits_{i=1}^N b_{ii}$

$b_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i\bullet}^2 - d_{\bullet j}^2 + d_{\bullet\bullet}^2)$

Строим по D матрицу Грама B:

$b_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i\bullet}^2 - d_{\bullet j}^2 + d_{\bullet\bullet}^2)$

Перепишем в матрично-векторном виде, $B = C_N A C_N$, где $A = -\frac{1}{2} D^2$ — поэлементное возведение в квадрат, $C_n = E - \frac{1}{n} \mathbf{1}\mathbf{1}^T$.

Выше получено, что $B = XX^T$, $X \in R^{N \times p}$, а значит B симметричная, неотрицательно определенная, ранга $rgB = rgXX^T = rgX = p$. B имеет p положительных собственных значений и $n - p$ нулевых собственных значений.

Используем спектральное разложение, $B = \Gamma \Lambda \Gamma^T$, где $\Lambda = diag(\lambda_1, ..., \lambda_p)$ и $\Gamma = (\gamma_1, ..., \gamma_p)$ матрица собственных векторов. Отсюда находим искомый итоговый результат $X = \Gamma \Lambda^{\frac{1}{2}}$.

Важное замечание: таким образом, мы найдем конфигурацию точек некоторой фиксированной, вообще говоря, нами не выбираемой, размерности p, которая может быть сильно больше целевых для задач визуализации $p = 2, 3$ (и многих других задач). Существуют теоремы, которые показывают, что можно смотреть на величину собственного значения, как на показатель полезности соответствующего собственного вектора, с точки зрения сохранения набольшей информации. Другими словами, наибольший собственный вектор описывает направление наибольшего разброса данных, то есть проецирование данных на это направление сохранит больше всего информации. Тогда, если полученное с помощью алгоритма представление данных X обладает неудовлетворительно большой размерностью, то смело можем оставить в матрице Γ только то число собственных векторов, какую размерность точек хотим получить, причем будем оставлять вектора, отвечающие наибольшим собственным значениям.

Рассмотрим теперь случай, когда расстояния были посчитаны произвольной, необязательно евклидовой метрикой. Существуют соответстующая теорема, которая обещает нахождение конфигурации с точными равенствами попарных расстояний для евклидовой метрики, но для неевклидовой метрики существование такой конфигурации обычно не гарантируется. Этому случаю соответствуют, например, существование отрицательных собственных значений в спектральном разложении. Здесь решение аналогичное: находим матрицу Грама, находим ее спектральное разложение, а далее оставляем все те же p собственных векторов, отвечающих наибольшим собственным значениям.

Далее будем рассматривать конкретное приложение классического многомерного шкалирования во многих задачах машинного обучения, а именно понижение размерности. Эта процедура применятся к данным для снижения вычислительной сложности последующих алгоритмов, которые будут с этими данными оперировать. В общей постановке многомерного шкалирования считается, что у нас от данных остаются только попарные расстояния, однако, для нашей цели этот лишний шаг избыточен: если у нас уже известны полные данные, нам нет смысла дополнительно считать матрицу попарных расстояний (обычно огромного размера), мы можем сразу

посчитать матрицу Грама, а по ней найти спектральное разложение, оставить в разложении p собственных векторов, отвечающих наибольшим собственным значениям, и спроецировать на них (с предварительным умножением на корни собственных значений) наши данные. Другими словами, используем всю доступную нам информацию: вместо вычисления матрицы попарных расстояний, восстановления по ней матрицы Грама, будем сразу же считать матрицу Грама.

Наше соображение о том, что в спектральном разложении нужно брать не все собственные векторы, чтобы получить представление нужной размерности, позволяет упростить задачу: нам не требуется строить полное спектральное разложение, а именно находить все собственные векторы и собственные значений, достаточно найти лишь несколько наибольших собственных значений и соответствующих им собственных векторов.

Тогда наш последовательный алгоритм на довольно высоком уровне абстракции будет выглядеть следующим образом:

- Получаем на вход матрицу объекты-признаки X.
- Строим матрицу Грама попарных скалярных произведений XX^T.
- Находим p ее наибольших собственных значений и отвечающие им собственные векторы.
- Организуем собственные векторы в столбцы матрицы Γ, корни собственных значений располагаем на диагонали матрицы $\Lambda^{\frac{1}{2}}$, причем в порядке невозрастания собственных значений.
- Возвращаем восстановленную матрицу $X = \Gamma\Lambda^{\frac{1}{2}}$.

Алгоритм другой постановки (рассмотренной в введении) будет отличаться лишь первым пунктом, вместо него следует выполнить следующее:

- Получаем на вход матрицу попарных расстояний D.
- Возводим матрицу поэлементно в квадрат D^2.

- Находим $A = -\frac{1}{2}D^2$.
- Находим матрицу Грама $B = C_N A C_N$, где $C_n = E - \frac{1}{n}\mathbf{1}\mathbf{1}^T$.

В подавляющем большинстве задач задана именно выборка X, поэтому далее будем реализовывать параллельно именно первый алгоритм.

Нахождение собственных значений

Конкретизируем теперь некоторые шаги последовательного алгоритма. Матрица Грама с точки зрения вычислений определяется вполне однозначно, то же самое и с вычислением $X = \Gamma\Lambda^{\frac{1}{2}}$ на последнем шаге. Основной вопрос в нахождении собственных векторов и собственных значений. Для нахождения всего спектра матрицы можно использовать и метод вращения Якоби, и алгоритм QR, но в нашей задаче эти методы избыточны, нам не требуется весь спектр, нам требуется лишь несколько наибольших его элементов. Здесь нам очень удачно подойдет степенной метод нахождения собственных значений. Это итерационный алгоритм поиска собственного значения с максимальной абсолютной величиной и одного из соответствующих собственных векторов для произвольной матрицы A.

- Берем некоторый начальный вектор r_0.
- Итеративно пересчитываем $r_{k+1} = \frac{Ar_k}{||Ar_k||}$ до сходимости, так находим собственный вектор.
- Вычисляем $\mu_k = \frac{r_k^T A r_k}{r_k^T r_k}$ — соответствующее собственное значение.

Так мы найдем набольшее собственное значение и соответственный собственный вектор матрицы Грама. Что если мы хотим не одномерное представление данных (так бывает чаще всего)? Необходимо найти следующее за наибольшим по невозрастанию собственное значение. Тут нам поможет следующий факт: известно, что для матриц нормальных операторов все собственные векторы взаимно ортогональны, а значит мы можем найти искомое второе по величине собственное значение следующим образом:

- Вычисляем $A_1 = A - \lambda r_k r_k^T$ — матрицу, сохраняющую все собственные значения матрицы A, кроме λ. В качестве λ кладем найденное до этого наибольшее по модулю собственное значение.
- Применяем предыдущие шаги для нахождения следующего по модулю собственного значения, и т.д.

Сложность алгоритма

Глобально алгоритм состоит из независимых подчастей, порядок выполнения между которыми строго определен, другими словами, на макроуровне алгоритм требует последовательного выполнения своих частей. Оценим последовательную и параллельную сложность каждой подчасти, а также всего алгоритма в совокупности. Напомним алгоритм:

1. Получаем на вход матрицу объекты-признаки X.
2. Строим матрицу Грама попарных скалярных произведений XX^T.
3. Находим p ее наибольших собственных значений и отвечающие им собственные векторы, а именно:
   (a) Берем некоторый начальный вектор r_0.
   (b) Итеративно пересчитываем $r_{k+1} = \frac{Ar_k}{||Ar_k||}$ до сходимости, так находим собственный вектор.
   (c) Вычисляем $\mu_k = \frac{r_k^T A r_k}{r_k^T r_k}$ — соответствующее собственное значение.
   (d) Вычисляем $A_{new} = A - \lambda r_k r_k^T$ — матрицу, сохраняющую все собственные значения матрицы A, кроме λ. В качестве λ кладем найденное до этого наибольшее по модулю собственное значение. Повторяем степенной метод.
4. Организуем собственные векторы в столбцы матрицы Г, корни собственных значений располагаем на диагонали матрицы $\Lambda^{\frac{1}{2}}$, причем в порядке невозрастания собственных значений.

5. Возвращаем восстановленную матрицу $X = \Gamma\Lambda^{\frac{1}{2}}$.

Пусть входная матрица имеет размер n x k, столбцы суть признаки, строки суть объекты. Могут быть разные вариации взаимного отношения n и k, но чаще всего $n > k$, причем в подавляющем числе случаев $n >> k$, бывает даже на порядки. То есть о матрице X можно мыслить, как о прямоугольной.

Обсудим сложность (2) шага. На этом шаге мы считаем скалярные произведения строк (объектов). Для непосредственного вычисления матрицы Грама нам не требуется явно транспонировать матрицу. Достаточно вычислить nk скалярных произвведений, каждое скалярное произведение — это n умножений и $n-1$ сложение. Тогда последовательная сложность на этом шаге $O(n^2k)$.

Обсудим сложность (3b) шага. Число размерности искомой конфигурации точек p обычно $2, 3$ для задач визуализации, и много меньше k для задач понижения размерности. Поэтому множитель числа итераций здесь уйдет в константу O большого. Далее нам нужно итеративно до сходимости пересчитывать r_{k+1}. На одной итерации нужно одно умножение матрицы на вектор, и вычисление нормы полученного результата, затем деление каждой компоненты результата на норму. Важно: теперь матрица A — суть матрица Грама, а значит имеет размеры n x k. Вычисление Ar_k требует n^2 умножений и сложений. Вычисление нормы по сути скалярный квадрат, то есть n умножений и $n-1$ сложение, деление на норму еще n умножений. Итоговая сложность этого шага $O(n^2)$, основные вычислительные затраты идут на умножение матрицы на вектор. Если получится распараллелить алгоритм так, что можно считать сложность умножения матрицы на вектор линейной, то от всего шага получится добится линейной сложности. Более того, мы ищем собственные значения итерационно, а значит этот шаг будет повторятся сильно чаще остальных, это тоже нужно держать в уме при оценки сложности всего алгоритма, а также при выборе стратегии распараллеливания.

Шаг (3c) представляет собой два скалярных произведения (умножение матрицы на вектор получено на предыдущем шаге), то есть имеет линейную сложность $O(n)$.

На шаге $(3d)$ требуется по вектору посчитать матрицу, это n^2 умножений, а затем

вычесть результат из матрицы предыдущего шага, это еще n^2 сложений. Последовательная сложность этого шага $O(n^2)$.

Шаг (4) может быть выполнен грамотной организацией структур данных еще на предыдущих шагах, из вычислительных операций здесь только взятие p корней, в общей сложности этот шаг учитывать не будем.

Шаг (5) по сути представляет собой умножение p векторов длины n на число, можем считать его сложность $O(n)$.

Параллельную сложность обсудим в следующих параграфах.

Ресурс параллелизма

Из обсуждения последовательной сложности алгоритма ясно следующее: просматривается последовательная структура на макроуровне, но на уровне каждой подзадачи оперируем с понятиями параллельной природы: матрицы, векторы, их совместные операции. По сложности выделяются первый этап вычисления матрицы Грама со сложностью $O(n^2k)$ и умножение матрицы на вектор $O(n^2)$. Причем, вторая операция имеет сильно больший приоритет — нам нужно умножать матрицу на вектор на каждой итерации степенного метода, а эти итерации будут повторяться до сходимости, то есть потенциально значительное число раз. Все остальные этапы по сути имеют линейную сложность $O(n)$, за исключением построения новой матрицы A, однако это операцию нужно провести всего p раз ($p << n$) по сравнению с несравнимо большим число итераций для сходимости степенного метода.

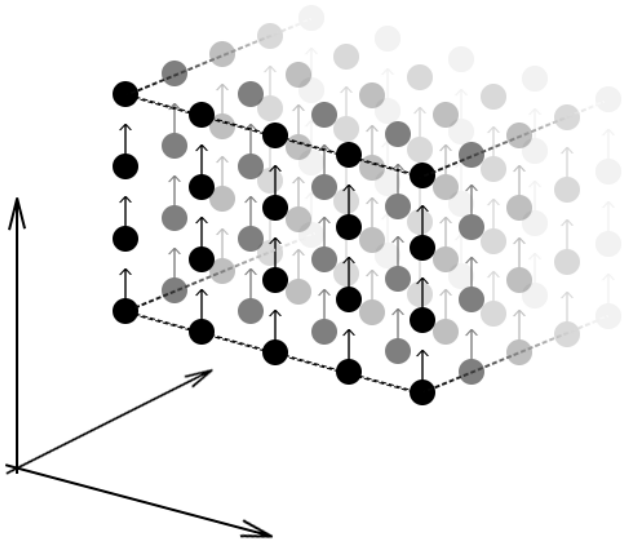

Рис. 1: Информационная структура алгоритма вычисления XX^T

Обсудим ресурса параллелизма вычисления XX^T. По сути это n^2 скалярных произведений векторов длины k.

Для вычисления XX^T нам потребуется выполнить следующие ярусы:

- n ярусов умножений и сложений, в каждом ярусе nk операций

Таким образом, по высоте ЯПФ имеем линейную сложность, по ширине билинейную.

Также улучшение можно сделать еще на уровне алгоритма: матрица симметрична, а значит половину вычислений можно упростить (визуально на Рис.1 нужно провести сечение параллелепипеда, проходящее через ближайшее к нам ребро, и через самое дальнее для нас ребро).

Обсудим ресурс параллелизма умножения матрицы на вектор Ar_k.

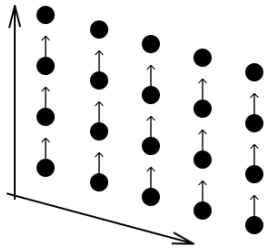

Рис. 2: Информационная структура алгоритма вычисления Ar_k

Для вычисления Ar_k нам потребуется выполнить следующие ярусы:

- n ярусов умножений, в каждом ярусе n операций

Таким образом, и по высоте ЯПФ, и по ширине ЯПФ имеем линейную сложность. Это важный результат, т.к. умножение матрицы на вектор — самая частая операция в нашей постановке алгоритма cMDS, потому что встречается на каждой итерации степенного метода.

Все остальные операции в последовательном алгоритме (см. шаги в предыдущем параграфе), как было выяснено, имеют линейную сложность, причем «природного» параллелизма в них нет. Тогда, если мы распараллелим вычисление матрицы Грама и умножение матрицы на вектор, то сложность алгоритма сведется к единразовому вычислению матрицы Грама с билинейной сложностью $O(nk)$, а затем вычислительное ядро алгоритма будет обладать линейной сложностью $O(n)$ (тут важно понимать, что число итераций степенного метода может несколько ухудшить эту оценку), p раз потребуется вычислить $r_k r_k^T$ за $O(n^2)$ операций.

Программная реализация

Рассмотрим теперь конкретную программную реализацию описанного алгоритма. Программа написана на C++ в процедурном стиле, стандарт $std = c++17$, векторы и матрицы реализованы посредством $std :: vector < double >$ и $std :: vector < std :: vector < double >>$ соответственно. Параллелизм достигается посредством исполь-

зования OpenMP, а именно директив параллельного вычисления соответствующих циклов for в функциях умножения матрицы на вектор, и вычисления матрицы Грама XX^T. Везде, где это возможно, матрицы и векторы передаются по константной ссылке, чтобы избежать лишнего копирования больших массивов данных. В местах алгоритма, где будет эффективно вычитать из матрицы матрицу процедурно (то есть изменяя исходную матрицу, вместо конструирования и заполнения новой), например, при пересчете матрицы A_{new} это выполнено. Последовательный код алгоритма писался с нуля непосредственно для выполнения этого практического задания. По сути единственный ключевой параметр компилятора: $-fopenmp$.

Замечания, обнаруженные при написании программы: важно следить за разделяемыми и приватными переменными, например, чтобы избежать race condition при вычислении XX^T, в момент добавления очередного слагаемого в скалярном произведении векторов, стоит завести локальную для каждого потока сумму, а уже в конце записать ее в общую переменную. При реализации степенного метода поиска собственного значения стоит под сходимостью понимать не только норму разницы вектора, полученного на предыдущей итерации и на текущей, но и ограничение на число итераций.

Проверка корректности работы программы именно с точки зрения правильности выходных данных осуществлялась следующим образом: был взят «игрушечный» датасет Ирисы Фишера. По сути, это таблица со 150 строками и 4 столбцами, каждая строка соответствует некоторому наблюдаемому цветку ириса, а в столбцах — некоторые величины, измерянные по цветку. Эта таблица была взята в качестве матрицы X в терминах алгоритма, далее к ней был применен алгоритм для $p = 2$ и $p = 3$, в результате было полученное двумерное и трехмерное представление для данных. Визуализацию результата можно увидеть на Рис.3 и Рис.4.

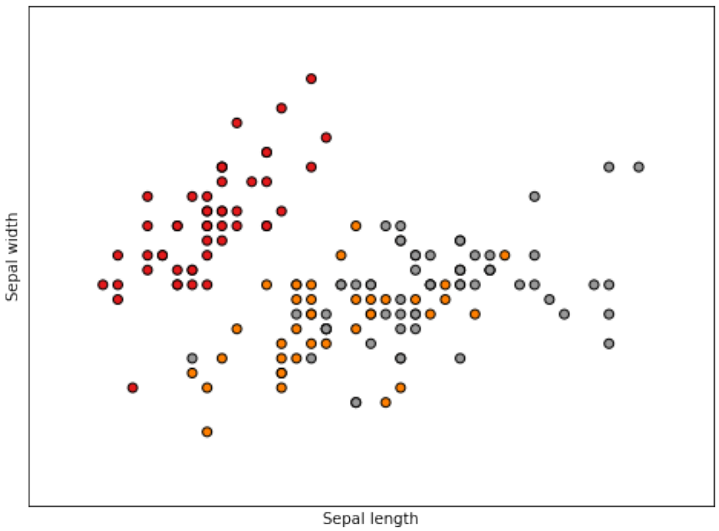

Рис. 3: Корректность выходных данных на датасете Ирисы Фишера при $p = 2$

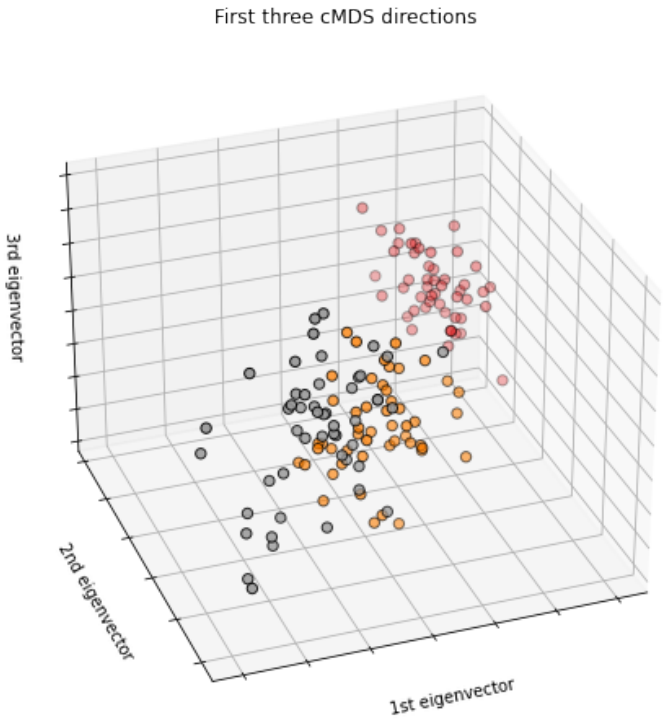

Рис. 4: Корректность выходных данных на датасете Ирисы Фишера при $p = 3$

Цвета на рисунке взяты из разметки, приложенной к данным: датасет подразумевается для решения задачи классификации, а именно определить к какому из трех сортов ирисов относится цветок. Результат корректен, т.к. явно видно, что при понижении размерности в данных сохранилась структура, кластеры цветков одного

сорта явно прослеживаются, можно применить стороннюю реализацию $cMDS$ из $sklearn$, и получить аналогичный результат.

Исследование сильной маштабируемости на суперкомпьютере

По определению, сильная масштабируемость — зависимость производительности от количества процессоров при фиксированной вычислительной сложности задачи. Вычислительная сложность обычно есть функция от размеров задачи. Соответственно, под фиксированной сложностью будем пониманить фиксирование входной матрицы X. Тогда будем варьировать число потоков $num\_threads$, и оценивать производительность. Кажется разумным вместо непосредственно производительности замерять время работы. Оценивание будем производить за суперкомпьютере Ломоносов 2, на одном узле *compute* (по причине слишком долгого ожидания в очереди при запросе большего числа узлов), а значит на процессоре Intel Xeon E5-2697 v3 2.60GHz с 14 CPU-ядер и 64 GB ОЗУ. В качестве входных данных будем генерировать случайную матрицу. Число признаков k фиксируем равным 3000, для большей полноты картины число объектов n — число столбцов матрицы X будем менять от 20000 до 40000 с шагом в десять тысяч. Каждый замер повторялся 5 раз, а затем результат усреднялся. Замеры отобразим на графике Рис.5.

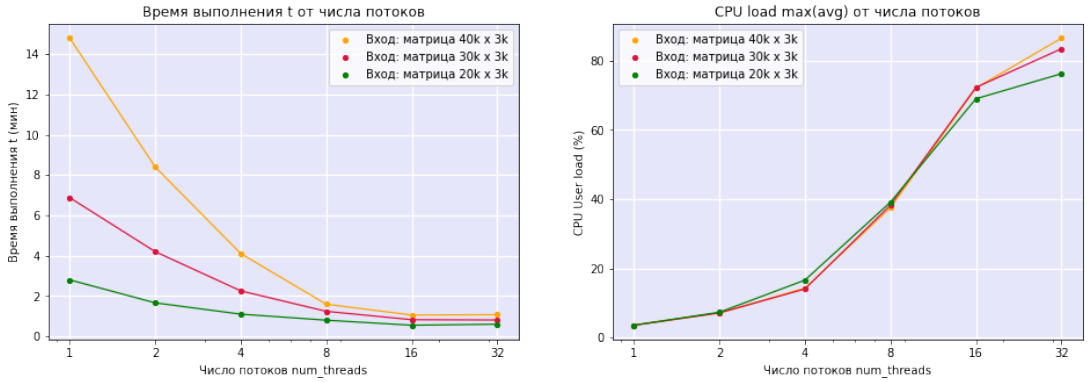

Рис. 5: Результаты запусков на суперкомпьютере Ломоносов 2. Процент загрузки CPU оценивался в разделе эффективность с Octoshell, для этого брался максимум средней загруженности avg на участке, соответствующем вычислению cMDS при данном n и p, этот показатель может иметь серьезную погрешность, стоит его рассматривать лишь для оценки общей динамики. С учетом 14 ядер, реализация закономерно нагружает процессор в соответствии с выбранным числом потоков.

Можем агрегировать полученные замеры на одном графике Рис.6.

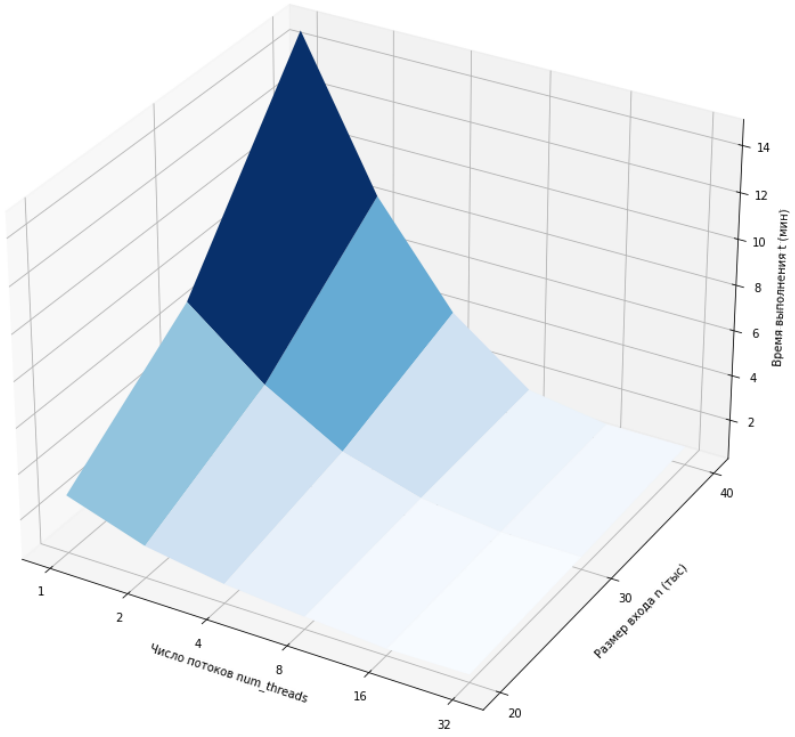

Рис. 6: Результаты запусков на суперкомпьютере Ломоносов 2.

Прокомментируем полученный результат. Действительно получилось значительно

ускорить время работы алгоритма за счет распараллеливания самых частых операций вычислительного ядра алгоритма. Однако, сложность итогового алгоритма далека от линейной, причинами этому являются последовательная структура алгоритма на макроуровне (параллелизма удалось достичь лишь на фрагментах микроуровня алгоритма), а также число итераций степенного метода (помним, что на одной итерации за счет параллелизма мы можем достичь линейной сложности) все таки усложняет алгоритм, кроме того, дает вклад билинейная сложность вычисления XX^T перед началом выполнения алгоритма. Увеличение сложности задачи (размера входной матрицы) закономерно приводит к значительному увеличению времени работы, однако параллелизм позволяет компенсировать это увеличение. Тем не менее, получен хороший результат: в случае матрицы 40000 строк на 3000 столбцов удалось сократить время работы с порядка 15 минут до около 1 минуты.

Выводы и результаты

- Для выполнения практического задания была написана реализация cMDS на C++ и OpenMP.
- Для читателя, незнакомого с алгоритмом, было представлено его объяснение, вывод, приведены примеры задач, в которых он применяется.
- Проведено исследование последовательной и параллельной сложности, сильной маштабируемости, проведены запуски на суперкомпьютере Ломоносов 2, результаты визуализированы и проинтерпретированы.