Алгоритм SparseDTW
SparseDTW: A Novel Approach to Speed up Dynamic Time Warping by Al-Naymat et al.

Белобородов Дмитрий

517 группа

20 мая 2018 г.

Постановка задачи

Dynamic Time Warping (DTW): выравнять последовательности $x \in \mathbb{R}^n, y \in \mathbb{R}^m$ с функцией штрафа f :

$$\sum_{i,j \in path} f(x_i, y_j) \to \min_{path}$$

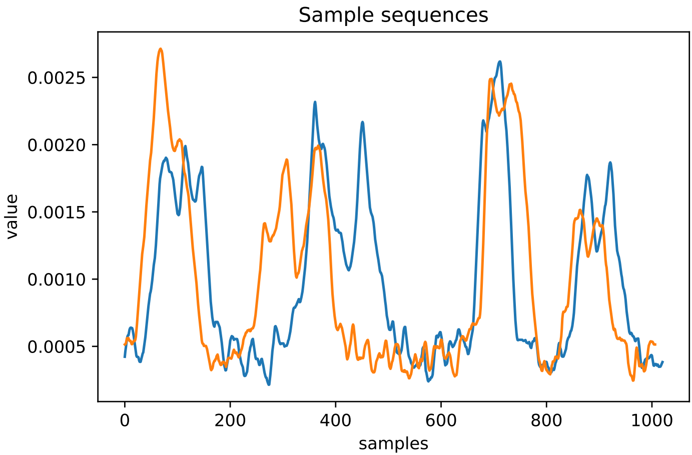
Требования к пути:

- Начинается в $(1, 1)$, заканчивается в (n, m);
- Индексы монотонно не убывают;
- Индексы могут возрастать только на 1.

Динамический пересчет цены пути:

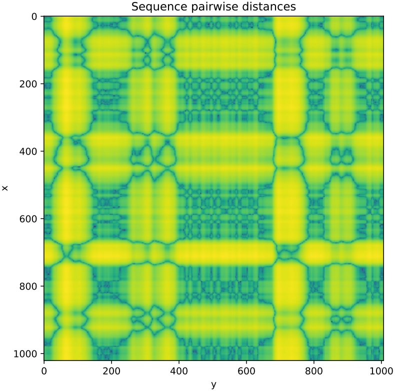
$$D_{ij} = f(x_i, y_j) + min(D_{i-1,j-1}, D_{i-1,j}, D_{i,j-1})$$

Пример выравнивания

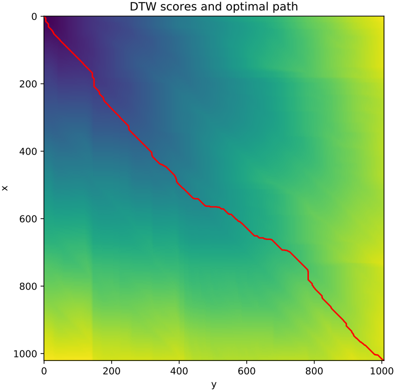


Пример: две последовательности

$$f(x_i, y_j) = |x_i - y_j|^{0.1}$$



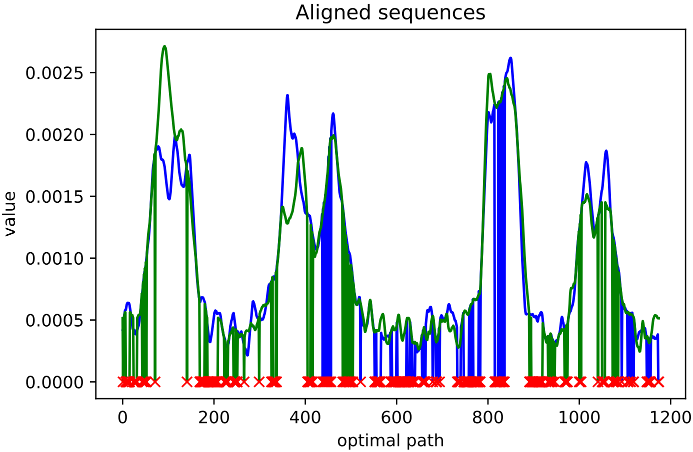
(a)



(b)

Попарные расстояния (a); стоимости путей и оптимальный путь (b).

Визуализация выравнивания



Выравнивание: пропуски соответствуют выравниванию с одним элементом

SparseDTW: идея и свойства

Идея: считать не все элементы матрицы, а только те, что соответствуют похожим элементам; хранить их в виде sparse-матрицы.

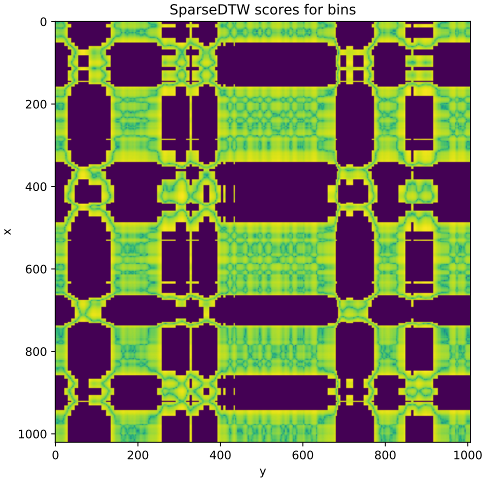
Свойства:

- Память $O(mn)$ (как и DTW);
- Время $O(mn)$ (как и DTW);
- Всегда оптимален (действительно?)
- Оптимальность не зависит от параметров (действительно?)
- Есть сомнения, что вообще работает.

Шаг 1: квантование

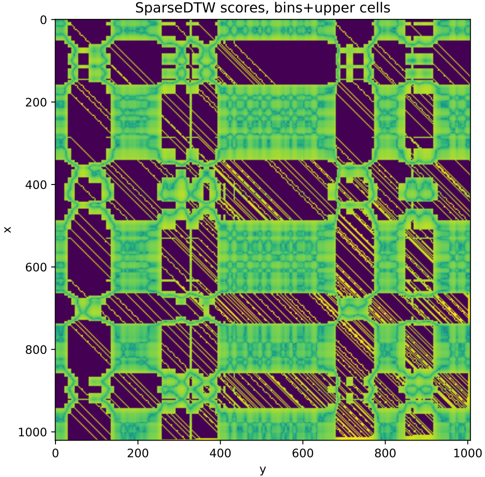
- Разобьем диапазоны значений x и y на B перекрывающихся ячеек с коэффициентом перекрытия β;
- Например, для
  $B = 4, \beta = 2: \;\; [0, 0.5], [0.25, 0.75], [0.5, 0.75], [0.75, 1];$
- Для каждой ячейки найдем индексы из x и y попадающих в нее элементов;
- Для всех таких пар индексов отметим ячейки $D_{ij} = f(x_i, y_j)$;
- Получаем блочную разреженную матрицу.

Шаг 1: квантование



Области могут быть разорваны, пути не удовлетворяют условиям.

Шаг 2: продолжение путей

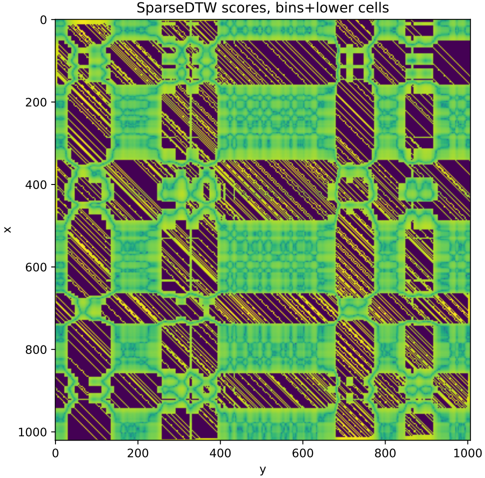


Если у элемента нет правых-нижних соседей, добавить их.

После этого предлагается запустить обычный пересчет DTW и искать путь обратным проходом.

На самом деле не работает: можем застрять в углу и не попасть в $(0, 0)$.

(Шаг 3): еще раз продолжение путей

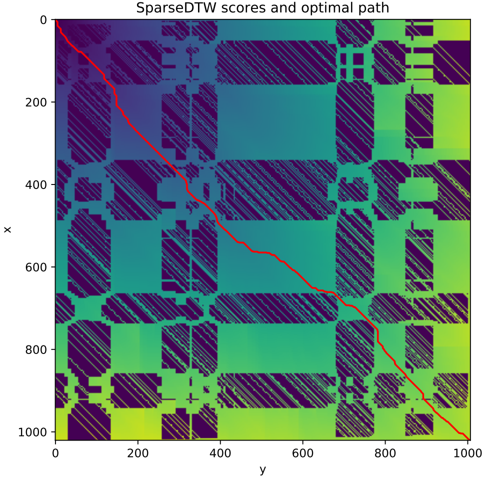


Если у элемента нет верхних-левых соседей, добавить их.

Хуже точно не станет: сложность не изменилась, ошибка может только уменьшиться.

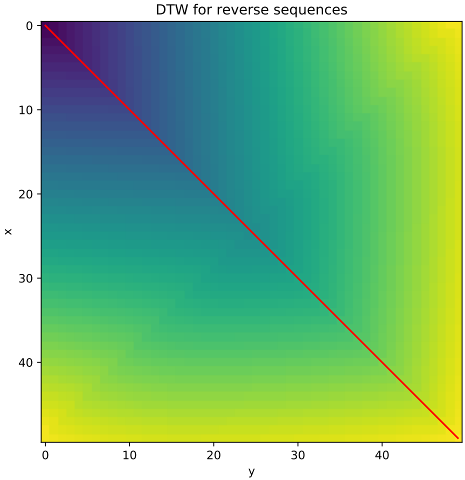
Этого нет в статье.

Шаг 4: подсчет стоимости и поиск пути



Так же, как в DTW, но с учетом только отмеченных ячеек.

Пример: перевернутая последовательность

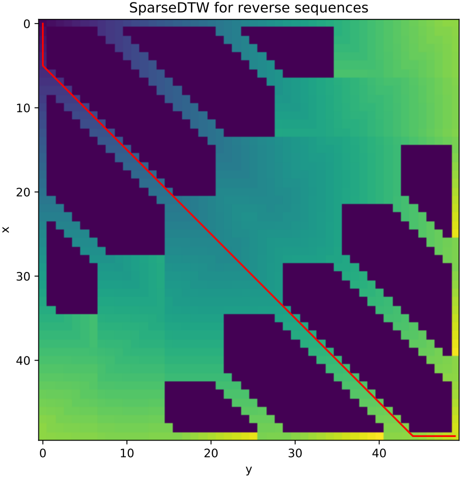


$$x = [1, \ldots, 50]$$

$$y = [50, \ldots, 1]$$

Оптимальный путь: прямое сопоставление элементов (главная диагональ в матрице путей).

Пример: перевернутая последовательность

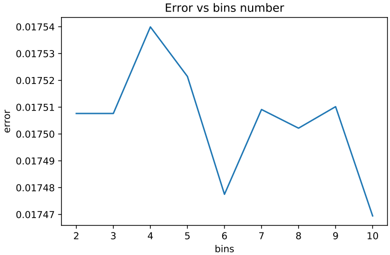


Пример для SparseDTW:

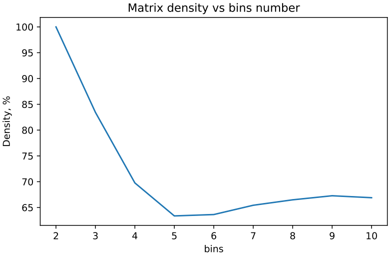
- Показывает необходимость Шага 3;
- Показывает неоптимальность;
- Позволяет оценить сложность:

$$N_{bin} \geqslant B\left(\frac{n}{B}\right)^2 = \frac{n^2}{B}$$
$$N_{path} = O(nB)$$
$$N = N_{bin} + N_{path} =$$
$$O(nB) + O(\frac{n^2}{B})$$

Для тестовых последовательностей:



(a)



(b)

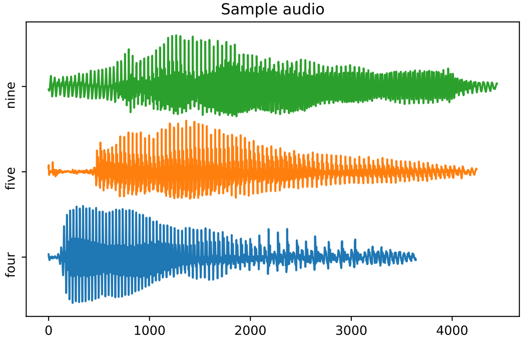
Зависимость ошибки (a) и плотности матрицы (b) от B.

Плотность матрицы не опускается ниже 0.6, при маленьких и больших B увеличивается.

Применение алгоритма DTW к звукам

Три аудиозаписи: «four», «five», «nine».

$z_k = 0.5(x_i + y_j), \; (i, j) = path_k$



four | five | nine

five-four | five-nine | nine-four

Итоги

- Не похоже, что SparseDTW работает с любыми последовательностями;
- Не похоже, что он всегда выдает оптимальный результат: зависит от параметров;
- По памяти и по сложности все еще $O(mn)$, хотя есть субоптимальные алгоритмы с $O(m+n)$ памятью;
- Все равно медленнее, чем DTW (особенность реализации);
- Возможно, стоит разбивать диапазон значений не равномерно, а по квантилям — будет меньше ячеек и разрывов (нет в статье).