Алгоритм GISTIS

Alex Zabelin

December 2018

1 Введение

В этом докладе будет рассмотрена задача поиска изображений по содержанию и один из подходов к её решению – алгоритм GISTIS. Задача в общем виде ставится следующим образом: необходимо в большой коллекции найти изображения с содержанием похожим на заданное. Содержание можно задавать разными способами, например текстовым описанием, похожей картинкой или распределением цветов. Кроме этого, по-разному можно определять похожи изображения или нет. Картинки могут быть визуально похожи друг на друга, между картинками могут быть семантические сходства, как между салоном самолета и лекционной аудиторией. Мы будем рассматривать задачу поиска изображений, которые визуально похожи на заданное.

2 Дескриптор GIST

Для описания изображений есть ряд хорошо известных, относительно простых методов, которые основаны на рассмотрении распределений градиентов. Одним из таких методов является метод Гистормм Направленных Градиентов (HOG). Изображение делится на блоки $\{Block_i\}$, в блоке для каждого пикселя $(j,k) \in Block_i$ считается градиент $\vec{g}_{kj}$. Дальше к направлению градиентов применяется квантование: все углы делятся на несколько секторов $Sect_n$. Для каждого сектора мы можем посчитать сколько пикселей в блоке имеют подходящее направление градиента $Angle(\vec{g}_{kj}) \in Sect_n$. В самом простом случае дескриптором блока является число пикселей блока, входящих в каждый сектор, а дескриптор всего изображения – конкатенация дескприпторов каждого блока. Чаще всего вклад пикселя в гистограмму блока делается пропорциональным норме градиента. Таким образом, мы фактически считаем силу градиента в заданом направлении в данном фрагменте изображения. Иногда вклад пикселя дополнительно шкалируют пропорционально удаленности от центра, что делает дескриптор более робастным по отношению к небольшим движениям изображения.

Однако, у такого дескриптора есть ряд недостатков. Применительно к нашей задаче можно выделить такой: дескриптор сильно меняется при изменении масштаба. Если мы хотим найти объект, изображенный на картинке-запросе, то ясно, что на других изображених он может быть разных размеров, не говоря уже о том, что одно и тоже изображение можно записать в разных разрешениях, поэтому необходимо учитывать градиенты, полученные на разных масштабах.

Чтобы решить эту проблему был придуман дескриптор GIST. Он похож на HOG, но в нем градиенты оцениваются на нескольких масштабах. Мы либо строим пирамиду разрешений, либо применяем фильтры поиска градиентов с разной степенью сглаживания. Для каждого масштаба строится своя гистограмма и в итоге для каждого блока получаем набор гистограмм для разных масштабов. К этому добавляют цветовую информацию – либо гистограмму цветов всего изображения, либо усредненный цвет каждого блока. Их можно по-разному взвешивать, увеличивая или уменьшая влияние цвета изображения.

Такой дескриптор хорошо зарекомендовал себя для поиска изображения по визуальному подобию, особенно для поиска “полудубликатов”, то есть слегка изменных версий картиноки, которые немного отличаются от заданой ракурсом, цветом, масштабом и тому подобными. Его можно считать по достаточно маленьким изображениям и он достаточно быстро вычисляется. Типичный размер дескриптора: 8 секторов для углов, 4 масштаба, 16 блоков, то есть все изображение описывается набором из 512 параметров.

3 Приближенный метод k-средних

Даже имея под рукой хороший дескриптор, перебирать всю коллекцию изображений каждый раз, чтобы найти похожее, очень дорого. Первый шаг, который нужно сделать, чтобы этого избежать, это квантование

векторов, то есть разбиение всей коллекции на группы похожих изображений.

Один из вариантов квантования – кластеризация на K кластеров, скажем, с помощью метода k-средних. Каждому объекту сопоставим битовый номер размером $log_2 K$ и создадим инвертированный индекс: списки изображений, попавших в каждый кластер.

Для поиска по изображению в индексе мы сначала квантуем запрос, смотрим в какой кластер он попал, ранжируем изображения из базы, которые лежат в этом кластере, и выдаем лучшие результаты. Ранжировать можно по-разному, самый простой вариант - сортировать элементы списка по расстоянию между их дескрипторами и дескриптором изображения-запроса.

У метода k-средних есть проблема: с одной стороны хочется увеличить число кластеров, чтобы увеличить точность аппроксимации, но с другой стороны чем больше кластеров, тем дольше работает кластеризация, одна итерация работает за $O(NK)$, где N - число примеров в коллекции.

Существует модификация алгоритма k-средних, иерархический метод k-средних, который заключается в том, что мы сначала кластеризуем на небольшое число кластеров, а затем каждый кластер разбиваем еще раз, пока получим дерево нужной высоты. Однако, этот алгоритм сильно проигрывает в точности обычному методу k-средних.

Чтобы ускорить метод k-средних нам потребуется KD-дерево. Оно строится следующим образом:

- Выбираем координату, вдоль которой в данных самая большая дисперсия.
- Находим медиану по этой координате и делим пространство на два полупростанства.
- С каждой из полученных половин делаем тоже самое, пока каждый элемент не окажется один в своей ячейке

Глубина дерева, таким образом, логарифмическая. К сожалению, KD-дерево плохо работает с большими размерностями. Чтобы решить эту проблему можно координату для разбиения выбирать случайно из нескольких направлений с наибольшей дисперсией и в качестве порога выбирать не медиану, а случайное число в некотором интервале вокруг неё. При этом будем делать не одно дерево, а несколько.

На основе рандомизированных KD-деревьев был придуман алгоритм “приближенных” k-средних:

- Строим лес из, скажем, 8 рандомизированных KD-деревьев
- В стандартном методе k-средних на каждом этапе ищем ближайший центр кластера с помощью леса KD-деревьев

у такого алгоритма сложность одной итерации уже падает до $O(Nlog(K))$.

4 LSH хэширование

Еще одним способом квантования элементов является семантическое хэширование. Идея состоит в том, чтобы отображать исходные вектора дескрипторов в бинарные коды так, чтобы близкие по L2 дескрипторы обладали кодами близкими по расстоянию Хэмминга (число различных битов).

Наиболее известный метод семантического хэширования – Локальное Чувствительное Хэширование (LSH). Работает оно так:

- Случайно выбираем прямую и проектируем кодируемые вектора на неё
- Выбираем порог, близкий к медиане полученных координат на прямой
- Все элементы, попавшие в одну половину помечаем единичками, остальные нулями.
- Повторяем несколько раз

В итоге получим разбиение пространства на ячейки, каждая из которых получает свой уникальный битовый код. Можно показать, что, если число прямых стремится к бесконечности, битовое расстояние между кодами будет аппроксимировать исходное L2 расстояние.

Этот алгоритм довольно быстрый, но приближение лишь ассимптотическое и на практике, чтобы достичь точности метода k-средних, могут потребоваться коды большой длины. Поэтому как замену методу k-средних LSH использовать нельзя.

5 Алгоритм GISTIS

Еще раз опишем, в чем проблемы k-средних: слишком маленькое число кластеров K ведет к большим ячейкам и очень грубому сравнению, а слишком большое K ведет к медленной кластеризации и маленьким ячейкам, куда могут попадать не все похожие примеры. Хочется все же взять небольшое K и как-то повысить точность разбиения. Предлагается для каждого кластера закодировать попавшие в него вершины с помощью LSH. Причем кодировать не сам вектор дескриптора, а разность между ним и дескриптором центра кластера. Такой алгоритм называется Hamming Embedding.

Алгоритм, который строит Hamming Embedding над дескриптором GIST, называется GISTIS (GIST Indexing Structure). Таким образом, весь алгоритм GISTIS выглядит так:

- Строим GIST для каждого изображения
- Кластеризуем все дескрипторы с помощью приближенного метода k-средних
- Применяем LSH к каждому кластеру и получаем бинарный код для каждого вектора
- Код и номер кластера храним в RAM, а полыне вектора GIST записываем на диск и загружаем по необходимости

Процедура поиска выглядит следующим образом:

- Выбираем несколько ближайших кластеров к картинке-запросу
- Считаем бинарный код картинки-запроса в каждом из этих кластеров
- Фильтруем картинки из найденных кластеров по порогу на расстояние Хэмминга между ними и соответствующему коду картинки-запроса
- Возвращаем все оставшиеся картинки, отсортированные по расстоянию Хэмминга

В конце, можно самые ближайшие результаты дополнительно отсортировать по L2 расстоянию между дескрипторами GIST, а затем можно еще и дополнительно отсортировать, основываясь на сходстве особых точек (SIFT). Такой алгоритм уже достаточно хорошо показывает себя на практике.