Метрические методы

Конспект Яндекс учебника https://education.yandex.ru/handbook/ml

Общие черты

Алгоритмы этого класса практически не имеют этапа обучения, они запоминают обучающую выборку, а на этапе предсказания основываются на похожих на целевой объекты (lazy learning). Метрические модели непараметрические, так как они не делают глобальных допущений при своей работе, поэтому линейная регрессия (исходит из предположения что выборка линейна разделима), линейная бинарная классификация (исходит из того, что гиперплоскость которая логично разделяет выборку на две части существует) не являются метрическими методами. Метрические модели локальны - предполагают, что мы можем предсказать информацию об объекте имея данные о его соседях.

Эти модели могут быть полезны, в случае когда наши данные достаточно сложны, чтобы использовать какую-то другую модель, но абсолютны непременимы при большом объёме данных из-за lazy learning'а.

Метод k-ближайших соседей (KNN)

Из-за отсутствия обучения практически нигде не применяется в чистом виде, но много где используется в качестве вспомогательной модели.

KNN

Собственно, всё интуитивно понятно. Если записать это более формально:

$$a(u) = \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{k} \mathbb{I}[y_u^{(i)} = y]$$

KNN

Также можно считать “вероятность” принадлежности к классу:

$$\mathbb{P}(u \sim y) = \frac{\sum_{i=1}^{k} \mathbb{I}[y_u^{(i)} = y]}{k}$$

“вероятность” в KNN

“Вероятность” - потому что фактически, мы рассматриваем это как веротяность, потому что оно ≥ 0 и ≤1, но строго говоря это не вероятность.

Теперь возникают вопросы, на какое число ближайших соседей нам смотреть и как нам считать расстояние между объектами?

Выбор метрики

Самый популярный способ вычисления расстояния - евклидово.

$$\rho(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

евклилово расстояние

Однако для разных ситуация могут быть полезны и другие способы:

$$\rho(x, y) = \sum_i |x_i - y_i|$$

манхэттенское

Манхэттенская метрика полезна при наличии выбросов, так как из-за модуля, в отличие от квадрата в евклидовой метрике, выбросы не будут сильно влиять на предсказание.

$$\rho(x, y) = \left(\sum_i |x_i - y_i|^p \right)^{1/p}$$

минковского

Метрика Минковского - обобщение манхэттенской и евклидовой метрик.

$$\rho(x, y) = 1 - \cos\theta = 1 - \frac{x \cdot y}{|x||y|}$$

косинусное

Косинусное расстояние часто применяется там, где не важны нормы векторов. В частности, когда нам нужно определить схожесть документов, то мы будем обращать внимание на количество слов в них, но так как повторяемость одинаковых слов в тексте не должна сильно влиять на ответ, то часто применяют косинусное расстояние.

$$\rho(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Жаккара

Расстояние Жакара применяют там, где рассматривают выборку как набор множеств, в таком случае нет нужны приводить множества к векторам.

Все выше описанные функции не являются метриками в полном смысле этого слова, и строго говоря не являются гиперпараметром. Однако имеет смысл

перебрать, по какой формуле мы будем считать расстояния, чтоюы достигнуть лучшего результата.

Обобщения KNN

Взвешенный KNN

Можно заметить, что у оригинального KNN есть недостаток: он не считает, что более близкие соседи должны влиять на результат больше, чем дальние. Тогда введём веса, которые ранжируются в зависимости от дальности соседа от объекта предсказания.

$$a(u) = \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{k} w_i \mathbb{I}[y_u^{(i)} = y]$$

weighted KNN

Как же нам вычислить нововведённые веса? Есть два главных подхода:

Первый, давайте отсортируем соседей объекта по расстоянию до него, и в зависимости от номера соседа присвоим ему вес.

$$\text{линейно } \left(w_i = \frac{k+1-i}{k}\right)$$

$$\text{экспоненциально } \left(w_i = q^i,\ 0 < q < 1\right)$$

Второй подход, давайте будем использовать не номера соседей, а расстояния до них. Если подумать, какими свойствами должна обладать функция, то она должна быть всегда положительной, она не должна монотонно возрастать, ну и чтобы веса ближних соседей были больше весов дальних. В общем, придумали ядерную функцию (kernel function).

$$a(u) = \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{k} K\left(\frac{\rho(u, x_u^{(i)})}{h}\right) \mathbb{I}[y_u^{(i)} = y]$$

h ≥ 0 - ширина окна

$$a(u) = \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{k} K\left(\frac{\rho(u, x_u^{(i)})}{h}\right) \mathbb{I}[y_u^{(i)} = y]$$

kernel function

Далее в зависимости от задачи выбирают тип ядра(обычно прямоугольное или гауссовское) и ширину окна(эталонного выбора нет, если слишком маленькое, то обучения не будет, если слишком большое, то модель слишком простая).

Kernel Regression

Всё, что было описано ранее решало задачу классификации, теперь попробуем решить задачу регрессии. Два способа которые сразу напрашиваются: брать среднее или среднее взвешенное.

$$a(u) = \frac{1}{k} \sum_{i=1}^{k} y_u^{(i)}$$

$$a(u) = \frac{\sum_{i=1}^{k} K\left(\frac{\rho(u, x_u^{(i)})}{h}\right) y_u^{(i)}}{\sum_{i=1}^{k} K\left(\frac{\rho(u, x_u^{(i)})}{h}\right)}$$

формула Надарая-Ватсона

Хорошо, а можем ли мы применить формулы из классификации? Так как теперь у нас не ограниченное число классов, а всевозможные числа, то сравнивать эти значения не имеет смысла, так как они практически никогда не будут равны. Значит нам нужно заменить индикатор сравнения из классификации на более гладкую функцию, возьмём квадрат евклидова расстояния. Минимизируя расстояние, мы будем максимизировать близость:

$$a(u) = \arg\min_{y \in \mathbb{R}} \sum_{i=1}^{k} K\left(\frac{\rho(u, x_u^{(i)})}{h}\right)(y - y_u^{(i)})^2$$

KNN для регрессии

Преимущества

1) Непараметрический, так как не делает явных предположений о нашей выборке.

2) Простой в объяснении и интерпретации

3) Достаточно точный, хоть и зачастую уступает другим моделям

4) Используется как для классификации, так и для регрессии

Недостатки

1) Хранит всю выборку, поэтому требует много памяти

2) Вычислительно дорогой из-за причины №1

3) Плохо реагирует на масштабность данных и на неинформативные фичи

4) Для применения алгоритма необходимо, чтобы метрическая близость объектов совпадала со смысловой. С этой проблемой может помочь использование представлений (организовываем данные таким образом, чтобы обозначить смысловую связь объектов между собой).

Применение

1) Рекомендательные системы: “предложить пользователю что-то похожее на то, что он любит”.

2) Поиск похожих документов

3) Поиск аномалий и выбросов

4) Кредитный скоринг

Сложность алгоритма

Поиск ближайших соседей

Если в лоб искать соседей и считать расстояние до объекта, то мы это сделаем за количество объектов * количество признаков, что крайне нежелательно, так как оба этих числа как правило большие. Есть две группы методов поиска ближайших соседей: точные и приближённые. Приближённые методы поиска соседей применяют не только в KNN, но и в системах поиска.

Точные методы поиска ближайших соседей

Существует два метода:

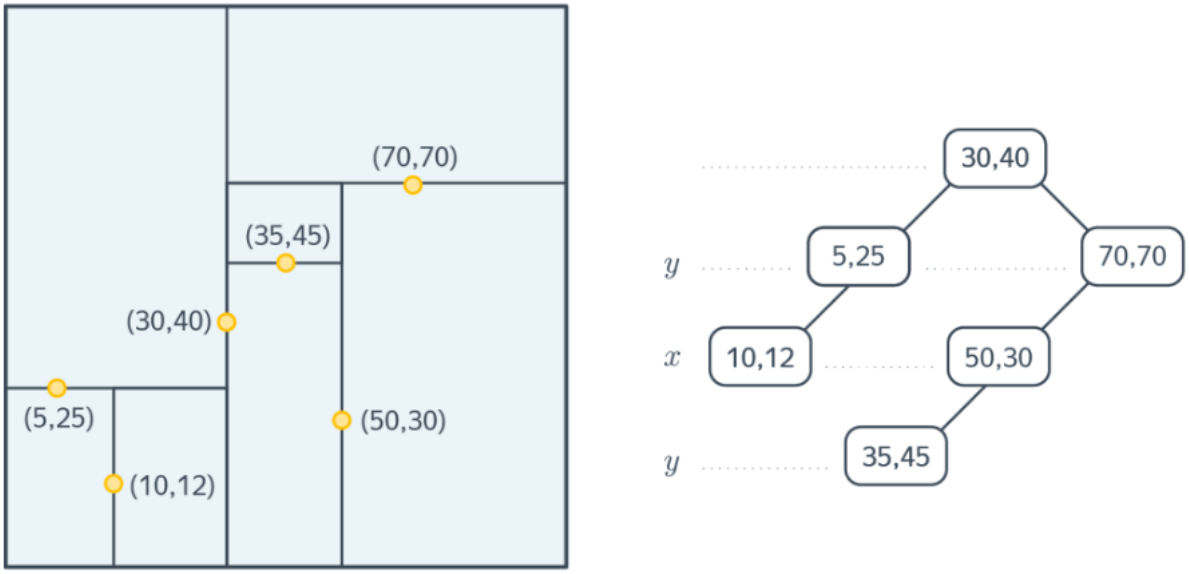
Первый, мы принебрегаем частью фич, тем самым мы оценим расстояния от соседей до объекта снизу, поэтому если так получилось, что расстояние до какого-то соседа оказалось больше, чем до похожего соседа, то можно сразу этот объект не учитывать. Это даёт выигрыш по времени, но не меняет ассимптотику.

Второй - k-d деревья.

k-d деревья

Допустим, у наших объектов всего лишь одна фича. Тогда для поиска ближайшего объекта можно применить бинарное дерево поиска, тем самым за логарифм находив объекты. В многомерном пространстве есть подобная структура - k-d дерево (k-d tree, k-dimensional tree).

Трудность состоит в том, что мы не можем сравнить два вектора также, как два числа.



двумерное дерево

В случае двумерного дерева поиск происходит следующим образом: выбирается первоначальная ось(фича), например корень будет отвечать за деление по x-координате, дети по y, внуки по x, и т.д. Посмотря на картинку возникает вопрос, как выбирать векторы, которые будут находиться в узлах дерева? Понятно, что нам нужно, чтобы дерево было сбалансированным, а значит нужно брать медиану соответствущей оси соотвествующего подуровня. Но на практике медиану берут очень холатно (просто случайная точка или медиана некоторого подмножества), чтобы ускорить построение дерева, хотя само дерево уже не будет очень сбалансированным. Добавляем вершины, соотвественно просто спускаясь вниз по дереву. Существуют варианты k-d деревьев, которые остаются сбалансированными при любых добавлениях/удалениях. В случаев деревьев ассимптотика получается

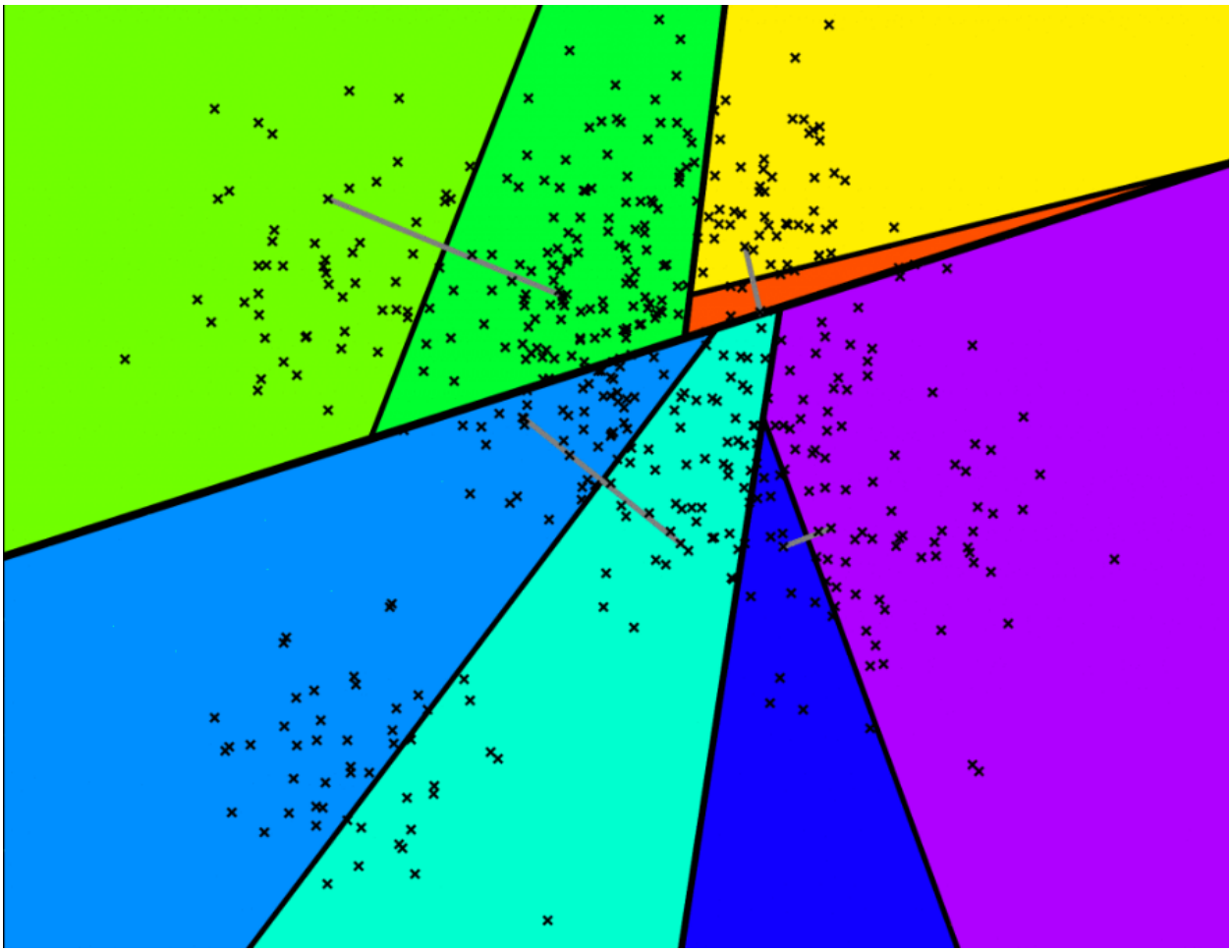
логарифмической, хотя число объектов и фич может повлиять настолько, что ассмиптотика будет близка к полному перебору.

Приближённые методы поиска ближайших соседей

Random projection trees

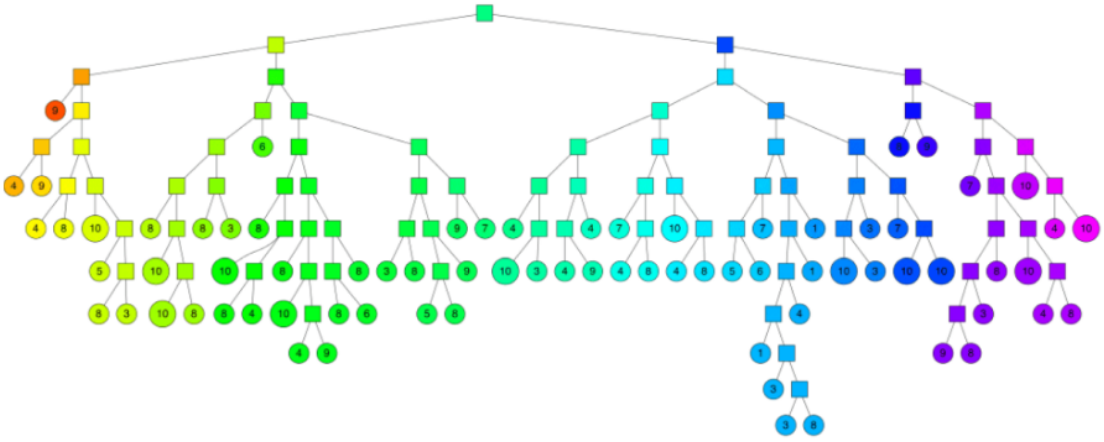
Идея, как и во всех алгоритмах применения деревьев для поиска соседей, разделить нашу выборку случайными гиперплоскостями, на основе этого разделения построить дерево, в листах которого будет малое число объектов.

Annoy - алгоритм, который применяет Spotify для рекомендаций. При его работе рекурсивно и случайно выбираются два объекта и проводится гиперплоскость симметрично их разделяющая до тех пор, пока в каждой области будет не более M (гиперпараметр) объектов.



Annoy

Таким образом задаётся бинарное дерево с глубиной порядка $O(\log N)$ в среднем.



Спускаясь по этому дереву, мы находим в какой области находится целевой объект и некоторое количество близких к нему элементов, но не факт, что это будут ближайшие объекты, поэтому запускается лес таких деревьев и берётся объединение (чем больше деревьев тем точнее, но медленнее).

Из плюсов оптимальное соотношение между скоростью и точностью с помощью грамотного подбора гиперпараметров. Из минусов его нельзя некомпозировать, то есть не делится на батчи, и его не запустишь параллельно на несеольких GPU.

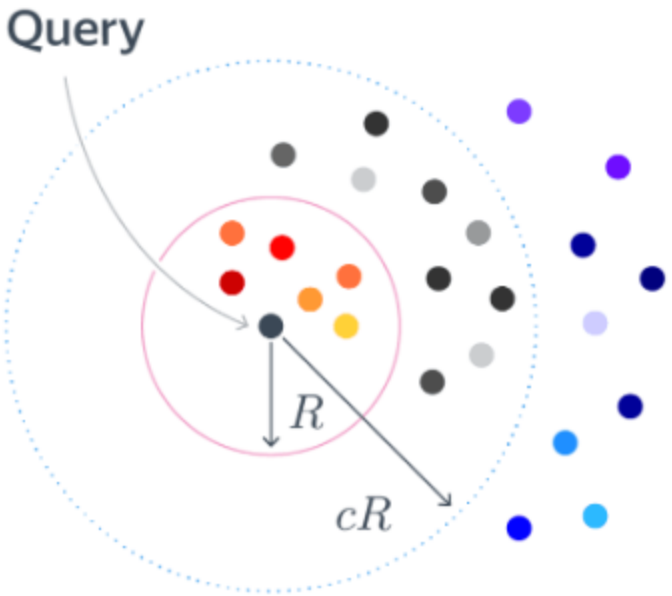
Locality-sensitive cashing (LSH)

Стоит задача - похожие объекты положить в один бакет (корзину, набор). Тогда будем вставлять наши объекты в хеш-таблицу, тогда посмотрев на коллизии нашей таблицы, мы найдём эти категории. Семейство таких алгоритмов называется locality-sensitive hashing (LSH).

Каким требованиям должна удовлетворять хеш-таблица? Чтобы вероятность коллизии на похожих объектах было высокой, а на разных низкой. Пусть высокая вероятность - p1, низкая вероятность - p2. Тогда формально наши критерии можно записать таким образом:

$$\text{для } \rho(x,y) < R \text{ вероятность коллизии } \Pr\left[h(x) = h(y)\right] > p_1;$$
$$\text{для } \rho(x,y) > cR \text{ вероятность коллизии } \Pr\left[h(x) = h(y)\right] < p_2.$$

А графически это можно преставить вот так:



В зависимости от выбранной функции расстояния выбирается своя хеш-функция.

Как правило, если использовать одну хеш-функцию, то p1 и p2 становятся близкими друг к другу, чтобы этого избежать можно использовать композицию хеш-функций, также использовать несколько хеш-функций и искать соседей среди коллизий всех хеш-таблиц.

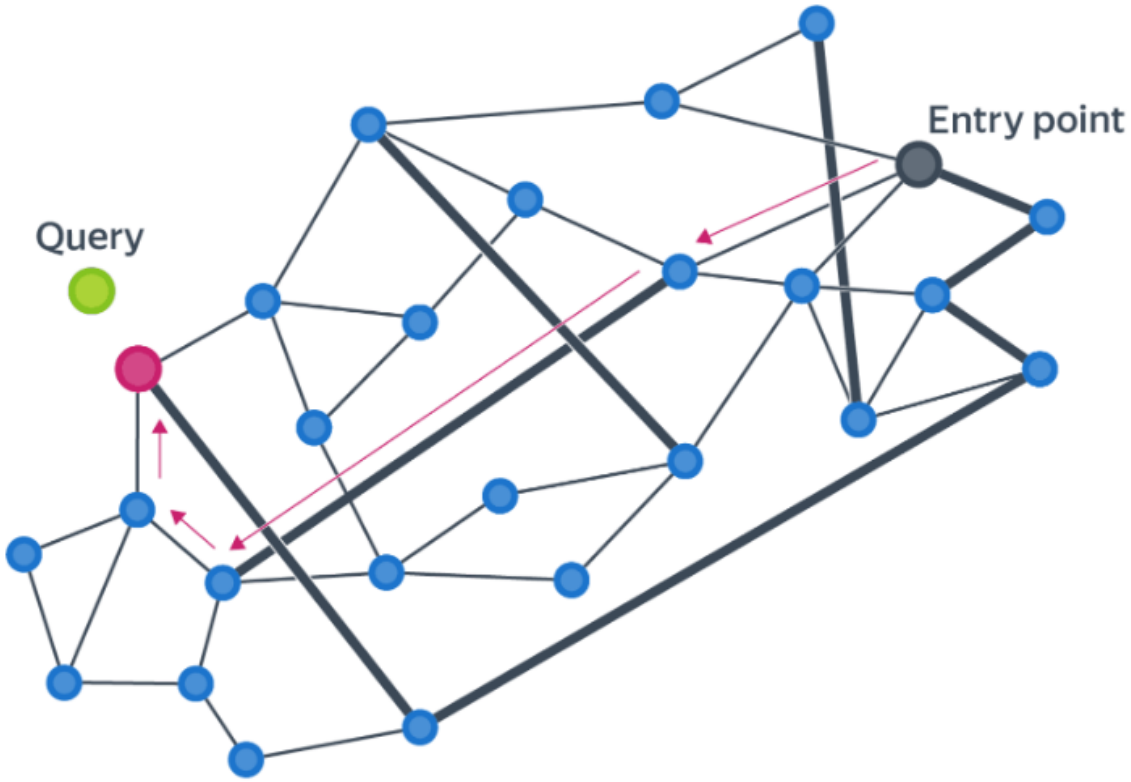
Из плюсов такого подхода хорошие гарантии сублинейной ассимптотики и точности.

Из минусов требует много памяти и плохо адаптируется под GPU.

Proximirity graphs & Hierarchical navigable small world (HNSW)

Это семейство алгоритмов, основанное на построении графа близости (proximity graph) на объектах выборки и последующему жадному поиску по этому графу. Самый популярный из них Navigable small world (NSW). Его суть: на выборке строится граф, между любыми двумя вершинами которого существует короткий путь и средняя степень вершины мала. В таком графе легко выполняется поиск соседа: выбирается случайная точка, дальше переходим к той точке, которая имеет ребро со случайно выбранной и ближе

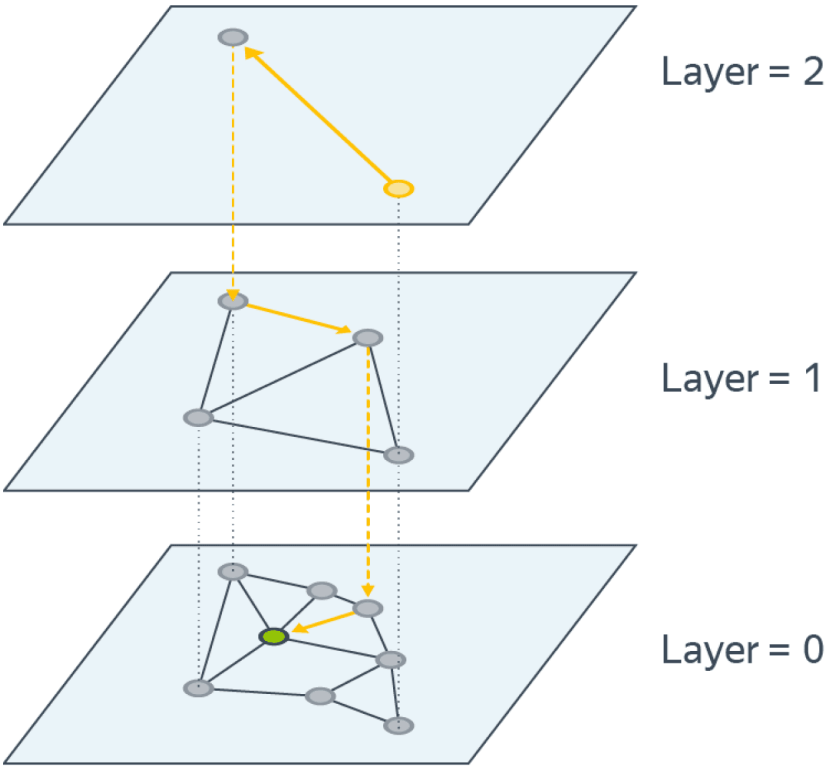
всего к искомой, таким образом за полилогарифмическое время выполняется поиск.



NSW

Есть небольшой нюанс, если entry point расположился в плотном кластере, то мы потратим много переходов, чтобы из этого кластера выбраться, в таком случае можно ввести иерархию, получим HNSW.

Первоначальный граф - нулевой слой. Чтобы получить следующий слой, мы берём предыдущий и переносим каждую его вершину в новый слой с вероятностью p, а затем мы из просочившихся вершин снова строим NSW. Всего сделаем log(N) слоёв.



HNSW

Поиск будет устроен так: начинаем с верхнего слоя(последнего построенного), находим ближайшую к искомой вершину, спускаемся на слой ниже, находим из предыдуше наденной вершины её ближайшего к искомой вершине соседа и т. д. Ускорение происходит из-за того, что в верхних слоях не будет плотных кластеров, а при переходе к нижним мы в них не попадём.

На данный момент этот метод является стандартом, хоть и требует много памяти(хранение вершин и связей) и требует перестройки в случае добавления новых объектов.

Уточнение

Все вышеперечисленные методы не являются универсальными, всё зависит от нашей задачи. А из-за того, что многие из них плохо переносятся на GPU, то на практике зачастую используют просто полный перебор.