KNN

Метод K-ближайших соседей и его усовершенствование

Рословец Владислав
27.09.19

*Также есть метрики Манхэттена и Минковского

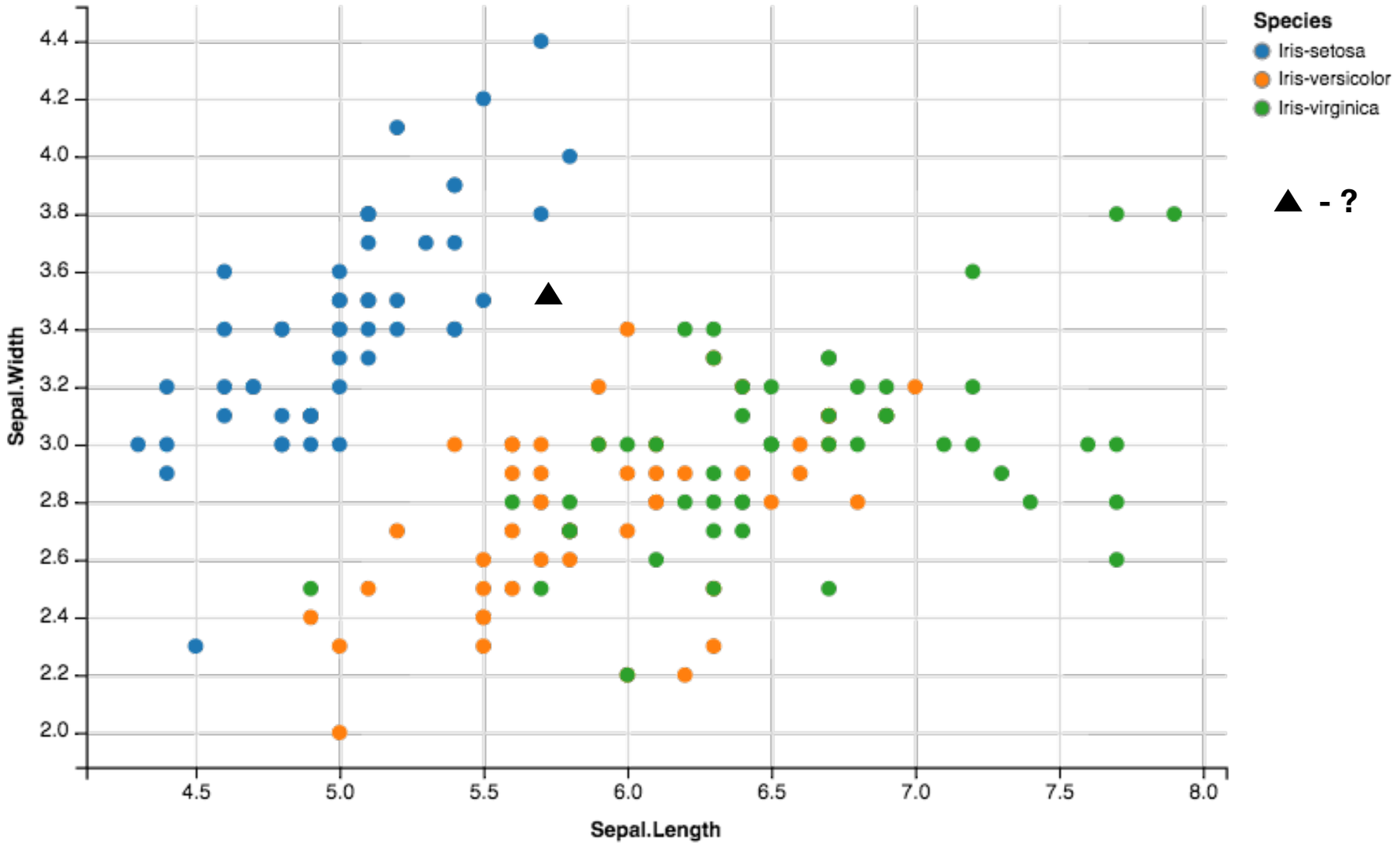
Постановка задачи

Имеется некоторый набор данных разделенный на 2 или более классов. При помощи метрики расстояния мы хотим уметь разделять данные и правильно классифицировать новые объекты.

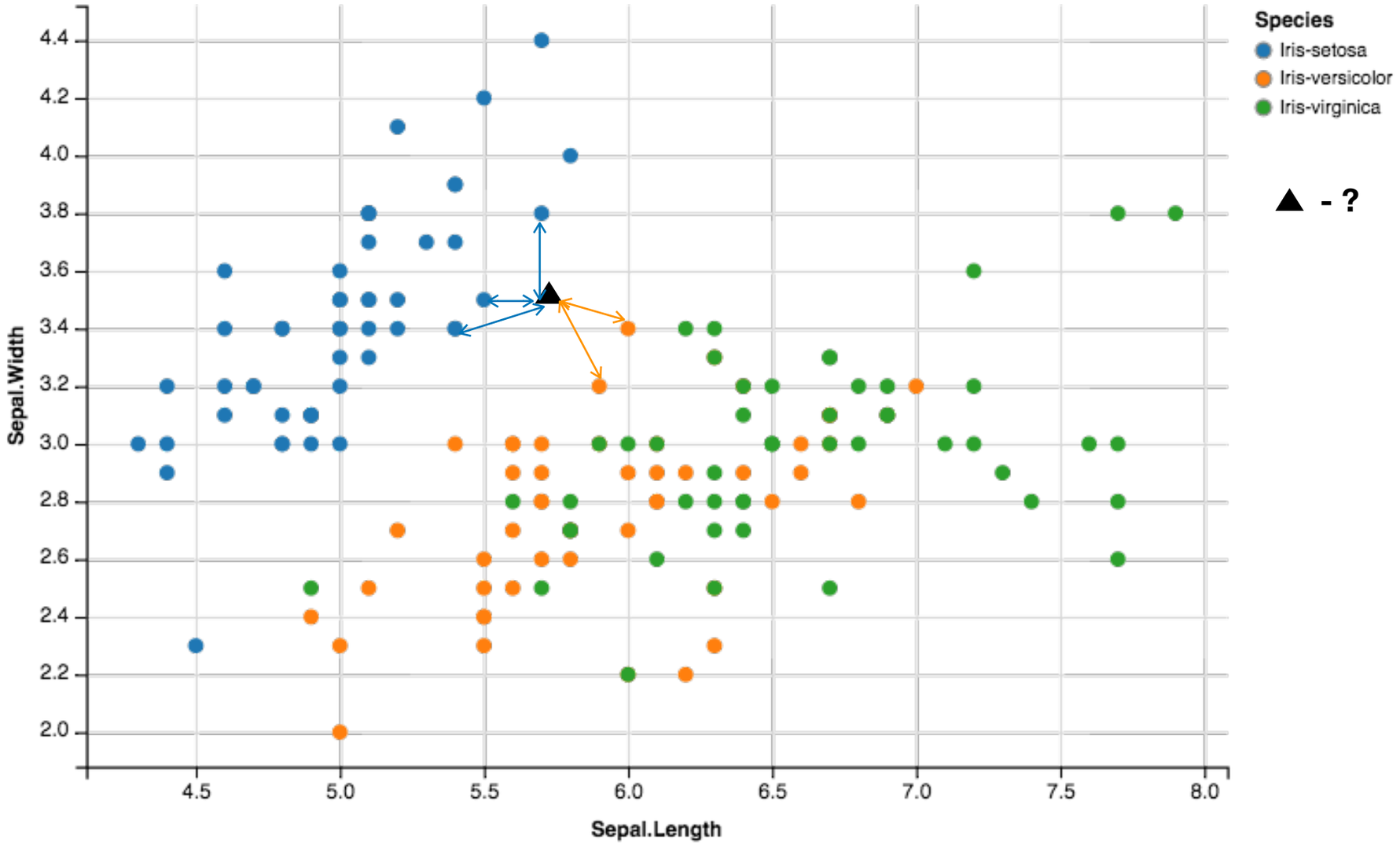
Далее, в качестве расстояния используем Евклидово расстояние:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots (x_l - y_l)^2} = \sqrt{\sum_{k=1}^{l} (x_k - y_k)^2}$$

Почему K-ближайших соседей?



Solution


Species
Iris-setosa
Iris-versicolor
Iris-virginica
▲ - ?
Sepal.Width
Sepal.Length
4.4
4.2
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
2.2
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

Формализация задачи

Посчитаем расстояние нового объекта до каждого другого из набора данных и отсортируем по возрастанию.

$y_x^{(i)}$ - Класс *i*-ого соседа

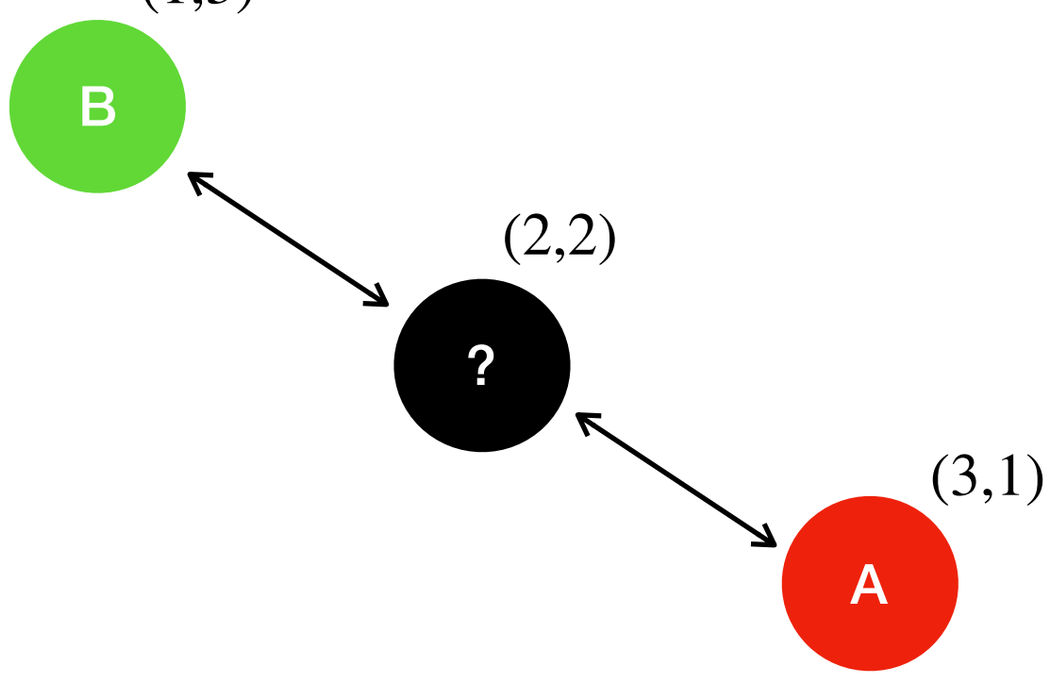
k - Кол-во соседей

x - Вектор признаков

y - Определенный класс

$$a(x) = argmax \sum_{i=1}^{k} [y_x^{(i)} = y]$$

Проблема одинаковых расстояний



Способ улучшить KNN

Формула обобщенного метрического классификатора

$$a(x) = argmax \sum_{i=1}^{k} [y_x^{(i)} = y] \cdot w(i, x)$$

$w(i, x)$ - вес *i*-ого соседа для *x*

Метод парзеновского окна

Обобщим knn с помощью функции ядра(kernel)

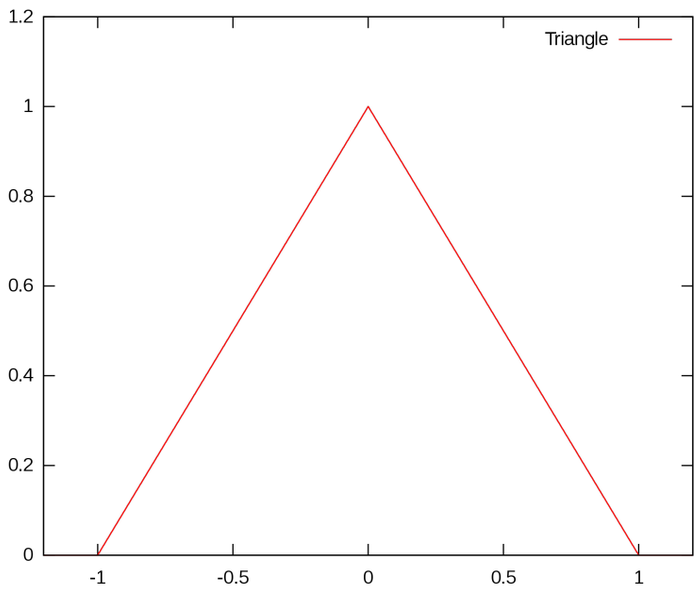
$$w(i,x) = K\left(\frac{\rho(x,u_i)}{h}\right)$$ Фиксированная длина h

$$w(i,x) = K\left(\frac{\rho(x,u_i)}{\rho(x,u_{k+1})}\right)$$ Переменной длины

Примеры ядер

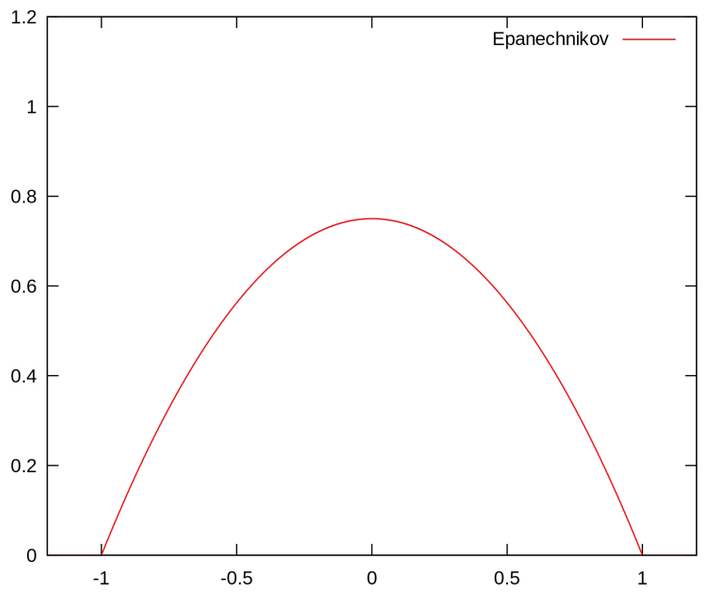
Triangular:

$$K(r) = (1 - |r|)$$



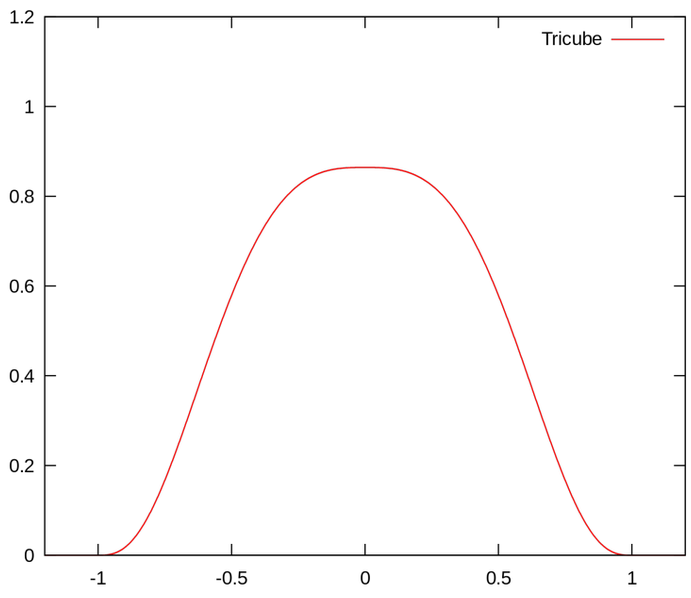
Parabolic:

$$K(r) = \frac{3}{4}(1 - r^2)$$



Tricube:

$$K(r) = \frac{70}{81}(1 - |r|^3)^3$$



Скользящий контроль

Функция скользящего контроля(leave-one-out)

$$LOO(k, X^l) = \sum_{i=1}^{l} \left[a(x_i; X^l \backslash \{x_i\}, k) \neq y_i \right] \rightarrow min$$

Оптимальное число соседей будет при минимальной ошибке

Преимущества:

1) Простота реализации

Недостатки:

1) Низкое качество классификации
2) Хранение всей выборки в памяти
3) Асимптотика $O(n \cdot m \cdot k)$

NSW и HNSW

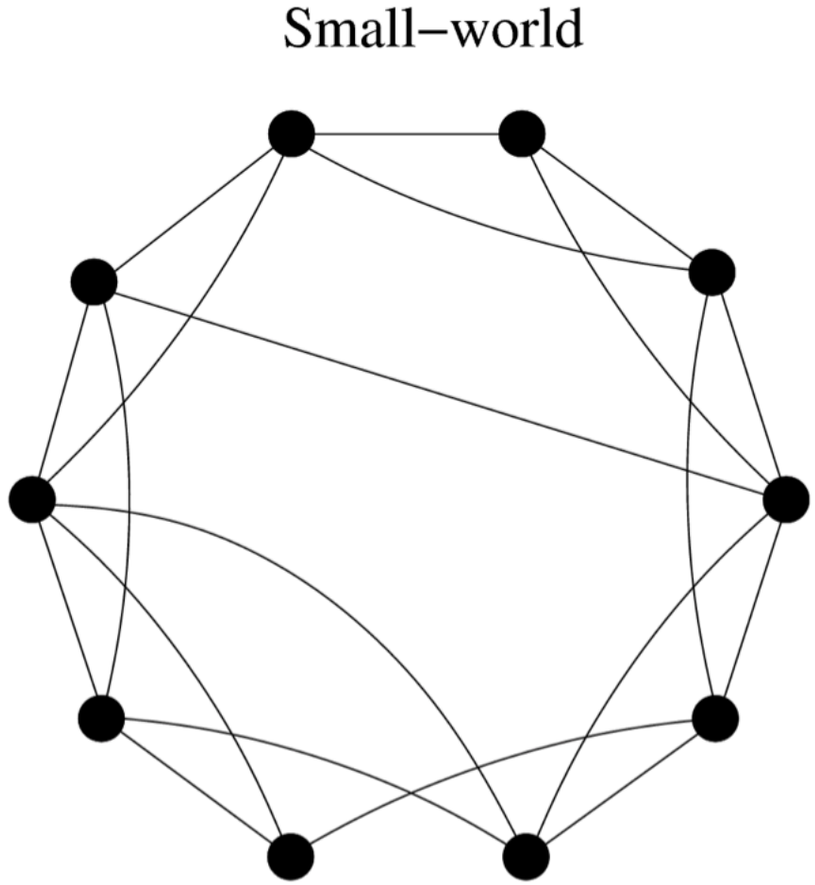
Методы приближенного поиска ближайших соседей

Small world graph

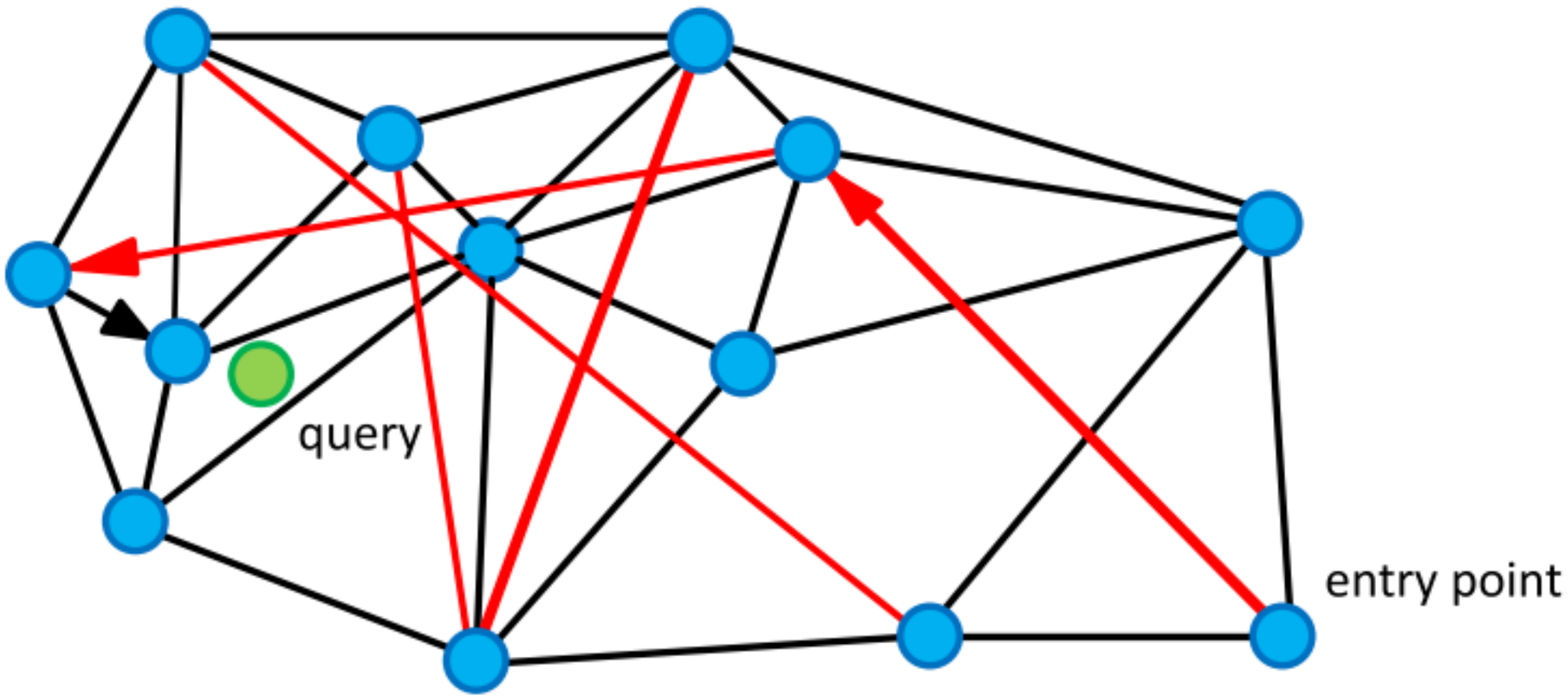
Свойства:

1. Малая длина кратчайшего пути в среднем
2. Большой коэффициент кластеризации
3. Расстояние между двумя случайными вершинами примерно $logN$
4. Пара вершин с большой вероятностью не смежны

Пример построения SW



Алгоритм NSW



Псевдокод:

```
K-NNSearch(vector target, int m, int k):
  TreeSet tempRes, candidates, visitedSet, result
  for (i = 0; i < m; i++):
     put random entry point in candidates
     tempRes = nil
     repeat:
       get element c closest from candidates to target
       remove c from candidates
       #check stop condition:
       if c is further than k-th element from result
          than break repeat
     #update list of candidates:
     for every element e from friends of c:
        if e is not in visitedSet than
           add e to visitedSet, candidates, tempRes

     End
     #aggregate the results:
     add objects from tempRes to result
  end
  return best k elements from result
```

Реальное применение

./fasttext nn model

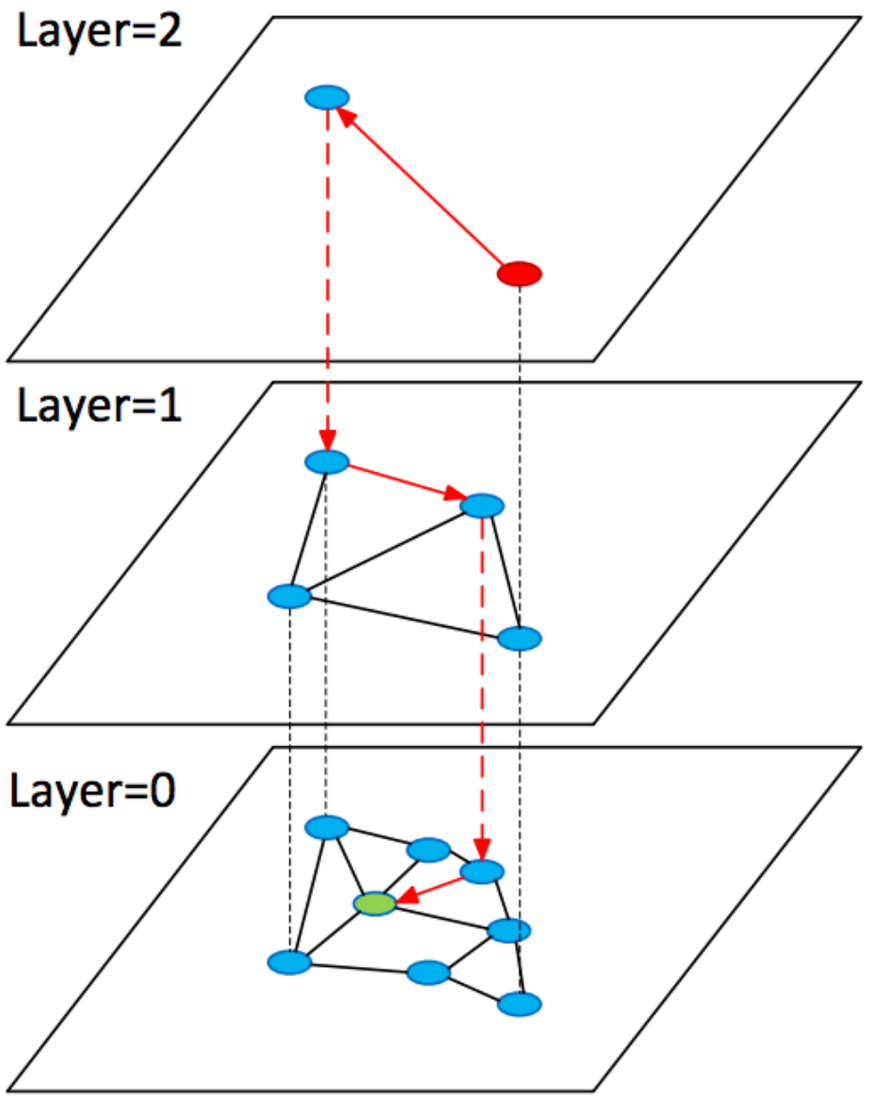
Query word? масло
биомасло 0.814148
маслище 0.801789
экомасло 0.785253
маслоars 0.75494
аромамасло 0.746124
масл 0.741603
мыло 0.715547
молочко 0.711884
масла 0.706256
аромомасло 0.699966

```
std::vector<std::pair<real, std::string>> FastText::getNN(
    const DenseMatrix& wordVectors,
    const Vector& query,
    int32_t k,
    const std::set<std::string>& banSet) {
  std::vector<std::pair<real, std::string>> heap;

  real queryNorm = query.norm();
  if (std::abs(queryNorm) < 1e-8) {
    queryNorm = 1;
  }

  for (int32_t i = 0; i < dict_->nwords(); i++) {
    std::string word = dict_->getWord(i);
    if (banSet.find(word) == banSet.end()) {
      real dp = wordVectors.dotRow(query, i);
      real similarity = dp / queryNorm;

      if (heap.size() == k && similarity < heap.front().first) {
        continue;
      }
      heap.push_back(std::make_pair(similarity, word));
      std::push_heap(heap.begin(), heap.end(), comparePairs);
      if (heap.size() > k) {
        std::pop_heap(heap.begin(), heap.end(), comparePairs);
        heap.pop_back();
      }
    }
  }
  std::sort_heap(heap.begin(), heap.end(), comparePairs);

  return heap;
}
```

HNSW

Все тоже самое, только быстрее

Пример:



Источники

1) https://m.habr.com/ru/company/mailru/blog/338360/
2) https://ru.wikipedia.org/wiki/Метод_k-ближайших_соседей
3) http://neerc.ifmo.ru/wiki/
4) https://www.youtube.com/watch?v=l1xGQMowWA4

Вопросы

1) Если вы решили использовать в качестве метрики расстояния что-то более специфичное, чем евклидово расстояние, какие аксиомы должны быть выполнены для такой метрики?
2) Написать и объяснить асимптотику knn(память + время)
3) Почему nsw быстрый?
4) Необязательный вопрос: асимптотика построения small world графа.
5) Формула скользящего контроля.(формула подбора оптимального числа соседей)