Approximate nearest neighbor search using the Hierarchical Navigable Small World (HNSW) algorithm

Sebastian Björkqvist

Lead AI Developer, IPRally

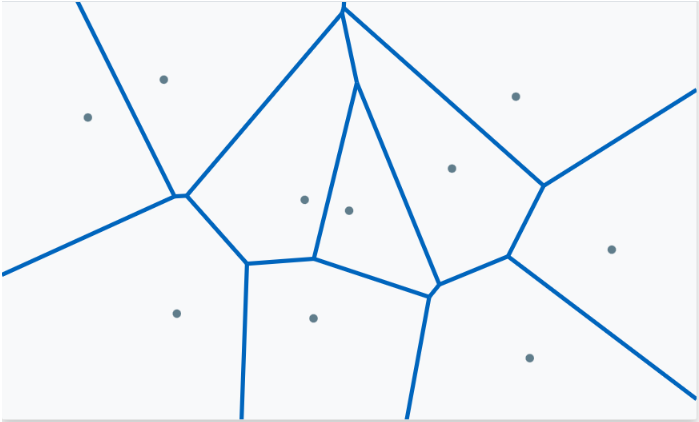
May 12, 2023

Outline

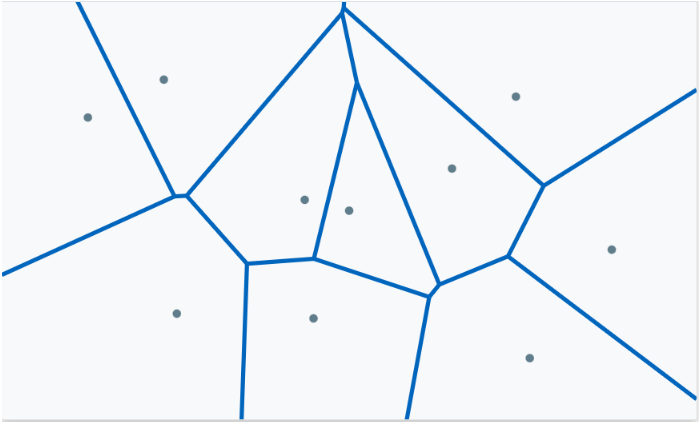
1. Theoretical foundations
   - Voronoi diagram
   - Delaunay graph
   - Greedy NN search using Delaunay graph
2. HNSW algorithm
   - Idea behind algorithm
   - Construction of search index
   - Nearest neighbor search using index
3. Performance
   - Search accuracy
   - Build time

Voronoi diagram for a set of points

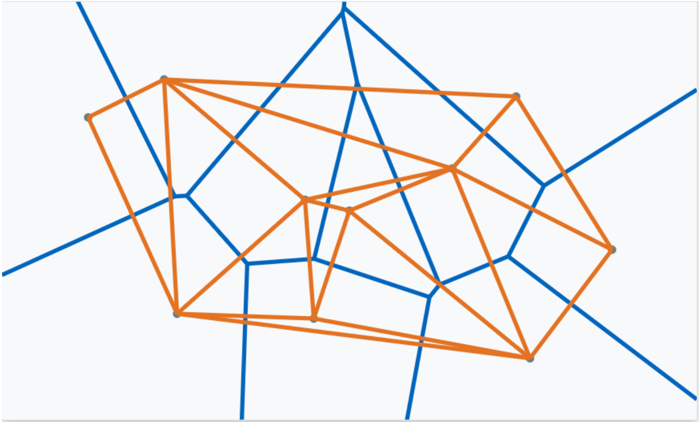
Voronoi diagram for a set of points



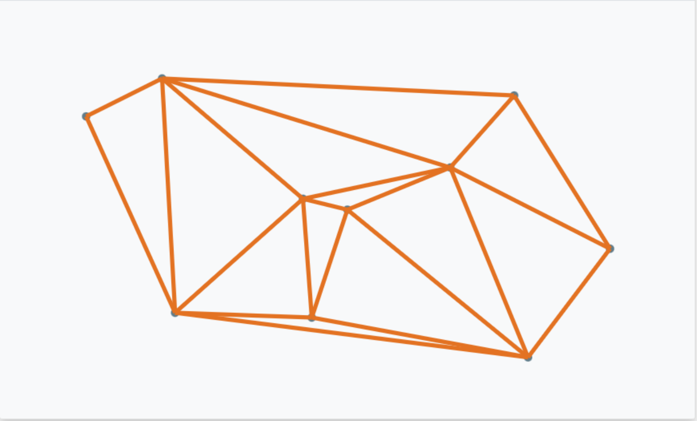
Voronoi diagram to Delaunay graph



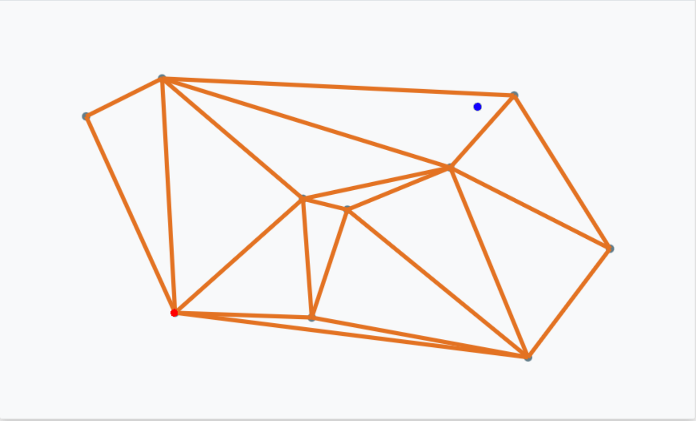
Voronoi diagram to Delaunay graph



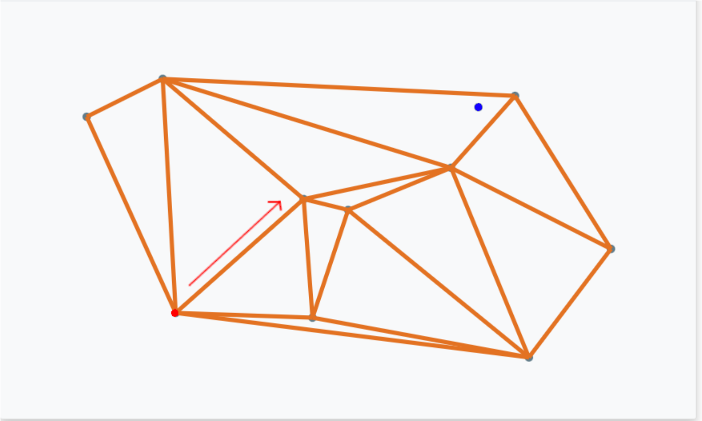
Delaunay graph



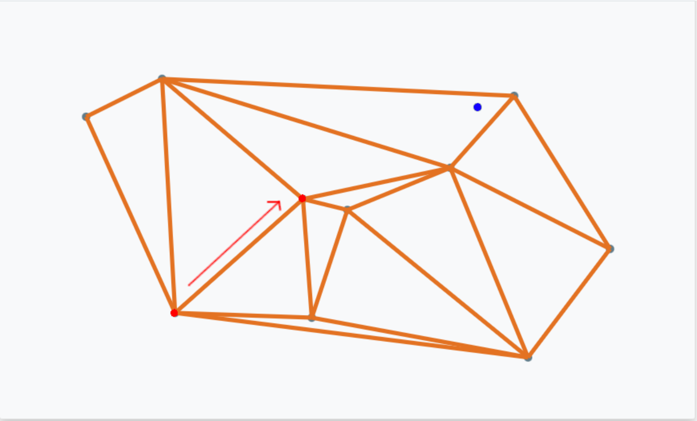
Greedy NN search start - Query and entry point



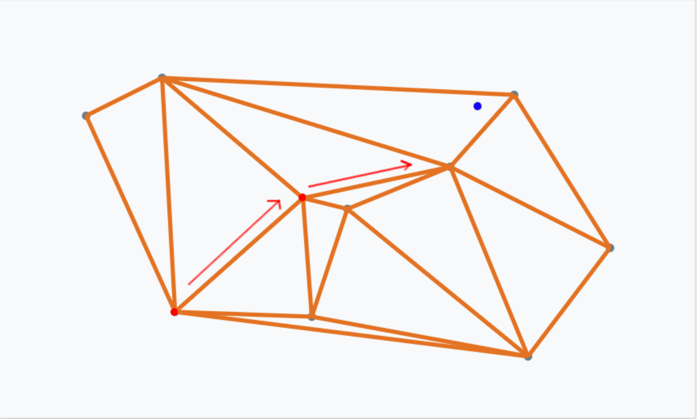
Greedy NN search - iteration



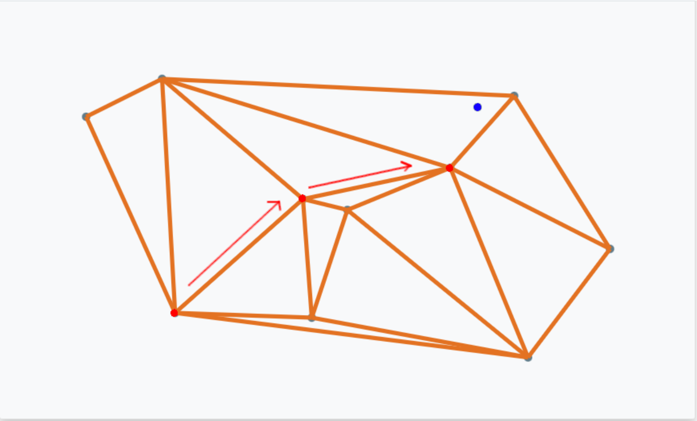
Greedy NN search - iteration



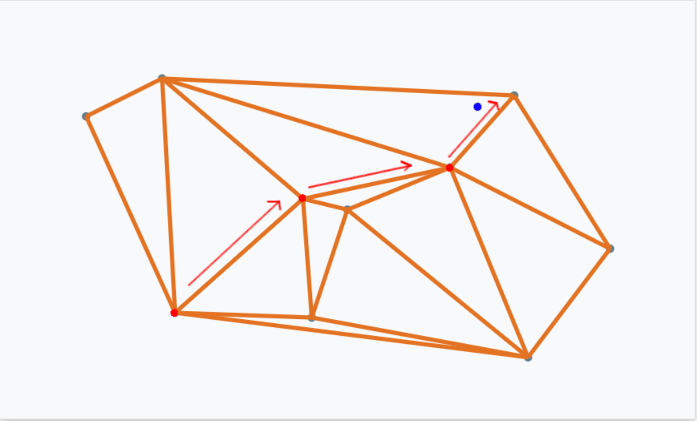
Greedy NN search - iteration



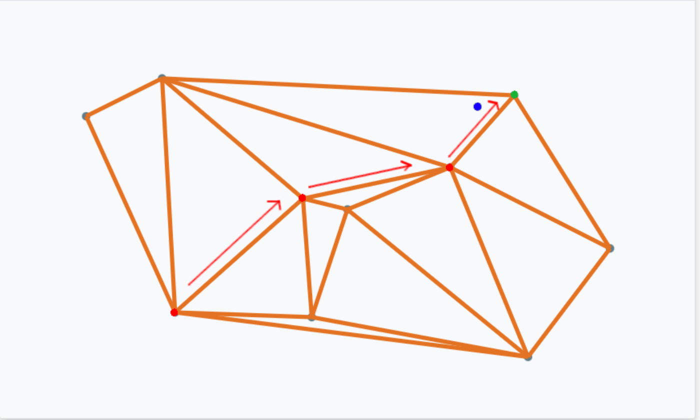
Greedy NN search - iteration



Greedy NN search - iteration



Greedy NN search done!



Drawbacks

- Delaunay graph intractable to construct for large, high-dimensional data sets
- Greedy search might be slow if graph is large

Navigable small world (NSW) graph

Navigable small world (NSW) graph

- Small world graph

Navigable small world (NSW) graph

- Small world graph
    - Distance of two random nodes is $log\ N$, where N is the number of nodes in graph

Navigable small world (NSW) graph

- Small world graph
    - Distance of two random nodes is $log\ N$, where N is the number of nodes in graph
    - Neighbors of a given node are likely to be neighbors of another (clustering coefficient is high)

Navigable small world (NSW) graph

- Small world graph
  - Distance of two random nodes is $log\ N$, where N is the number of nodes in graph
  - Neighbors of a given node are likely to be neighbors of another (clustering coefficient is high)
- Navigability

Navigable small world (NSW) graph

- Small world graph
  - Distance of two random nodes is $log\ N$, where N is the number of nodes in graph
  - Neighbors of a given node are likely to be neighbors of another (clustering coefficient is high)
- Navigability
  - Greedy search algorithm has logarithmic scalability

Why is an NSW useful for nearest neighbor search?

Why is an NSW useful for nearest neighbor search?

- Logarithmic distance allows us to get anywhere in the graph quickly

Why is an NSW useful for nearest neighbor search?

- Logarithmic distance allows us to get anywhere in the graph quickly
- Navigability ensures that the greedy algorithm finds the logaritmic path

Why is an NSW useful for nearest neighbor search?

- Logarithmic distance allows us to get anywhere in the graph quickly
- Navigability ensures that the greedy algorithm finds the logaritmic path
- High clustering coefficient lets us zoom in on the actual correct node when we're in the right area

Properties of NSW graph

- An NSW graph is not necessarily a Delaunay graph (or have one as a subgraph)

Properties of NSW graph

- An NSW graph is not necessarily a Delaunay graph (or have one as a subgraph)
- Thus the greedy algorithm doesn't always return the actual nearest neighbor

Properties of NSW graph

- An NSW graph is not necessarily a Delaunay graph (or have one as a subgraph)
- Thus the greedy algorithm doesn't always return the actual nearest neighbor
- Ok since we're doing approximate nearest neighbor search!

Constructing NSW graph

Constructing NSW graph

- Approximation of graph is enough (since we're doing approximate nearest neighbor search)

Constructing NSW graph

- Approximation of graph is enough (since we're doing approximate nearest neighbor search)
- Navigability: Greedy search algorithm has logarithmic scalability

How?

- We can learn a distribution for a document instead of just a single vector
- Model prior art relation as KL divergence of distributions

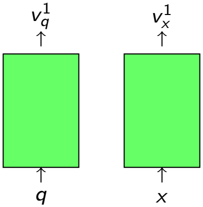
Distance functions for metadata

Why?

- We can do soft filtering (by country, patent class etc.)
- Can be useful if match is not found by strict filters

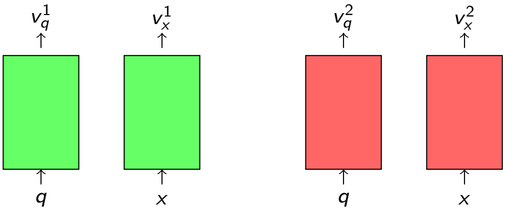
Learning multiple distance functions - naive way

$$d_1(q, x) = v_q^1 \cdot v_x^1$$



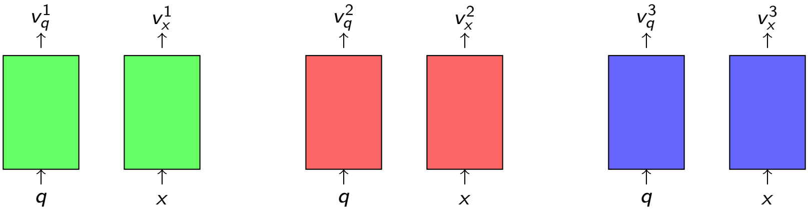
Learning multiple distance functions - naive way

$d_1(q,x) = v_q^1 \cdot v_x^1$ $d_2(q,x) = v_q^2 \cdot v_x^2$



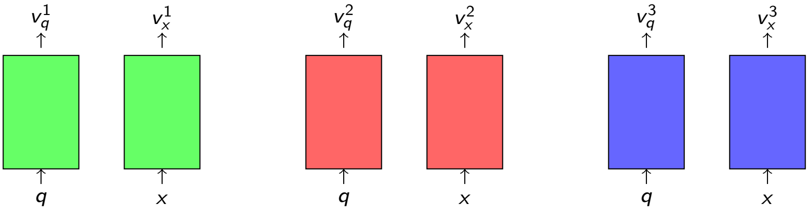
Learning multiple distance functions - naive way

$d_1(q,x) = v_q^1 \cdot v_x^1$ $d_2(q,x) = v_q^2 \cdot v_x^2$ $d_3(q,x) = v_q^3 \cdot v_x^3$



Learning multiple distance functions - naive way

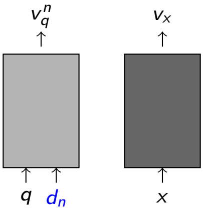
$d_1(q, x) = v_q^1 \cdot v_x^1$ $d_2(q, x) = v_q^2 \cdot v_x^2$ $d_3(q, x) = v_q^3 \cdot v_x^3$



Drawback: multiple embeddings of same document must be indexed!

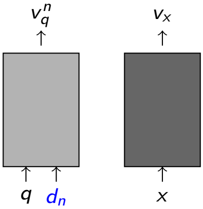
Learning multiple distance functions - efficient way

$$d_n(q, x) = v_q^n \cdot v_x$$



Learning multiple distance functions - efficient way

$$d_n(q, x) = v_q^n \cdot v_x$$



Only one meta-embedding per document is indexed!

Forward thinking

Forward thinking

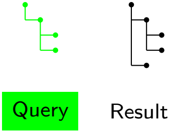
Why?

- We can train deeper models but keep batch size the same
- Training of deep models can take less wall clock time

Forward thinking - paper

- Forward Thinking: Building and Training Neural Networks One Layer at a Time (Hettinger et al.) `https://arxiv.org/abs/1706.02480`

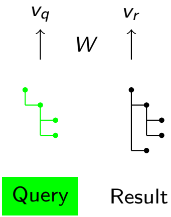
Current method - using gradients



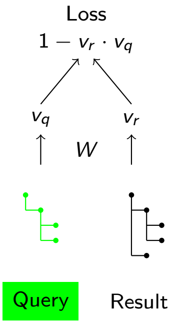
Query

Result

Current method - using gradients



Current method - using gradients



Loss

$1 - v_r \cdot v_q$

v_q

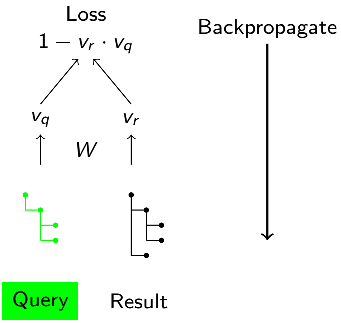
v_r

W

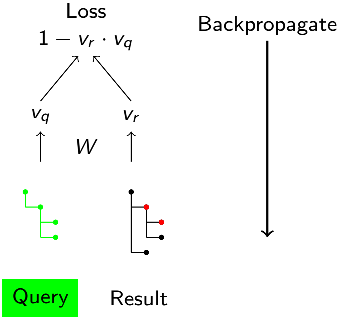
Query

Result

Current method - using gradients



Current method - using gradients

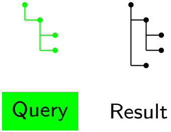


Nodes with highest gradient are considered most important

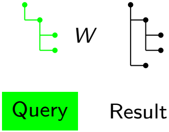
Drawbacks with using gradients

- Compute-intensive, since we need to do backwards pass
- Quality of explanations is not the best
  - Evaluating Recurrent Neural Network Explanations (Arras et al.) `https://arxiv.org/abs/1904.11829`

Comparing node embeddings

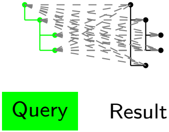


Comparing node embeddings



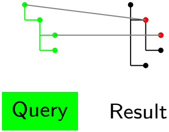
1. Embed graphs using model

Comparing node embeddings



1. Embed graphs using model
2. Compare each pair of node embeddings

Comparing node embeddings



1. Embed graphs using model
2. Compare each pair of node embeddings
3. Highlight most similar nodes

Comparing node embeddings

Why?

- Faster than using gradients (no backprop step needed)
- Might give more relevant explanations
- Can be useful for finding missing features

References

- *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs (Malkov et al.* `https://arxiv.org/abs/1603.09320`
- *Approximate nearest neighbor algorithm based on navigable small world graphs (Malkov et al* `https://doi.org/10.1016/j.is.2013.10.006`
- *Voronoi diagrams—a survey of a fundamental geometric data structure (Aurenhammer)* `https://dl.acm.org/doi/10.1145/116873.116880`
- *Hierarchical Navigable Small Worlds (HNSW) (Pinecone blog)* `https://www.pinecone.io/learn/hnsw/`