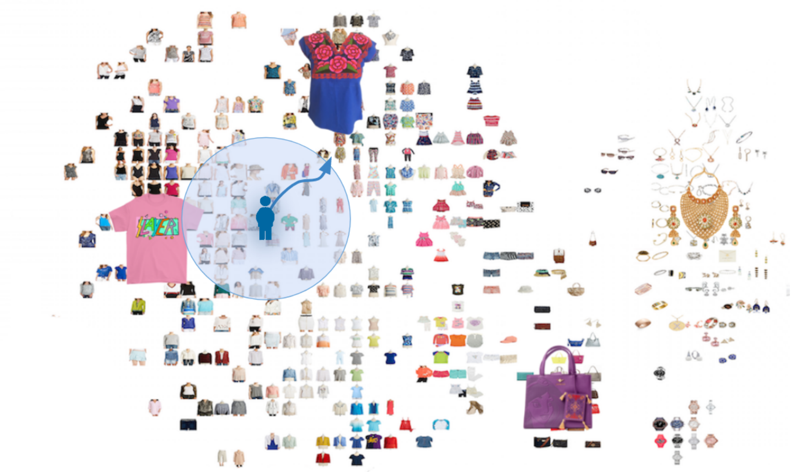
why



- ALS find user recos after matrix factorization
- use currently brute force

→ can we improve efficiency (talk topic)?

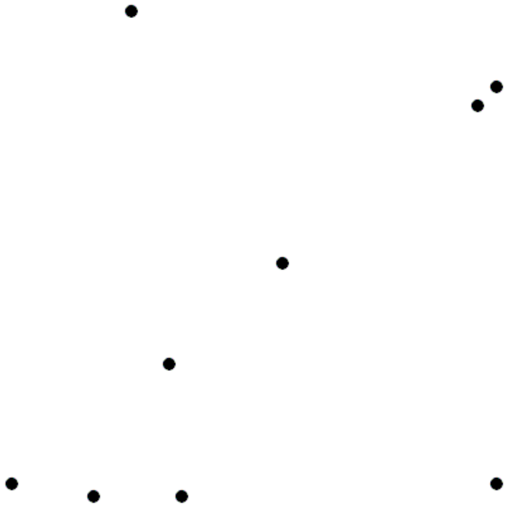
nmslib contains several methods https://github.com/nmslib/nmslib

Non Metric Space library with focus on approximative queries.

- suggested by Faiss (Facebook library which is better for 1000 times bigger data)
  https://code.facebook.com/posts/1373769912645926/faiss-a-library-for-efficient-similarity-search/
- many distance implementations also for text and images → good as overview
  - Space partitioning methods Example 1 next slides
  - Locality Sensitive Hashing (LSH) methods
  - Filter-and-refine methods based on projection to a lower-dimensional space
  - Filtering methods based on permutations
  - Methods that construct a proximity graph Example 2 next slides
  - Miscellaneous methods

All nmslib methods by name: vptree, mvptree, ghtree, list_clusters, satree, bbtree, lsh_multiprobe, lsh_gaussian, lsh_cauchy, lsh_threshold, proj_incsort, proj_vptree, omedrank, pp-index, mi-file, napp, perm_incsort_bin, perm_bin_vptree, sw-graph, hnsw, nndes, seq_search

Example 1: Voronoi (e.g., Spatial Approximation tree (SA-tree))

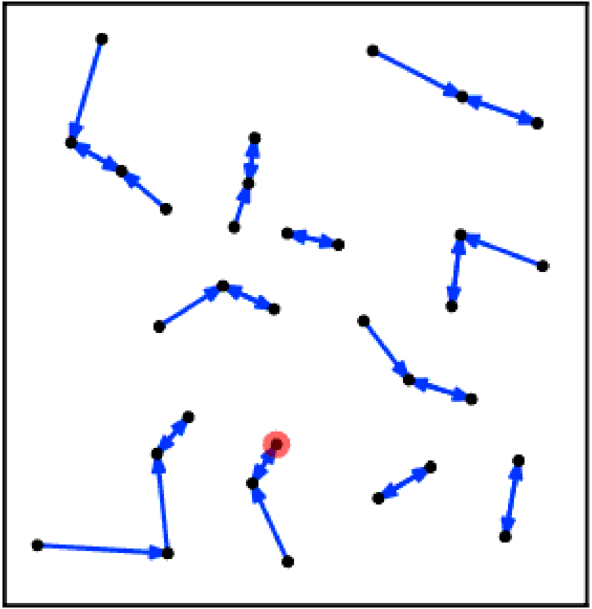


rough sketch:

- choose generator points
- group by closest generator point and assign index

https://en.wikipedia.org/wiki/Voronoi_diagram

Example 2: SW-graph ∈ Neighborhood Graphs



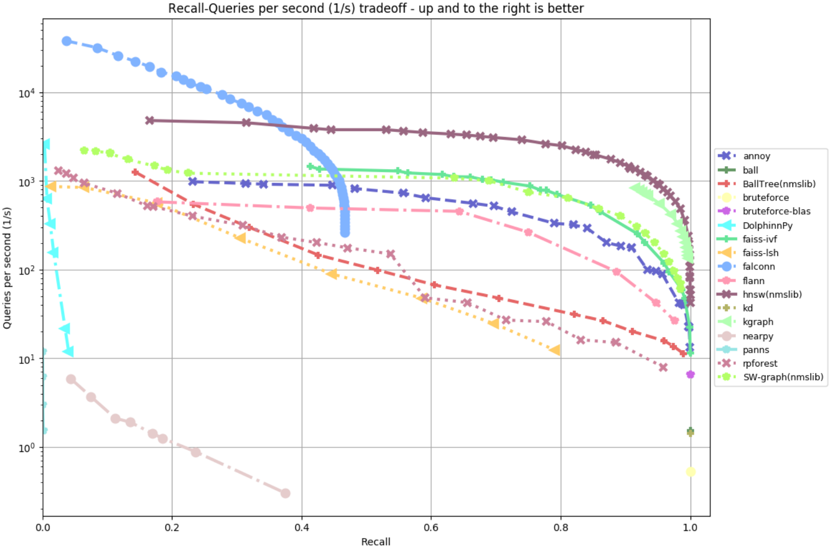
wiki Nearest-neighbor chain algorithm

- If S is empty, choose an active cluster arbitrarily and push it onto S.
- Let C be the active cluster on the top of S. Compute the distances from C to all other clusters, and let D be the nearest other cluster.
- If D is already in S, it must be the immediate predecessor of C. Pop both clusters from S and merge them.
- Otherwise, if D is not already in S, push it onto S.

query similar to Dijkstras shortest-path

what should be possible performance (glove-100-angular)

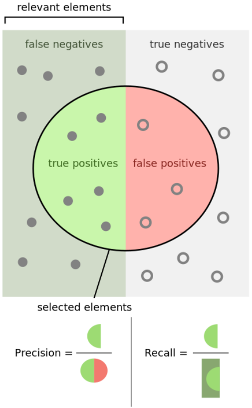
https://github.com/erikbern/ann-benchmarks (python)



c5.4xlarge machine on AWS

result /nmslib/similarity_search/release/experiment (random data, P50)

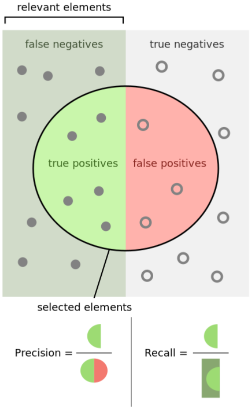
MethodName	Recall	Recall@1	RPE	NPC	QTime	DistComp	ImprEff	ImprDist	Mem	IdxT	QPSec
vptree	1	1	1	0	21.7	400000	0.72	1	201	4	46
mvp-tree	1	1	1	0	112.6	399999	0.14	1	137	2	9
ghtree	1	1	1	0	25.7	400000	0.61	1	231	1	39
clusters	1	1	1	0	17.5	399898	0.89	1.00	122	489	57
satree	1	1	1	0	124.7	400000	0.13	1	158	9	8
omedrank	0.63	1	1.54	0	45.1	126792	0.35	3.15	127	0	22
inverted idx	0.41	1	2.43	0	16.6	20512	0.93	19.50	176	28	60
permutation	1.00	1	1.00	0	19.8	292296	0.79	1.37	586	4	50
perm bin	0.20	1	5.25	0	16.0	20016	0.97	19.99	106	1	62
bin perm	0.21	1	5.20	0	17.1	20016	0.90	19.98	246	2	58
sw-graph	0.06	0.78	2.37	14	0.3	1516	50.87	263.79	650	23	3298
hnsw	0.19	0.63	3.01	10	0.2	897	77.28	446.16	652	105	4916
seq search	1	1	1	0	14.6	400000	1.07	1	106	0	69



- Recall@1: Percentage of queries for which the true nearest neighbor is returned first in the result list. ?
- RPE: RelPosError $\frac{1}{N}\sum_{i=1}^{N}\frac{\mathrm{pos}(o_i)}{i}$
- NPC: NumPointsCloser (points closer than best query return, optimal 0)
- QTime: Query runtime [ms]
- DistComp: Number of distance computations.
- ImprEff: Improvement in runtime (improvement in efficiency) with respect to a sequential search (brute force).
- ImprDist: Improvement in the number of distance computations.
- Mem: Amount of memory used by the index and the data [MB].
- IdxT: Index time.
- QPSec: Queries per second.

result /nmslib/similarity_search/release/experiment (tesla data, P50)

MethodName	Recall	Recall@1	RPE	NPC	QTime	DistComp	ImprEff	ImprDist	Mem	IdxT	QPSec
vptree	1	1	1	0	15.4	223805	2.24	3.2	424	9	65
mvp-tree	1	1	1	0	58.8	193013	0.58	3.7	317	4	17
ghtree	1	1	1	0	53.2	504958	0.65	1.4	1569	29	19
clusters	1	1	1	0	16.8	243877	2.10	2.9	282	1662	59
satree	1	1	1	0	99.6	277970	0.36	2.6	446	16	10
omedrank	0.98	1	1.02	0	134.6	390457	0.27	1.8	295	0.7	7
inverted idx	0.90	1	1.11	0	30.4	36007	1.18	19.7	383	55	32
permutation	1.00	1	1.00	0	37.5	307035	0.98	2.3	801	9	26
perm bin	0.29	1	4.33	0	27.1	35511	1.33	20	257	1	37
bin vptree	0.30	0.39	4.27	6	18.6	35511	1.91	20	496	2	54
sw-graph	0.41	0.93	2.13	2	0.2	881	181	805	801	27	5145
hnsw	0.51	0.77	1.86	17	0.1	370	305	1917	804	69	8389
seq search	1	1	1	0	33.7	709910	1.07	1	257	0	30



- Recall@1: Percentage of queries for which the true nearest neighbor is returned first in the result list. ?
- RPE: RelPosError $\frac{1}{N}\sum_{i=1}^{N}\frac{\mathrm{pos}(o_i)}{i}$
- NPC: NumPointsCloser (points closer than best query return, optimal 0)
- QTime: Query runtime [ms]
- DistComp: Number of distance computations.
- ImprEff: Improvement in runtime (improvement in efficiency) with respect to a sequential search (brute force).
- ImprDist: Improvement in the number of distance computations.
- Mem: Amount of memory used by the index and the data [MB].
- IdxT: Index time.
- QPSec: Queries per second.

result scala (https://github.com/sambackhaus/sandbox.git), P50

```
val queries: Seq[GenericQuery] = Seq(
   new KdtreeQuery("dataPath"),
   new LshQuery("dataPath"),
   new NmslibQuery("dataPath"),
   new ReferenceQuery("dataPath"))

val resultProfiles = queries.map(q => {
     System.gc()
     q.tearUp()
     val deadline: Deadline = deadlineSeconds.seconds.fromNow
     while (deadline.hasTimeLeft)
            {queryVectors(Random.nextInt(queryVectors.size)), 150)}
     val profile = q.getProfile()
     q.tearDown()
     profile})
```

nmslib essence:

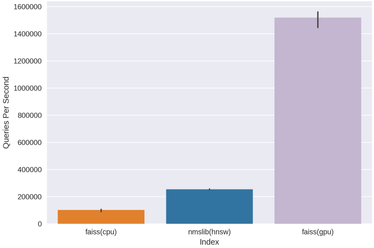
```
val pathToQueryServer =
      "../nmslib/query_server/cpp_client_server/query_server"
s"$pathToQueryServer -i ./$dataUrl -s l1 -m hnsw -c
      efConstruction=400,delaunay_type=0 -p 10000" run

val transport: TSocket = new TSocket("localhost", 10000)
transport.get.open()

val protocol: TBinaryProtocol = new TBinaryProtocol(transport.get)
val client: QueryService.Client = new QueryService.Client(protocol)

result: Seq[ReplyEntry] = client.knnQuery(nearestNeighborCount,
      queryString, true, false).toList
```

Result (random):
numPoints: 400000
dimensions: 90
neighbours: 150
KdtreeQuery, avg query: 1606 ms
LshQuery, avg query: 44 ms
NmslibQuery, avg query: **0.6 ms**
ReferenceQuery, avg query: 56 ms

Result (random):
numPoints: 1000000
dimensions: 90
neighbours: 150
KdtreeQuery, avg query: 3889 ms
LshQuery, avg query: 157 ms
NmslibQuery, avg query: **0.5 ms**
ReferenceQuery, avg query: 410 ms Result

(tesla data):
numPoints: 709910
dimensions: 94
neighbours: 150
KdtreeQuery, avg query: 2244 ms
LshQuery, avg query: 2 ms
NmslibQuery, avg query: **0.5 ms**
ReferenceQuery, avg query: 112 ms

outlook

- signifficant improvement possible up to $\mathcal{O}(100)$
- organization of data important (random vs real)
- removing, adding data & re-indexing (seems not entirely easy)
- was@bi has Faiss running on the BI-Power with GPUs. Talk, e.g., to Darius Morawiec (vectors $\simeq$ 4000 dim and several million sets)
- 

  comparison by Ben Frederickson (just one random benchmark)