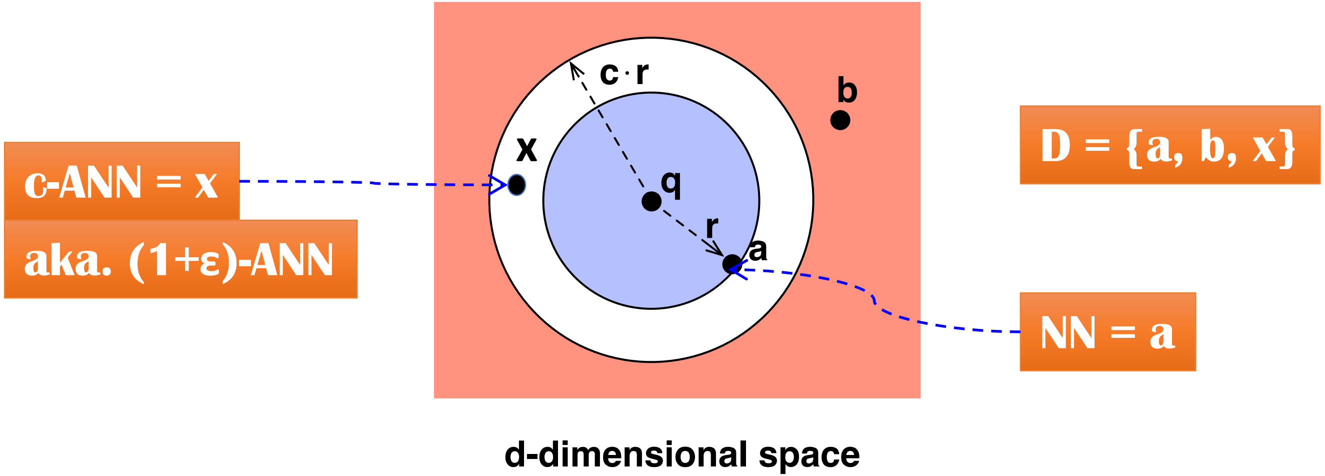
Approximate Nearest Neighbor Search based on Product Quantization

Wei Wang

CSE, UNSW

Nearest Neighbor (NN) and its Variants



- Given: n points in d-dimensional Euclidean space
- NN query: find the closest point, o*, in D, wrt a query point q
- Approximate NN query: find a near point
- All extendible to kNN versions for k > 1

Motivations

- Theoretical interest
  - Fundamental geometric problem: “post-office problem”
  - Known to be hard due to high-dimensionality
- Many applications
  - Feature vectors: Data Mining, Multimedia DB
  - Applications based on similarity queries, e.g.,
    - Quantization in coding/compression
    - Recommendation systems
    - Bioinformatics/Chemoinformatics
  - Machine Learning
    - Representation learning (e.g., embedding)
    - Dimensionality reduction; collaborative filtering; kNN classifier; Kernel?

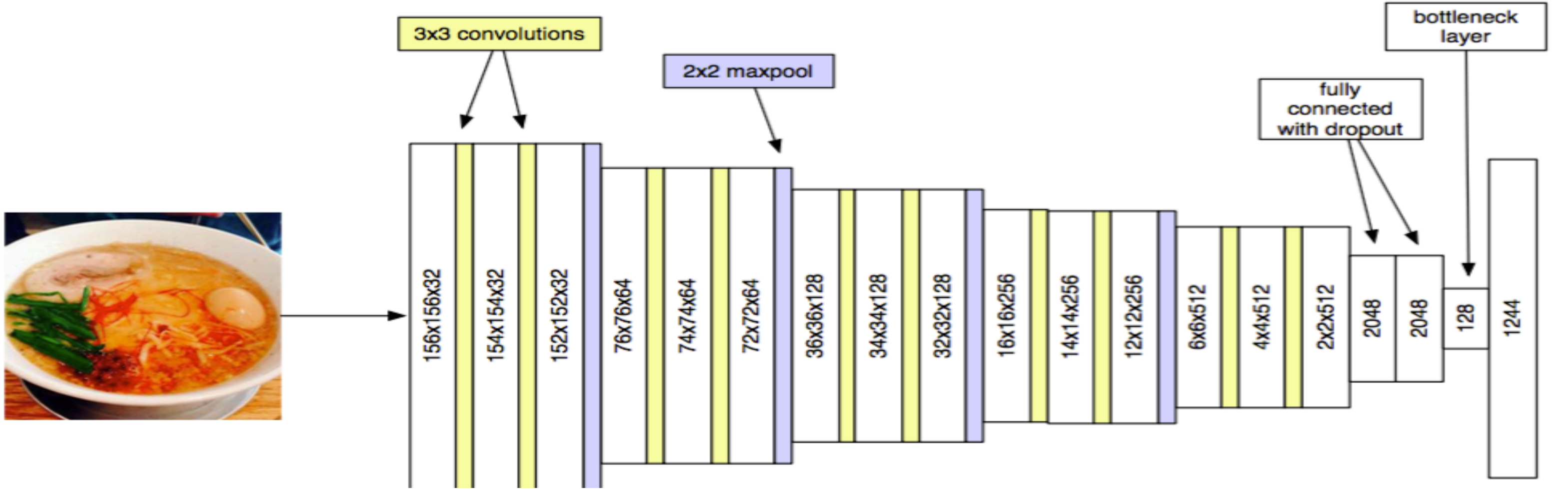
Dimensionality reduction for ML

- Start with high-dimensional data
- Run dimensionality reduction
- Do stuff in a small dimensional space

Erik Bernhardsson, “Approximate nearest neighbor methods and vector models”, 2015

Deep Learning for food

- Deep model trained on a GPU on 6M random pics downloaded from Yelp



Erik Bernhardsson, "Approximate nearest neighbor methods and vector models", 2015

Distance in Smaller Space

1. Run image through the network
2. Use the 128-dimensional bottleneck layer as an item vector
3. Use cosine distance in the reduced space

$f($  $) = \mathbf{v} \qquad (\mathbf{v} \in \mathbb{R}^{128})$

$$d(\mathbf{u}, \mathbf{v}) = \left(\frac{\mathbf{u}}{|\mathbf{u}|} - \frac{\mathbf{v}}{|\mathbf{v}|} \right)^2$$

Erik Bernhardsson, “Approximate nearest neighbor methods and vector models”, 2015

Nearest Neighbor Food Pics

Erik Bernhardsson, "Approximate nearest neighbor methods and vector models", 2015

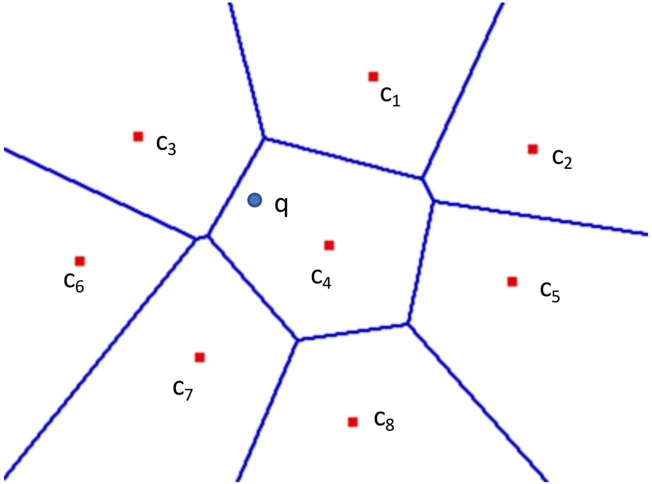
Preliminary: Vector Quantization (with K-means)

- Idea: compressed representation of vectors
  - Each D-dim vector x is represented by QZ(x), where QZ() is a quantizer
  - dist(x, o) ≈ dist(x, QZ(o))
- Encode the vectors:
  - Learn a codebook $W = \{c_1, c_2, \ldots, c_K\}$ via K-means
  - Assign o to its **nearest** codeword in W
    - E.g., $QZ(o) = c_i$ ($i \in 1 \ldots K$), such that $dist(x, c_i) \leq dist(x, c_j)$ for $\forall j$
  - Represent each vector o by its assigned codeword
- Assume D = 256, $K = 2^{16}$
  - Before: 4 bytes * 256 = 1024 bytes for each vector
  - Now:
    - data: 16 bits = 2 bytes
    - codebook: $4 * 256 * 2^{16}$

$\boxed{\text{smaller when } n > 4*256*2^{16}/1022 = 65{,}664}$

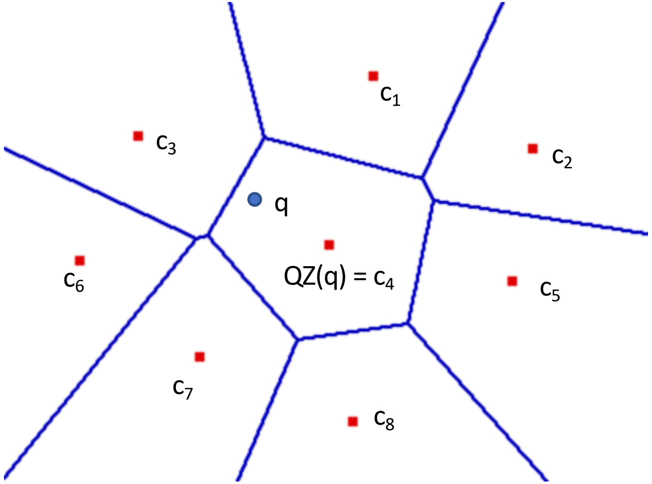
Vector Quantization – Query Processing

- Given query q, how to find a point close to q?



Vector Quantization – Query Processing

- Given query q, how to find a point close to q?
- Algorithm:
  1. Compute QZ(q)
  2. Candidate set C = all data vectors associated with QZ(q)
  3. Verification: compute distance between Q and $o_i \in C$
     - Requires loading the vectors in C



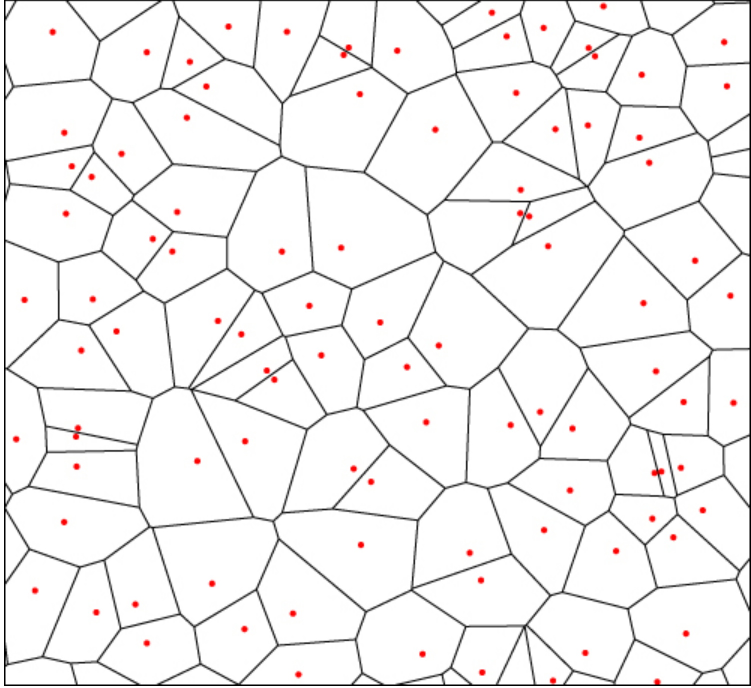
Inverted index: a hash table that maps c_j to a list of o_i that are associated with c_j

- Any problem/improvement?

Limitations of VQ

- To achieve better accuracy, fine-grained quantizer with large K is needed
- Large K ➔
  - Costly to run K-means
  - Computing QZ(q) is expensive: O(K M)
  - May need to look beyond q(Q) cell

- Solutions:
  - Product Quantization, or
  - Hierarchical k-means



Product Quantization

- Idea:
  - Partition the dimension into m partitions
    - Accordingly a vector ➔ m subvectors
  - Use separate VQ with k codewords for each chunk
- Example:
  - 10-dim vector decomposed in m = 2 subvectors
    - $o^T = [o_1 : o_2]^T$
  - Each codebook has 4 codewords, denoted as $c_{i,j}$
  - Total space in bits:
    - data: m log(K)
    - codebook: m * ((D/m) * K)

7	2	-1	5	6	4	3	-5	-1	-7
-2	3	0	6	4	2	2	4	5	-8
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…

0	1	0	0
1	0	1	1
…	…	…	…
…	…	…	…
…	…	…	…

$C_{1,1}$	5.9	2.3	-2.7	3.9	6.1
$C_{1,2}$	-1.3	1.8	7.4	5.5	0.9
$C_{1,3}$	…	…	…	…	…
$C_{1,4}$	…	…	…	…	…

$C_{2,1}$	-0.1	3.5	1.4	9.6	5.5
$C_{2,2}$	…	…	…	…	…
$C_{2,3}$	…	…	…	…	…
$C_{2,4}$	…	…	…	…	…

Distance Estimation

- Euclidean distance between a query point **q** and a data point encoded as **t**
  - Restore the virtual joint center by looking up each partition of **t** in the corresponding codebooks ➔**p**

$$d^2(\mathbf{q}, \mathbf{t}) = \sum_{i=1}^{D} (\mathbf{q}_i - \mathbf{p}_i)^2$$

  - This is the Asymmetric Distance (AD)

$$d^2(\mathbf{q}, \mathbf{t}) = \sum_{i=1}^{m} (\mathbf{q}_{(i)} - \mathbf{c}_{i,\mathbf{t}_{(i)}})^2$$

q

7	2	-1	5	6	4	3	-5	-1	-7

t

0	1	0	0

$C_{1,1}$	5.9	2.3	-2.7	3.9	6.1
$C_{1,2}$	-1.3	1.8	7.4	5.5	0.9
$C_{1,3}$	…	…	…	…	…
$C_{1,4}$	…	…	…	…	…

$C_{2,1}$	-0.1	3.5	1.4	9.6	5.5
$C_{2,2}$	…	…	…	…	…
$C_{2,3}$	…	…	…	…	…
$C_{2,4}$	…	…	…	…	…

p

-1.3	1.8	7.4	5.5	0.9	-0.1	3.5	1.4	9.6	5.5

PQ – Query Processing

q

7	2	-1	5	6	4	3	-5	-1	-7

t	0	1	0	0
t'	1	0	1	1
	1	0	1	0
	0	0	1	0
	1	1	1	1

- Naïve:
  - Perform ADC for every **t** in the database
  - Candidate = those with the k smallest AD
  - [Optional] Reranking (if k > 1):
    - Load the data vectors and compute the actual Euclidean distance
    - Return the one with the smallest distance
- Pruning:
  - AD is monotonic in each component

$$d^2(\mathbf{q}, \mathbf{t}) = \sum_{i=1}^{m} (\mathbf{q}_{(i)} - \mathbf{c}_{i,\mathbf{t}_{(i)}})^2$$

$$(\mathbf{q}_{(1)} - \mathbf{c}_{i,\mathbf{t}_{(1)}})^2 + (\mathbf{q}_{(2)} - \mathbf{c}_{i,\mathbf{t}_{(2)}})^2$$

$$(\mathbf{q}_{(1)} - \mathbf{c}_{i,\mathbf{t}'_{(1)}})^2 + (\mathbf{q}_{(2)} - \mathbf{c}_{i,\mathbf{t}'_{(2)}})^2$$

There is an optimal order to consider each encoded points!

Compute it efficiently and incrementally!

NOTE: Reusing figure in the paper. m = 2, each code book has 6 codewords, named $\mathbf{U_i}$ and $\mathbf{V_j}$

Multi-index Algorithm

1. Sort each code book in increasing order of their partial AD distance to q

2. Maintain the non-dominated combinations into a min heap H

NOTE: slightly different from the paper – Skyline algorithm

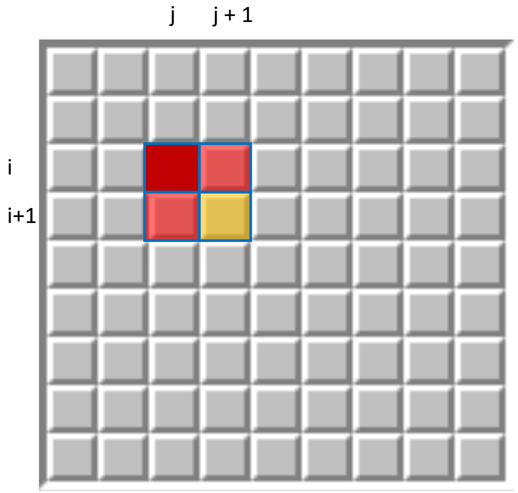
$\mathbf{q}_{(1)}$ vs. $\mathcal{U}$

i	$\mathbf{u}_{\alpha(i)}$	r
1	$\mathbf{u_3}$	0.5
2	$\mathbf{u_4}$	0.7
3	$\mathbf{u_5}$	4
4	$\mathbf{u_2}$	6
5	$\mathbf{u_1}$	8
6	$\mathbf{u_6}$	9

$\mathbf{q}_{(2)}$ vs. $\mathcal{V}$

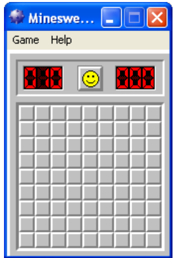
j	$\mathbf{v}_{\beta(j)}$	s
1	$\mathbf{v_4}$	0.1
2	$\mathbf{v_3}$	2
3	$\mathbf{v_5}$	3
4	$\mathbf{v_2}$	6
5	$\mathbf{v_6}$	7
6	$\mathbf{v_1}$	11

partial AD



(i,j) ➔ pushes (i+1, j) and (i, j+1) into H

Dedup: need to check if it has EVER been pushed into H or not



NOTE: Reusing figure in the paper. m = 2, each code book has 6 codewords, named **U_i** and **V_j**

Multi-index Algorithm

1. Sort each code book in increasing order of their partial AD distance to q

2. Maintain the non-dominated combinations into a min heap H

$\mathbf{q}_{(1)}$ vs. $\mathcal{U}$

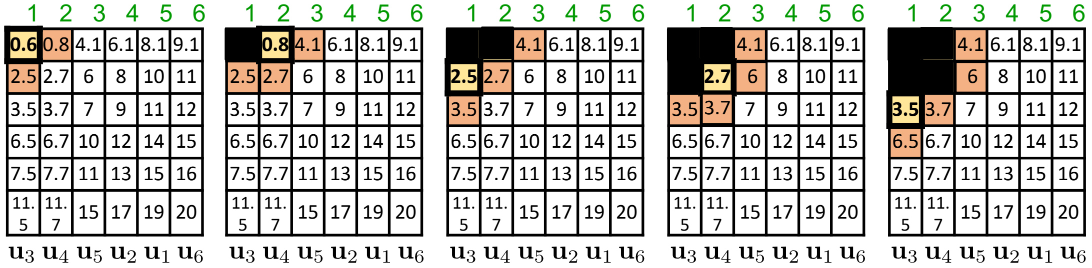
i	$\mathbf{u}_{\alpha(i)}$	r
1	$\mathbf{u_3}$	0.5
2	$\mathbf{u_4}$	0.7
3	$\mathbf{u_5}$	4
4	$\mathbf{u_2}$	6
5	$\mathbf{u_1}$	8
6	$\mathbf{u_6}$	9

$\mathbf{q}_{(2)}$ vs. $\mathcal{V}$

j	$\mathbf{v}_{\beta(j)}$	s
1	$\mathbf{v_4}$	0.1
2	$\mathbf{v_3}$	2
3	$\mathbf{v_5}$	3
4	$\mathbf{v_2}$	6
5	$\mathbf{v_6}$	7
6	$\mathbf{v_1}$	11

partial AD

		1	2	3	4	5	6
$\mathbf{v}_4$	1	0.6	0.8	4.1	6.1	8.1	9.1
$\mathbf{v}_3$	2	2.5	2.7	6	8	10	11
$\mathbf{v}_5$	3	3.5	3.7	7	9	11	12
$\mathbf{v}_2$	4	6.5	6.7	10	12	14	15
$\mathbf{v}_6$	5	7.5	7.7	11	13	15	16
$\mathbf{v}_1$	6	11.5	11.7	15	17	19	20
		$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_2$	$\mathbf{u}_1$	$\mathbf{u}_6$



(2,2) not pushed again

(2,3) not pushed again

Exercises:

- Write out H
- How to perform dedup?
- Generalize to m-dimensional case
- [hard] perform dedup without additional data structure (and cost)

References

- Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. TPAMI 2014.
- Artem Babenko and Victor Lempitsky. The Inverted Multi-Index. TPAMI 2014.