Nearest Neighbor Search

• Preprocess a set of \( n \) vectors \( V \) in \( d \) dimensions into a search data structure.
• Input: A query vector \( q \).
• Output: The vector \( v \) in \( V \) that is nearest to \( q \). That is, the vector \( v \) in \( V \) that minimizes

\[
\| v - q \|_2^2 = \sum_{i=1}^{d} (v(i) - q(i))^2
\]

NNS in VQ

• Used in codebook design.
  – Used in GLA to partition the training set.
  – Since codebook design is seldom done then speed of NNS is not too big a issue.
• Used in VQ encoding.
  – Codebook size is commonly 1,000 or more.
  – Naive linear search would make encoding too slow.
  – Can we do better than linear search?

Naive Linear Search

• Keep track of the current best vector, best-\( v \), and best distance squared, best-\( \text{squared} \).
  – For an unsearched vector \( v \) compute \( \| v - q \|_2^2 \) to see if it smaller than best-\( \text{squared} \).
  – If so then replace best-\( v \).
  – If \( d \) is moderately large it is a good idea not to compute the squared distance completely. Bail out when \( k < d \) and

\[
\text{best - squared} - d \leq \sum_{i=1}^{d} (v(i) - q(i))^2
\]

Orchard’s Method

• Invented by Orchard (1991)
• Uses a “guess codeword”. The guess codeword is the codeword of an adjacent coded vector.
• Orchard’s Principle.
  – If \( r \) is the distance between the guess codeword and the query vector \( q \) then the nearest codeword to \( q \) must be within a distance of \( 2r \) of the guess codeword.
Orchard Data Structure

- For each codeword sort the other codewords by distance to the codeword.

Basic Orchard Algorithm

Let i be the index of initial guess codeword:
- \( r := ||c_i - q||; \)
- \( \text{best-index} := i; \)
- \( j := 1; \)
- while \( A[i,j].\text{distance} < 2 \cdot r \) do
  - if \( ||c_j - q|| < r \) then \( \text{best-index} := j; \)
  - \( j := j + 1; \)

This algorithm searches all the codewords within a distance \( 2r \) of the guess codeword.

Orchard Improvements

- Early bailout using squared distance:
  - \( ||c_j - q|| < r \) is done by early bailout using the comparison \( ||c_j - q||^2 < r^2 \).
- Switching Lists:
  - When a nearer codeword is found then switch the search to its list.
  - Care must be taken to avoid doing a distance computation the same codeword twice. Marking visited codewords solves this problem.

Switching Lists (1)

Switching Lists (2)

Orchard Notes

- Very fast.
  - Appears \( O(\log n) \) average time per search, but there is no proof of this performance.
- Requires too much memory.
  - Requires \( O(n^2) \) memory for the sorted lists.
  - Modification for large \( n \). Just store the first \( m \) closest to make an \( n \times m \) array. If the search runs off the array then revert to linear search.
k-d Tree

- Jon Bentley, 1975
- Tree used to store spatial data.
  - Nearest neighbor search.
  - Range queries.
  - Fast look-up
- k-d tree are guaranteed $\log_2 n$ depth where $n$ is the number of points in the set.
  - Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

k-d Tree Construction

- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.
- Division strategies
  - divide points perpendicular to the axis with widest spread.
  - divide in a round-robin fashion.

k-d Tree Construction (1)

k-d Tree Construction (2)

k-d Tree Construction (3)

k-d Tree Construction (4)
k-d Tree Construction (5)

k-d Tree Construction (6)

k-d Tree Construction (7)

k-d Tree Construction (8)

k-d Tree Construction (9)

k-d Tree Construction (10)
k-d Tree Construction (11)

k-d Tree Construction (12)

k-d Tree Construction (13)

k-d Tree Construction (14)

k-d Tree Construction (15)

k-d Tree Construction (16)
k-d Tree Construction Complexity

- First sort the points in each dimension.
  - $O(dn \log n)$ time and $dn$ storage.
  - These are stored in $A[1..d,1..n]$.
- Finding the widest spread and equally divide into two subsets can be done in $O(dn)$ time.
- Constructing the k-d tree can be done in $O(dn \log n)$ and $dn$ storage.

k-d Tree Codebook Organization

k-d Tree Splitting

- sorted points in each dimension
- $x: 1 2 3 4 5 6 7 8 9$
- $y: a b c d e f g h i$
- max spread is the max of $x_i - a_i$ and $y_i - a_i$
- In the selected dimension the middle point in the list splits the data.
- To build the sorted lists for the other dimensions scan the sorted list adding each point to one of two sorted lists.

Node Structure for k-d Trees

- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)
k-d Tree Nearest Neighbor Search

NNS(q: point, n: node, p: ref point w: ref distance)
if n.left = n.right = null then {leaf case}
w' := ||q - n.point||;
if w' < w then
    w := w';
    p := n.point;
else
    if q(n.axis) < n.value then {query to left}
        NNS(q, n.left, p, w);
    if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
    else {query to right}
        NNS(q, n.right, p, w);
    if q(n.axis) - w < n.value then NNS(q, n.left, p, w)
NNS(q, root, p, infty) initial call
Notes on k-d NNS

- Has been shown to run in \( O(\log n) \) average time per search in a reasonable model. (Assume \( d \) a constant)
- For VQ it appears that \( O(\log n) \) is correct.
- Storage for the k-d tree is \( O(n) \).
- Preprocessing time is \( O(n \log n) \) assuming \( d \) is a constant.

Alternative is PCP-Tree

- Zatloukal, Johnson, Ladner (1999)
- Organize a tree using principal components partitioning.
  - Partition the data perpendicular to the line that minimizes the sum of the distances of the points to the line. Eigenvector computation required.
- About is easy to construct as the k-d tree.
- In NNS processing per node in the PCP tree is more expensive than in the k-d tree, but fewer codewords are searched.
Principal Component Partition

PCP Tree vs. k-d tree

PCP Tree vs. k-d tree

16D Codewords Searched

16D Execution Time

Comparison in Time per Search

NNS Summary

- Orchard
  - fastest
  - excessive memory
  - good guess codeword needed
- k-d tree
  - good in low dimension
  - small storage
  - no guess codeword
- PCP
  - best in high dimension
  - fewer codewords searched than k-d
  - small storage
  - no guess codeword

4,096 codewords
Notes on VQ

• Works well in some applications.
  – Requires training
• Has some interesting algorithms.
  – Codebook design
  – Nearest neighbor search
• Variable length codes for VQ.
  – PTSVQ - pruned tree structured VQ (Chou, Lookabaugh and Gray, 1989)
  – ECVQ - entropy constrained VQ (Chou, Lookabaugh and Gray, 1989)