CSE 589
Applied Algorithms
Autumn 2001

Semitur Conclusion
Image Compression
Vector Quantization
Nearest Neighbor Search

Semitur Algorithm
Input the first symbol s to create the production S -> s;
multiple
match an existing rule:
A -> ...XY...
B -> XY
create a new rule:
A -> ...XY...
B -> ...C...
remove a rule:
A -> ...B...
B -> X,Y...
input a new symbol:
S -> X,Y...
until no symbols left

Basic Encoding a Grammar

Grammar

Symbol Code
S 010
A 011
B 100
D 101
# 110

Grammar Code

D B D # b e # A # b B e
101 100 101 110 000 001 110 011 110 000 100 001 39 bits

r = number of rules
s = sum of right hand sides
a = number in original symbol alphabet

Compress Quality

• Neville-Manning and Witten 1997

<table>
<thead>
<tr>
<th>size</th>
<th>compress</th>
<th>gprp</th>
<th>semiquot</th>
<th>PPMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>bib</td>
<td>111261</td>
<td>3.35</td>
<td>2.51</td>
<td>2.48</td>
</tr>
<tr>
<td>book</td>
<td>768771</td>
<td>3.46</td>
<td>3.35</td>
<td>2.82</td>
</tr>
<tr>
<td>geo</td>
<td>1024003</td>
<td>6.06</td>
<td>5.34</td>
<td>4.74</td>
</tr>
<tr>
<td>clips</td>
<td>246814</td>
<td>4.17</td>
<td>2.63</td>
<td>2.68</td>
</tr>
<tr>
<td>jpg</td>
<td>513216</td>
<td>0.97</td>
<td>0.82</td>
<td>0.90</td>
</tr>
<tr>
<td>progc</td>
<td>38613</td>
<td>3.87</td>
<td>2.68</td>
<td>2.83</td>
</tr>
</tbody>
</table>

Files from the Calgary Corpus
Units in bits per character (8 bits)
compress - based on LZW
gprp - based on LZ77
PPMC - adaptive arithmetic coding with context
Notes on Sequitur

- Very new and different from the standards.
- Yields compression and hierarchical structure simultaneously.
- With clever encoding is competitive with the best of the standards.
- Practical linear time encoding and decoding.
- Alternatives
  - Off-line algorithms – (i) find the most frequent digram, (ii) find the longest repeated substring

Exercise

- Apply Sequitur to the string aaaaaaaaaaa (16 a’s).
- What is the length of the code assuming a 2 symbol alphabet?

Lossy Image Compression Methods

- Basic theory - trade-off between bit rate and distortion.
- Vector quantization (VQ).
  - A indices of set of representative blocks can be used to code an image, yielding good compression. Requires training.
- Wavelet Compression.
  - An image can be decomposed into a low resolution version and the detail needed to recover the original. Sending most significant bits of the wavelet coded yields excellent compression.

JPEG Standard

- JPEG - Joint Photographic Experts Group
  - JPEG 2000 uses wavelet compression.

Barbara

- Original
- JPEG
- VQ
- Wavelet-SPHT

32:1 compression ratio
.25 bits/pixel (8 bits)

JPEG

- Current image compression standard.
**Distortion**

- Lossy compression: $x \neq \hat{x}$
- Measure of distortion is commonly mean squared error (MSE). Assume $x$ has $n$ real components (pixels).

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2$$

**PSNR**

- Peak Signal to Noise Ratio (PSNR) is the standard way to measure fidelity.

$$PSNR = 10 \log_{10} \left( \frac{m^2}{MSE} \right)$$

where $m$ is the maximum value of a pixel possible.

For gray scale images (8 bits per pixel) $m = 255$.

- PSNR is measured in decibels (dB).
- .5 to 1 dB is said to be a perceptible difference.

**Rate-Fidelity Curve**

- Increasing
- Slope decreasing
PSNR is not Everything

\[
\text{PSNR} = 25.8 \text{ dB}
\]

\[
\text{PSNR} = 25.8 \text{ dB}
\]

PSNR Reflects Fidelity (1)

\[
\text{PSNR} 25.8
\]

\[
.63 \text{ bpp}
\]

\[
12.8 : 1
\]

PSNR Reflects Fidelity (2)

\[
\text{PSNR} 24.2
\]

\[
.31 \text{ bpp}
\]

\[
25.6 : 1
\]

PSNR Reflects Fidelity (2)

\[
\text{PSNR} 23.2
\]

\[
.16 \text{ bpp}
\]

\[
51.2 : 1
\]

Vector Quantization

- The image is partitioned into \(a \times b\) blocks.
- The codebook has \(n\) representative \(a \times b\) blocks called codewords, each with an index.
- Compression is

\[
\frac{\log \frac{n}{ab}}{ab} \text{ bpp}
\]

- Example: \(a = b = 4\) and \(n = 1,024\)
  - compression is \(10/16 = .63\) bpp
  - compression ratio is \(8 : .63 = 12.8 : 1\)
Examples

4 x 4 blocks .63 bpp
4 x 8 blocks .31 bpp
8 x 8 blocks .16 bpp

Codebook size = 1,024

Encoding and Decoding

• Encoding:
  – Scan the $a \times b$ blocks of the image. For each block find the nearest codeword in the code book and output its index.
  – Nearest neighbor search.
• Decoding:
  – For each index output the codeword with that index into the destination image.
  – Table lookup.

The Codebook

• Both encoder and decoder must have the same codebook.
• The codebook must be useful for many images and be stored somewhere.
• The codebook must be designed properly to be effective.
• Design requires a representative training set.
• These are major drawbacks to VQ.

Codebook Design Problem

• Input: A training set $X$ of vectors of dimension $d$ and a number $n$, ($d = a \times b$ and $n$ is number of codewords)
• Output: $n$ vectors $c_1, c_2, ..., c_n$ that minimizes the sum of the distances from each member of the training set to its nearest codeword. That is minimizes

$$\sum_{x \in X} \| x - c_{d(x)} \|^2 = \sum_{x \in X} \sum_{i=1}^{n} (c_{d(x)}(i) - x(i))^2$$

where $c_{d(x)}$ is the nearest codeword to $x$.

Algorithms for Codebook Design

• The optimal codebook design problem appears to be an NP-hard problem.
• There is a very effective method, called the generalized Lloyd algorithm (GLA) for finding a good local minimum.
• GLA is also known in the statistics community as the k-means algorithm.
• GLA is slow.

GLA

• Start with an initial codebook $c_1, c_2, ..., c_n$ and training set $X$.
• Iterate:
  – Partition $X$ into $X_1, X_2, ..., X_k$ where $X_i$ includes the members of $X$ that are closest to $c_i$.
  – Let $x_i$ be the centroid of $X_i$.
  
  $$x_i = \frac{1}{|X_i|} \sum_{X_i} x$$

  – if $\sum |x_i - c_i|$ is sufficiently small then quit, otherwise continue the iteration with the new codebook $c_1, c_2, ..., c_n = x_1, x_2, ..., x_k$. 
GLA Example (1)

GLA Example (2)

GLA Example (3)

GLA Example (4)

GLA Example (5)

GLA Example (6)
C S E 589 - Lecture 7 - Autumn 2001

GLA Example (7)

codeword

training vector

GLA Example (8)

codeword

training vector

GLA Example (9)

codeword
centroid

GLA Example (10)

codeword
centroid

C S E 589 - Lecture 7 - Autumn 2001

37

38

39

40

Codebook

1 x 2 codewords
Note: codewords diagonally spread

C S E 589 - Lecture 7 - Autumn 2001

41

42

GLA Advice

• Time per iteration is dominated by the partitioning step, which is m nearest neighbor searches where m is the training set size.
  – Average time per iteration O(m log n) assuming d is small.
• Training set size.
  – Training set should be at least 20 training vectors per code word to get reasonable performance.
  – Too small a training set results in "over training".
• Number of iterations can be large.
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 $\|v - q\|^2 = \sum_{i=1}^{d}((v(i) - q(i))^2$.

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 an 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-squared-$d$.
  - For an unsearched vector $v$ compute $||v - q||^2$ to see if it smaller than best-squared-$d$.
  - 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 $\sum_{i=1}^{k}((v(i) - q(i))^2$.

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:

\[ r := b := ||c_i - q||; \]
\[ \text{best-index} := i; \]
\[ j := 1; \]
\[ \text{while } A[i,j].\text{distance} < 2^r \text{ do} \]
\[ \text{if } ||c_j - q|| < b \text{ then} \]
\[ \text{best-index} := j; \]
\[ b := ||c_j - q||; \]
\[ j := j + 1; \]

This algorithm searches all the codewords within a distance $2r$ of the guess codeword.

Orchard Improvements

- Early bailout using squared distance:
  - $||c_i - q|| < b$ is done by early bailout using the comparison $||c_i - q||^2 < b^2$.
- 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

- 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

- Max spread is the max of $x_i - a_i$ and $y_i - b_i$.
- 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

```c
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 w = infinity then
        if q(n.axis) < n.value then
            NNS(q, n.left, p, w);
        if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
        else
            NNS(q, n.right, p, w);
    if q(n.axis) - w < n.value then NNS(q, n.left, p, w);
    else {w is finite}
        if q(n.axis) - w < n.value then NNS(q, n.left, p, w);
        if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
```

initial call: NNS(q, root, p, infinity)
k-d Tree NNS (13)

k-d Tree NNS (14)

k-d Tree NNS (15)

k-d Tree NNS (16)

k-d Tree NNS (17)

k-d Tree NNS (18)
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

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

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)