

L8 -- SIFT + Near-Neighbor Search
[Jeff Phillips - Utah - Data Mining]

Real Data:

text documents

key words searches

image data

Abstract Data w/ abstract distance:

sets of objects | Jaccard distance

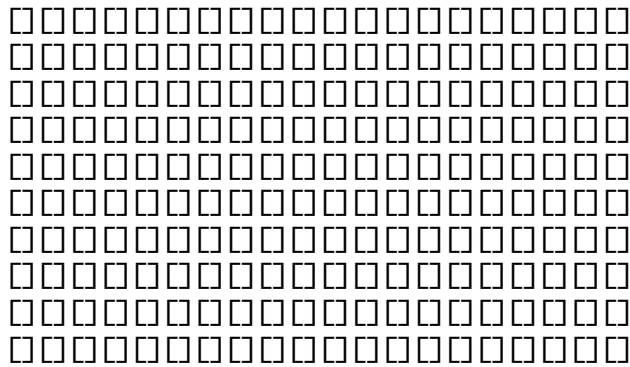
strings | edit distance

SIFT features R^{128} | Euclidean distance

What are SIFT features:

(scale-invariant feature transform)

What is an image:



each \square has rgb-values (lets assume $[0,1]$)

Each \square might have a SIFT feature

-only collect features for extremal points in "scale space"

corners of object in pictures, where color changes abruptly

-determine "scale" σ at which feature is sharpest

Gradient Histogram:

[1][2][3]

[4][X][5]

[6][7][8]

--> gradient histogram:

something like: [1-X][2-X][3-X][4-X][5-X][6-X][7-X][8-X]

shows relative change in magnitude

Consider 4x4 grid with scale σ , vertex at X

```

| | | | |
| 1| 2| | 3| 4|
-----
| | | | |
| 5| 6| | 7| 8|
-----X-----
| | | | |
| 9|10| |11|12|
-----
| | | | |
|13|14| |15|16|

```

for each grid cell i in $[16]$,
 compute a gradient histogram (8 bins) H_i
 make it relative to H_X
 something like: $H_i = H_X/H_i$

X has $8 \times 16 = 128$ vector V_X
 normalize so $\|V_X\| = 1$
 if any component is $> .2$, reset to $.2$ and renormalize

Compare distance between $d(V_X, V_Y)$ as Euclidean distance.
 Use approximate search to speed things up.

 How to find (approximate) near neighbors

Set P subset R^d $|P| = n$. d is large (e.g. 128)

Query point $q \in R^d$
 $p^* = \arg \min_{\{p \in P\}} d(p, q)$

Goal: find p in P s.t.
 $\text{dist}(p, q) \leq (1+\text{eps})\text{dist}(p^*, q)$

centered at q :
 circle C_r radius $r = d(p^*, q)$
 circle $C_{r, \text{eps}}$ radius $(1+\text{eps})r$
 annulus $C_{r, \text{eps}} \setminus C_r$ == don't care

LSH not explicitly designed for ANN. Returns all within r , maybe within $(1+\text{eps})r$. Where r is fixed.
 Can run with progressively larger values of r . But loses some factor. but works ok for very high d (see Andoni code: google "LSH")

****kd-tree:**

divide space by R^d into two points split in dimension i
alternate i in $[d]$ in cyclic order
each step have half remaining points each side

****quad-tree:**

divide space into 2^d axis-aligned rectangles each round,
each has at most $n/2$ points (hopefully less)

****R-tree:**

split points into two covering rectangles each round
searching in $O(2^d \log n)$

****B-tree:** ($\text{dim} = 1$)

split points into B sub-intervals each round.
each "node" stored on one disk block of size B
hard to implement efficiently for $d > 1$

Stop when leaf has $\text{CONSTANT} > 1$ number of points

Now given a query q in R^d :

- find leaf which contains q (find closest point)
- search nearby nodes to see if closer
- don't search sub-trees if ****all**** further than $(1-\text{eps})d(p',q)$

* may need to search many subtrees. Runtime $\sim O(2^d \log n)$ or $O(\log^d n)$

* adds overhead to linear scan (IO efficient)

* with $\text{eps}=0$, linear scan cheaper when $d > 5$ or so

Problem w/ high dimensions

- want ball, get cube

$$\begin{aligned} \text{volume ball}(d, \text{rad}=1) &= \pi^{d/2} / \Gamma(d/2+1) \text{ rad}^d \\ &\sim \pi^{d/2} / ((d/2)!) \\ &\text{gets small} \quad \rightarrow 0 \end{aligned}$$

$$\begin{aligned} \text{volume cube}(d, \text{rad}=1) &= 2^d \\ &\text{gets large} \quad \rightarrow \text{infy} \end{aligned}$$

So with rectilinear search, we get everything in the d -cube, but want everything in d -ball

Approximate methods can go up to maybe $d=8-20$.

Google: "ANN" 3rd hit (which is amazing for the name Ann)

Advanced techniques:

how to choose better split?

- cluster all data (k-means -> split k ways)
- project to k-dim, split 2^k ways.

improve greatly if data is intrinsically in lower dimensions.