Locality-Sensitive Hashing

& Image Similarity Search

Andrew Wylie

Overview; LSH

- given a query *q* (or not), how do we find similar items from a large search set quickly?
  - Can’t do all pairwise comparisons; $nC2$ pairs
- define a measure of similarity for the items, then hash them into buckets using the measure.
  - Items which are similar will be in the same bucket.
- then when given a query *q*, we hash it and return items in the same bucket.

Overview; LSH

- it's a way to do **approximate** near-neighbour search
    - Item signatures used are approximate (mostly)
    - Items hashing to the same bucket is probabilistic

- so multiple hash tables are composed for better accuracy

Overview; LSH

- there are many similarity/distance measures
  - Jaccard
  - Euclidean
  - Hamming
  - Cosine
  - Edit
  - Chi^2
  - *p*-stable
  - Kernelized

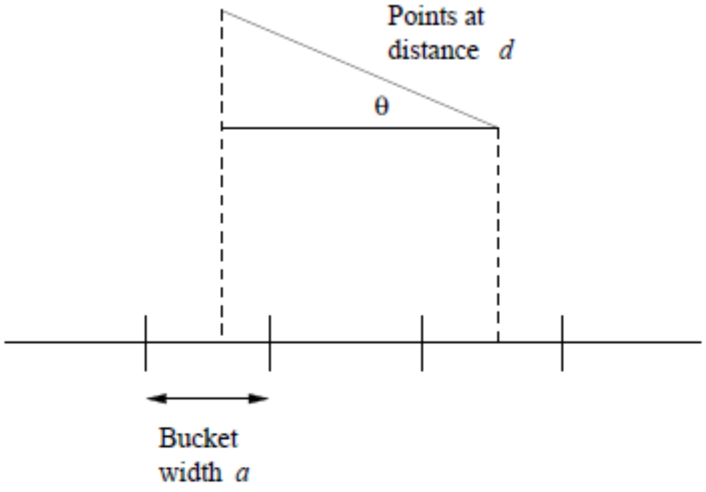
- allows sublinear query time of $O(dn^{1/\,1+\epsilon})$
- preprocessing varies based on data & representation

Euclidean Distance

- *n*-dimensional space
- most often l_2 norm, l_1 & l_∞ norms also used
- $d(v, u) = (\sum_i |v_i - u_i|^p)^{1/p}$
- eg. x = [7, 2, 3], y = [5, 0, -2]
  - $d_2(x, y) = [(7 - 5)^2 + (2 - 0)^2 + (3 - (-2))^2]^{½}$
  - $d_2(x, y) = 29^{1/2} = 5.39$

Euclidean Distance & Random Projections

- we won't compute the distance between the points!
- use a randomly chosen line in 2-space (for each hash fn)
- select a constant *a* to divide line into equal width segments
- points projected onto the line, buckets are the segments
- *(a/2, 2a, 1/2, 1/3)*-sensitive family



Cosine Distance

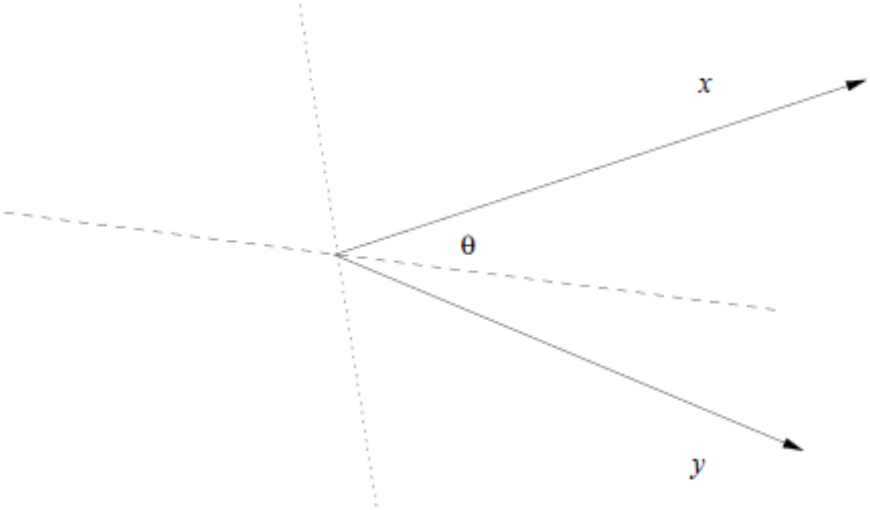
- it's the angle between two vectors/points (in degrees)
- calculated as their dot product divided by l_2 norms
- eg. x = [7, 2, 3], y = [5, 0, -2]
  - $d(x,y) = (7*5) + (2*0) + (3*(-2)) / ||x||_2 \, ||y||_2$
  - $d(x,y) = 29 / 62^{1/2} * 29^{1/2}$
  - $d(x,y) = \cos^{-1}(0.684)$
  - $d(x, y) = 46.8$ degrees

Cosine Distance & Random Hyperplanes

- don’t actually compute this distance for *x* & *y*
- consider a random plane through the origin w/ normal *v*
- compute instead *v.x* & *v.y*

Cosine Distance & Random Hyperplanes

- we'll say they're similar if they have the same sign
- $(d_1, d_2, (180 - d_1)/180, (180 - d_2)/180)$-sensitive



p-Stable Distribution Scheme

- locality-sensitive families for l_p norm using p-stable distribution
  - eg. Gaussian distribution is 2-stable

- distribution is stable if
  - $\sum_i v_i X_i$ has same distribution as $(\sum_i |v_i|^p)^{1/p} X$
- so with v & X as vectors the dot product estimates the l_p norm

p-Stable Distribution Scheme

- dot product is instead used to assign a hash value to *v*
  - projects to a value on the real line
  - split line into equal-width segments of size *r* for buckets
- two vectors which are close have a small difference between norms, and should collide

- $h_{a,b}(v) = \lfloor (a.v + b) / r \rfloor$
- family is (r_1, r_2, p_1, p_2)-sensitive

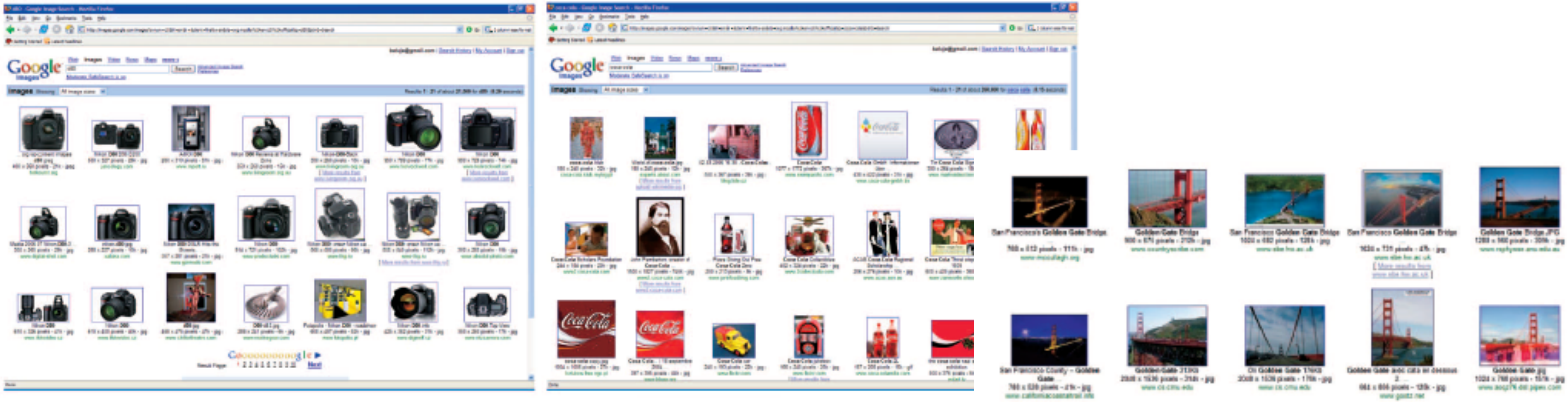
Image Similarity Search

- consider the case of search in web engines
  - most engines return image search matches based on
    - surrounding text on the page
    - image metadata
- could lead to incorrect results for mislabelled images &c

- can we do better than this?
  - should also match on similar images

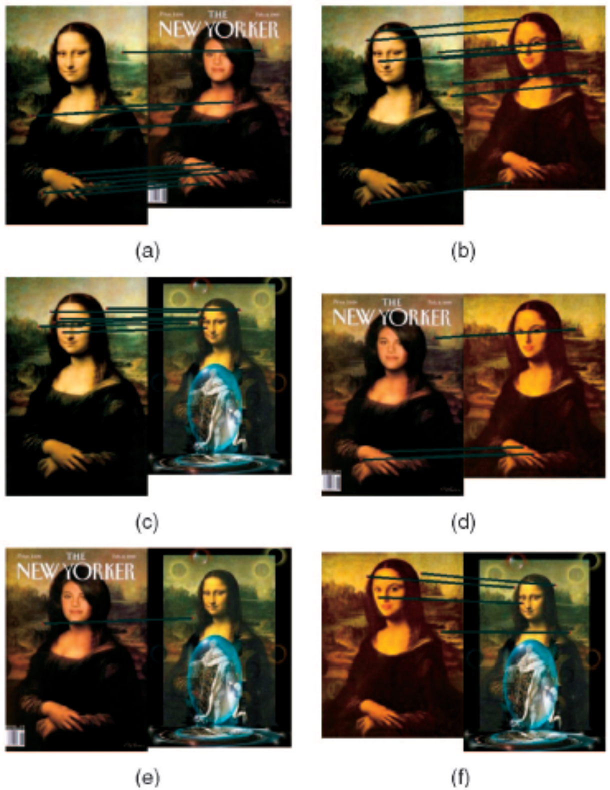
Google Image Search (VisualRank)

- uses PageRank for initial candidate results
- feature vectors extracted using SIFT (local features)



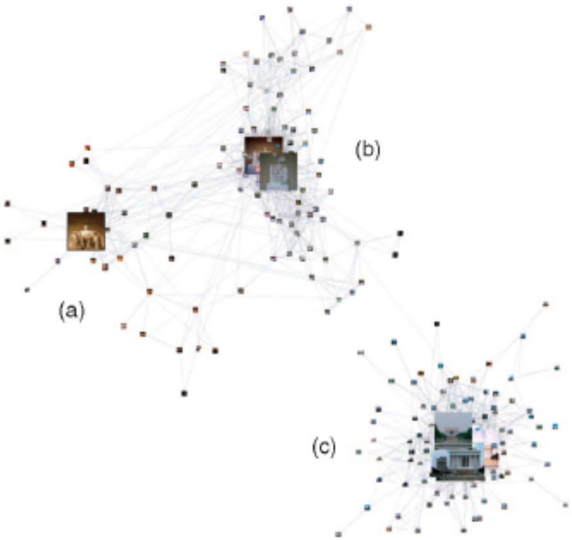
Google Image Search (VisualRank)

- clusters images based on similarity
  - measured using *p*-stable
  - Gaussian distribution
  - l_2 norm



Google Image Search (VisualRank)

- top results selected as graph center
  - eigenvector centrality measure



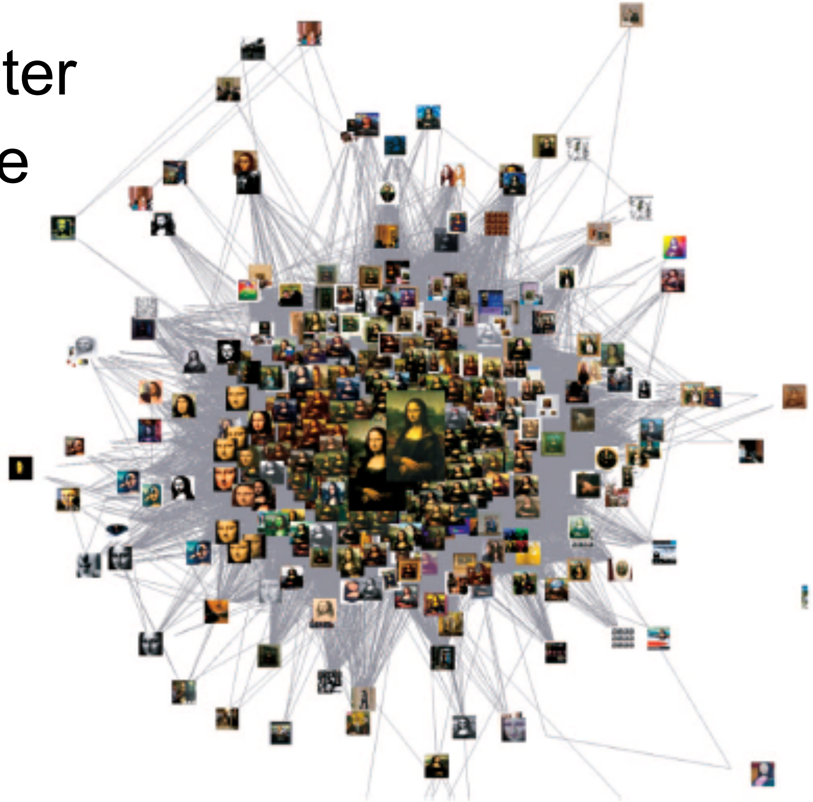


Image Similarity Search

- other methods have been proposed...

- chi^2 distance scheme
  - also based on *p*-stable
  - modified to use X^2 distance measure
  - similarity more accurate wrt/ global image descriptors
    - eg. color histograms (what's mostly used)

Image Similarity Search

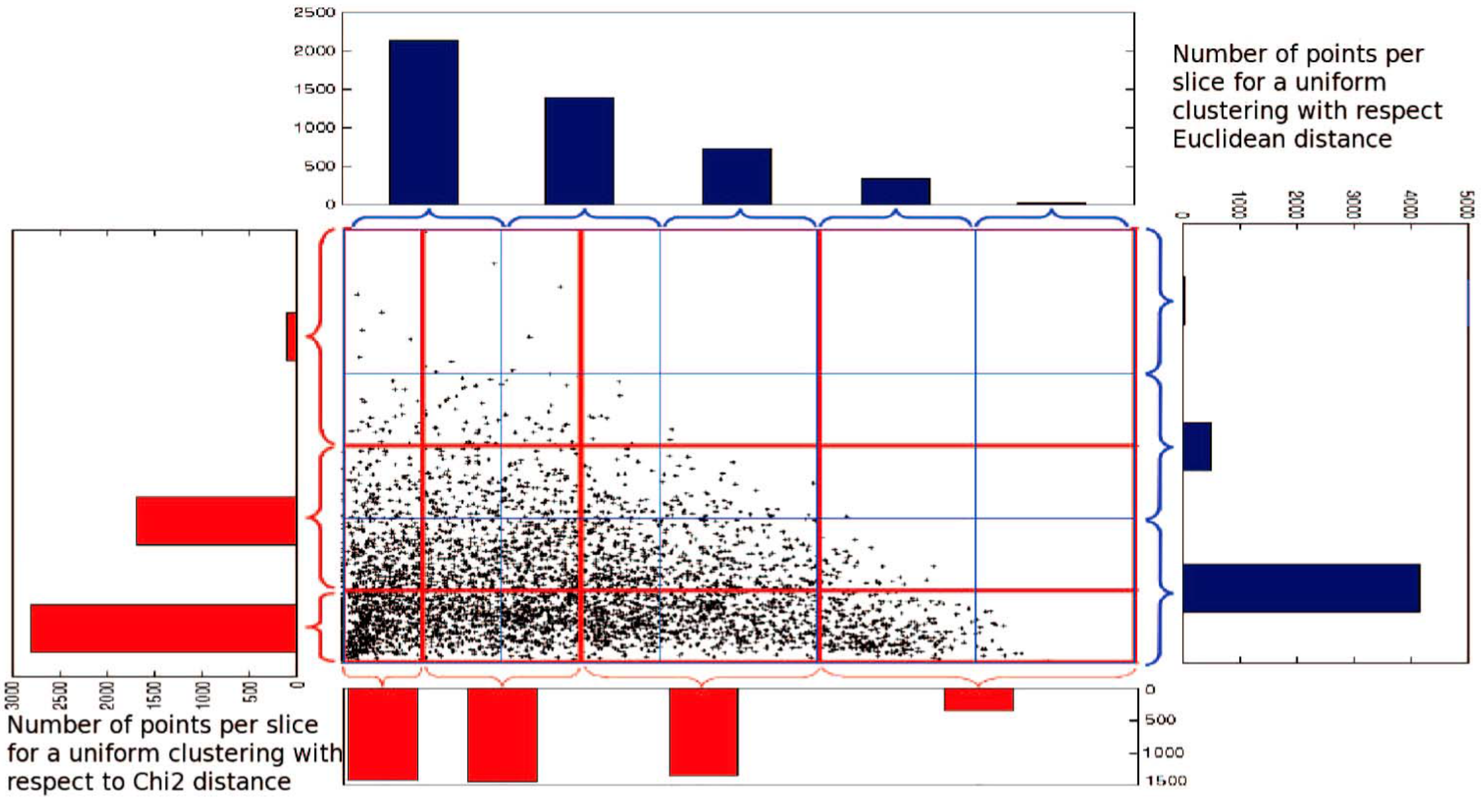
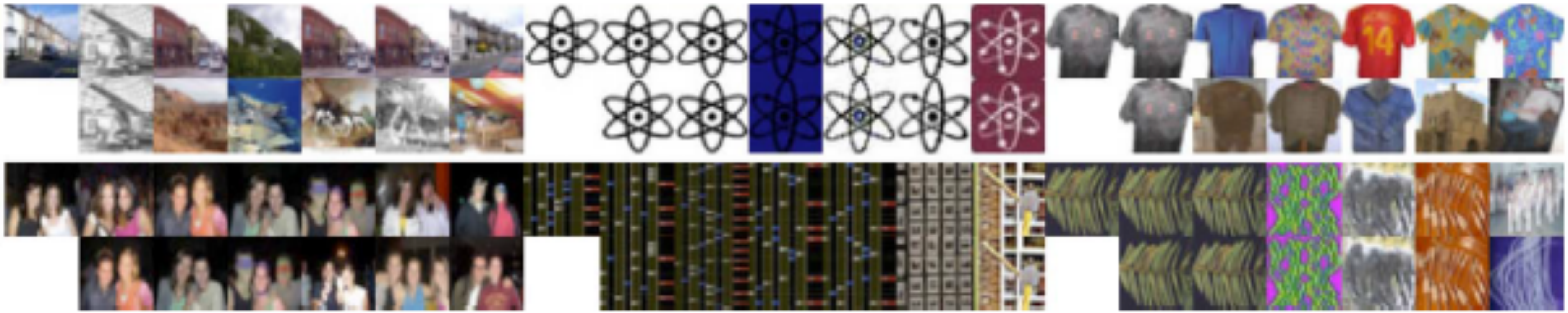


Image Similarity Search

- kernelized lsh (afaik)
  - constructed using kernel function (& some database items)
    - eg. gaussian blur, radial basis functions
    - method allows functions with unknown embeddings
  - given kernelized data & kernel function
    - need to use random hyperplane in kernel-induced feature space
    - construct hyperplane as weighted sum of random items
    - transform to change to normal distribution
    - which is used with the (modified) random hyperplane method

Image Similarity Search

- kernelized lsh (example)
  - 80 million images; extracting 384-dimensional vector
  - image → gist descriptor → Gaussian RBF Kernel
  - only .098% of all images searched



References

- Mayur Datar and Piotr Indyk. Locality-sensitive hashing scheme based on p-stable distributions. ACM Press, 2004.
- Yushi Jing and Shumeet Baluja. VisualRank: Applying PageRank to Large-Scale Image Search. 2008.
- Gorisse, D. and Cord, M. and Precioso, F. Locality-Sensitive Hashing for Chi2 Distance. 2012
- Kulis, B. and Grauman, K. Kernelized Locality-Sensitive Hashing. 2012
- Ullman, J. and Rajaraman, A. and Leskovec, J. Mining of Massive Datasets. 2010