Multimedia Indexing and Search Architecture

References

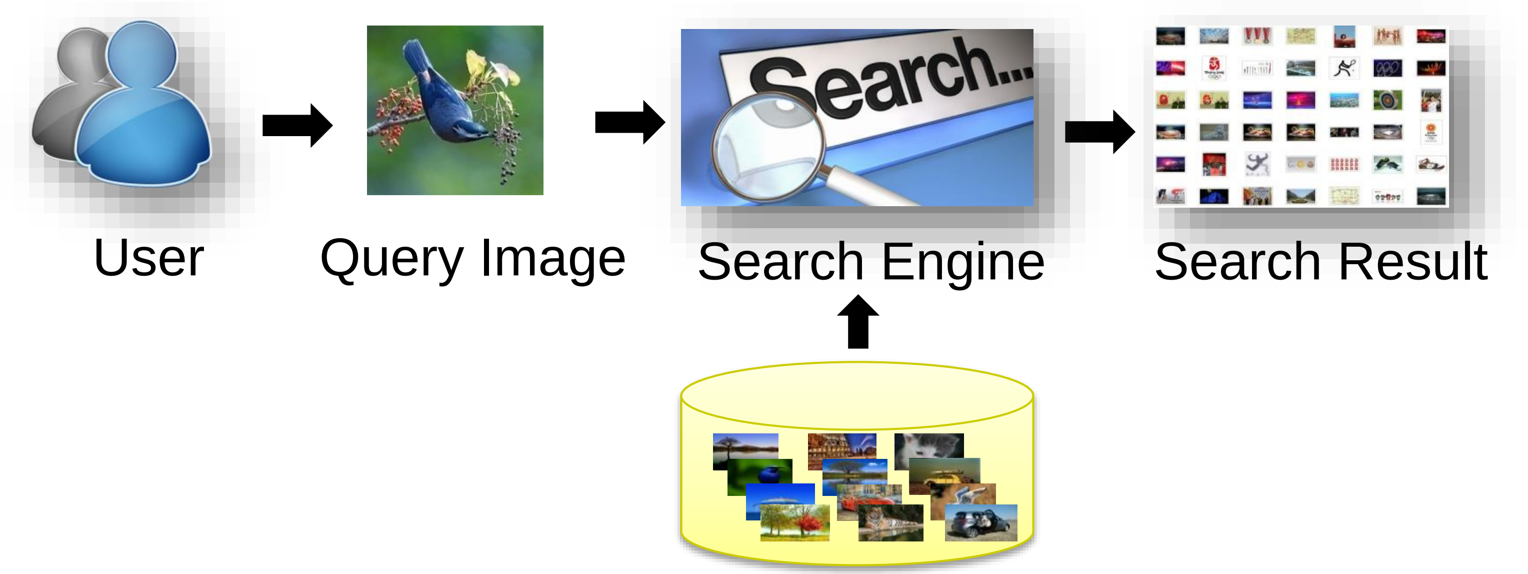
- KD-Tree: http://en.wikipedia.org/wiki/K-d_tree.
- J. Sivic & A. Zisserman (2003). Video Google: A Text Retrieval Approach to Object Matching in Videos. Int'l Conference on Computer Vision.
- A. Andoni & P. Indky (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of ACM, 51(1).
- S. Brin & L. Page (1998). The anatomy of a large-scale hypertextual web search engine. 19 pages

Contents

- Introduction to Image Indexing
- Tree-based Indexing
- Inverted File Indexing
- Hashing-based Indexing
- MM Indexing Strategy
- Google Search Architecture
- Summary

Introduction -1

- Pipeline of Content-based Image Retrieval



User

Query Image

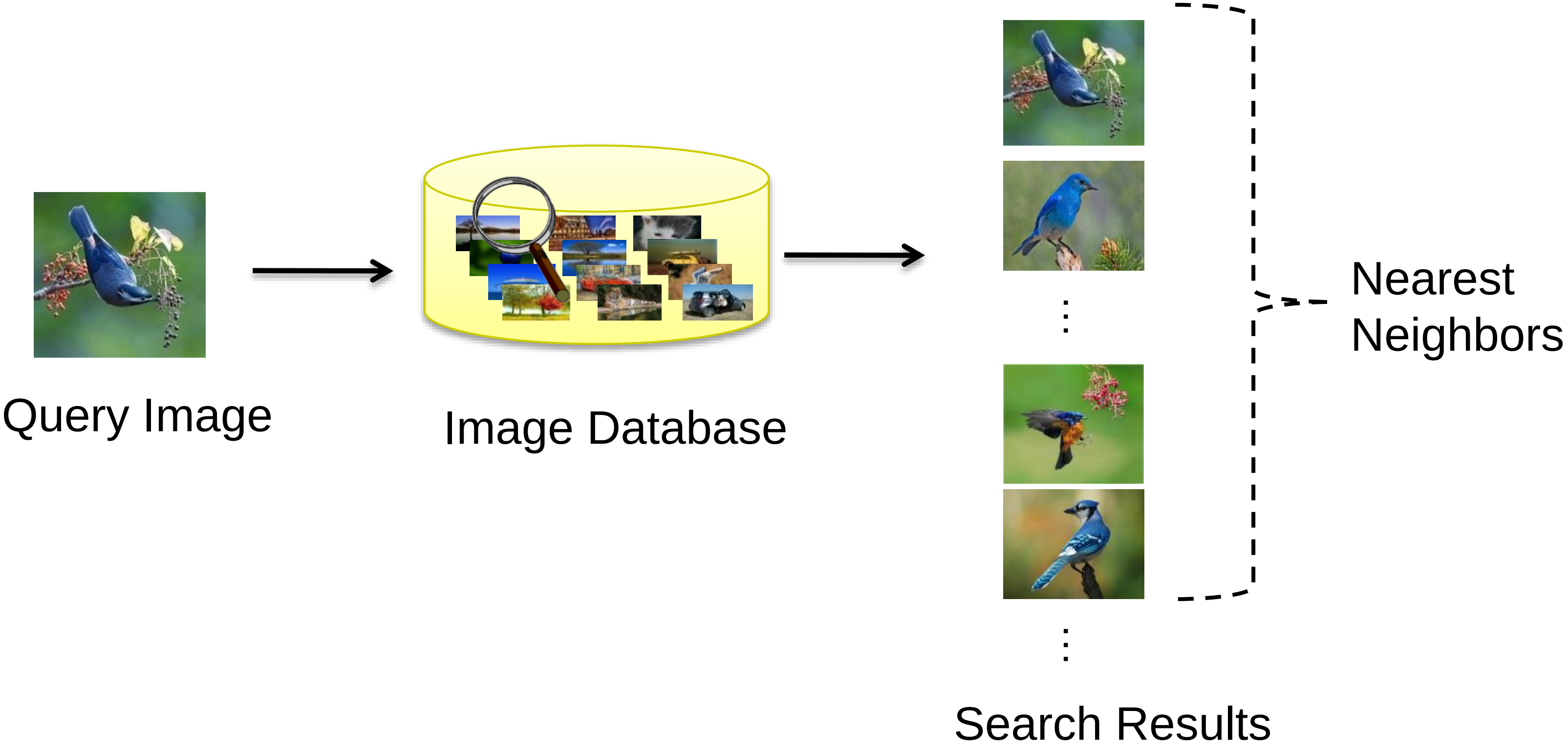
Search Engine

Search Result

Database

Introduction -2

- Search for similar images (nearest neighbors) within a database



Query Image

Image Database

Nearest Neighbors

Search Results

Introduction -3

- How to find similar images?
  - Naive approach: Linear Scan

    Compute the similarities of query image to all the images in the database based on visual features, and then find the similar ones.

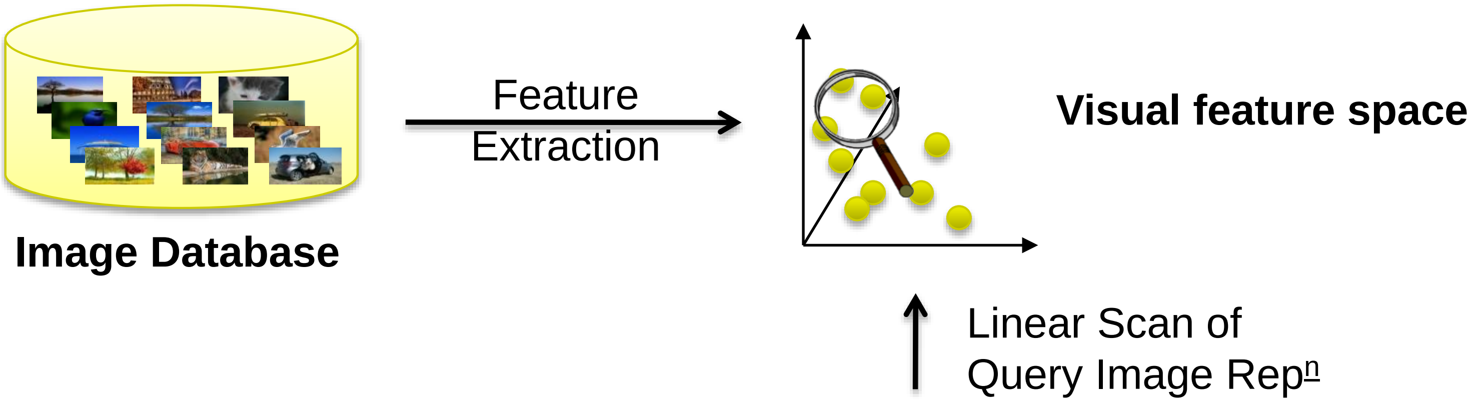


Image Database

Feature Extraction

Visual feature space

Linear Scan of Query Image Rep^{n}

  - Key Limitation:
    - Linear scan becomes very slow for large-scale database (e.g., millions of images).
  - Solution: Indexing!

Introduction -4

> *"If I had eight hours to chop down a tree, I'd spend six sharpening my ax."*
>
> *Abraham Lincoln*

- Indexing
  - Index database images into one/more index tables in advance
  - Perform search over index tables
  - Improve the speed of search

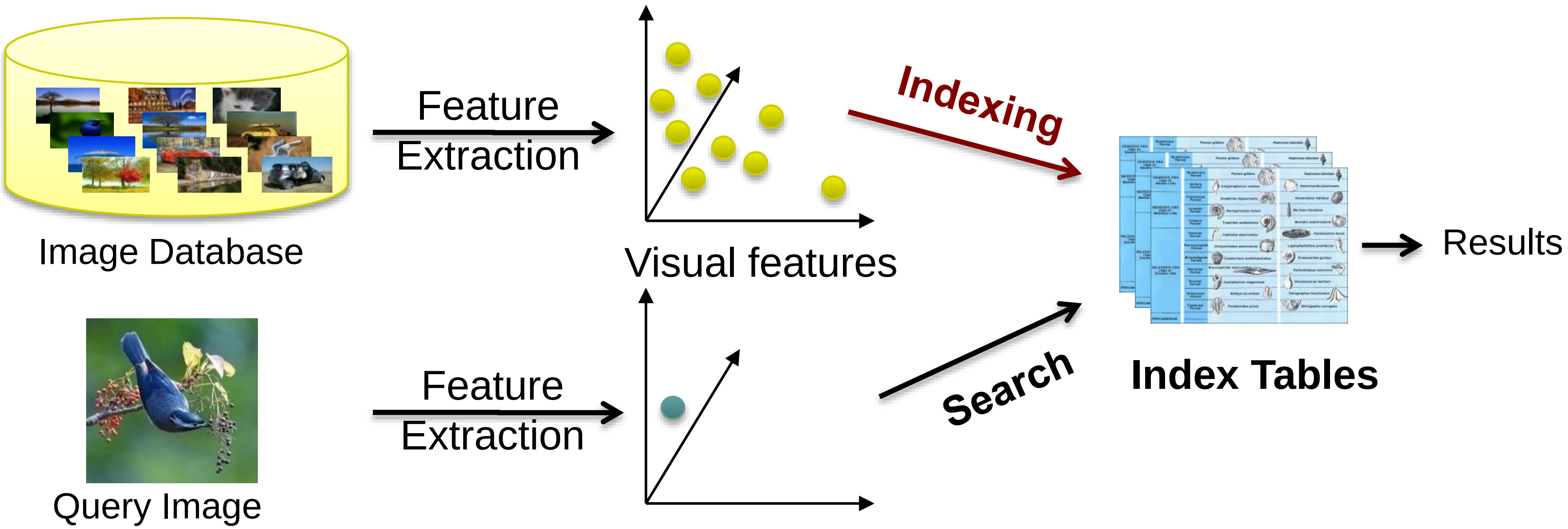


Image Database

Feature Extraction

Visual features

Indexing

Index Tables

Results

Query Image

Feature Extraction

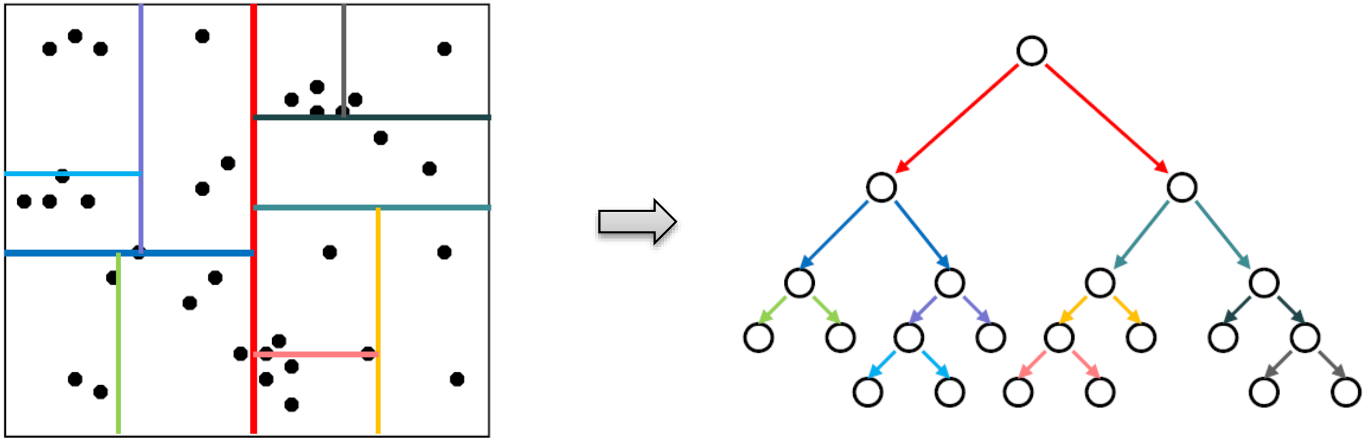
Search

Contents

- Introduction to Image Indexing
- Tree-based Indexing
- Inverted File Indexing
- Hashing-based Indexing
- MM Indexing Strategy
- Google Search Architecture
- Summary

Tree-based Indexing

- Split feature space using spatial partitions and recursive hyperplane decomposition, resulting in a tree structure
- Sort database samples in the leaf nodes
- One example is KD-Tree, K-Dimensional Tree, which is widely used in early image systems, e.g. IBM QBIC



2-d features

KD-Tree -1

- A binary tree storing *k*-dimension database samples in its leaf nodes
- Recursively partitions the samples into axis-aligned cells, dividing the samples approximately in half by a line perpendicular to one of the *k* coordinate axes
- Division strategies – How to Choose the next axis to split?
  - Cycle through the axes in order
  - Or choose the axis that has the largest variance among the database points

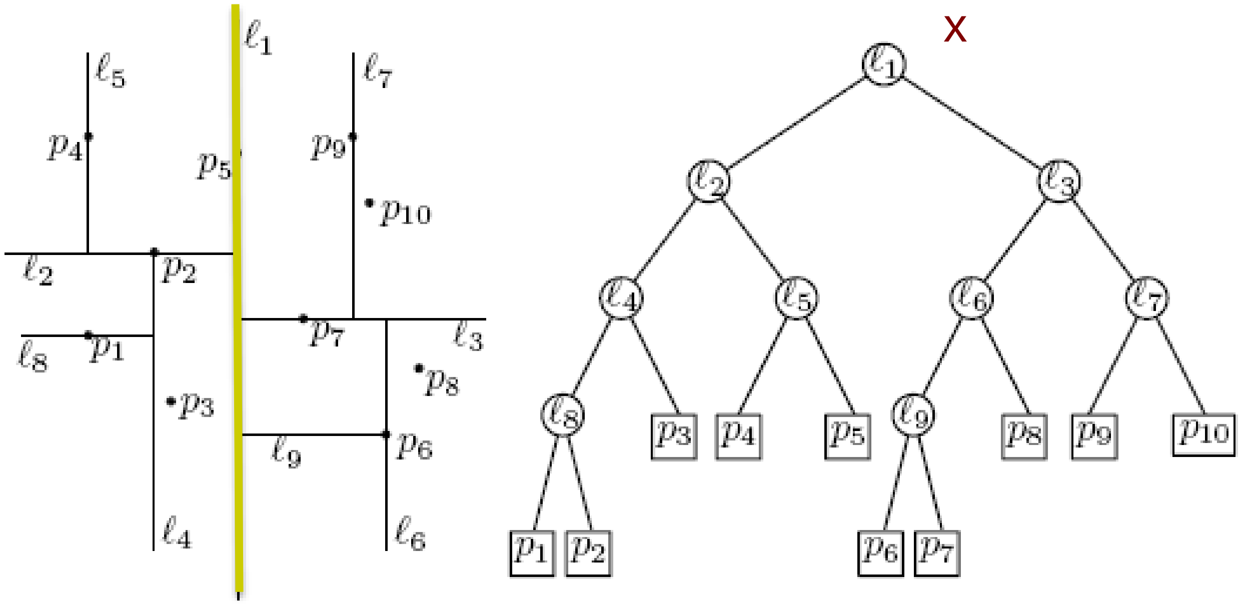
  - Cycling through the axes in order is widely used in practice

KD-Tree -2

- KD Tree construction (Cycle through the axes in order)
- As an example, we show the construction of kd tree in 2-dimension
- Division strategy
  - Split by x-coordinate*:* split by a vertical line that has (ideally) half the points left or on, and half right.
  - Split by y-coordinate*:* split by a horizontal line that has (ideally) half the points below or on and half above.
  - Typically choose the Medium point for splitting

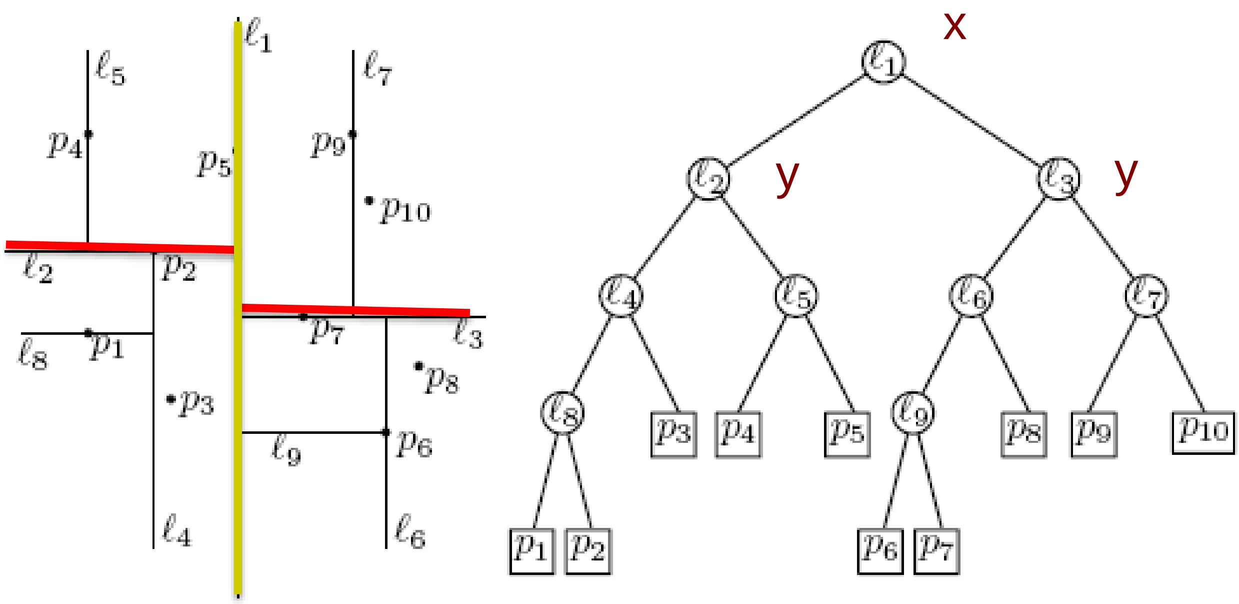
KD-Tree –Example -1

Split by x-coordinate: split by a vertical line that has approximately half the points left or on, and half right.



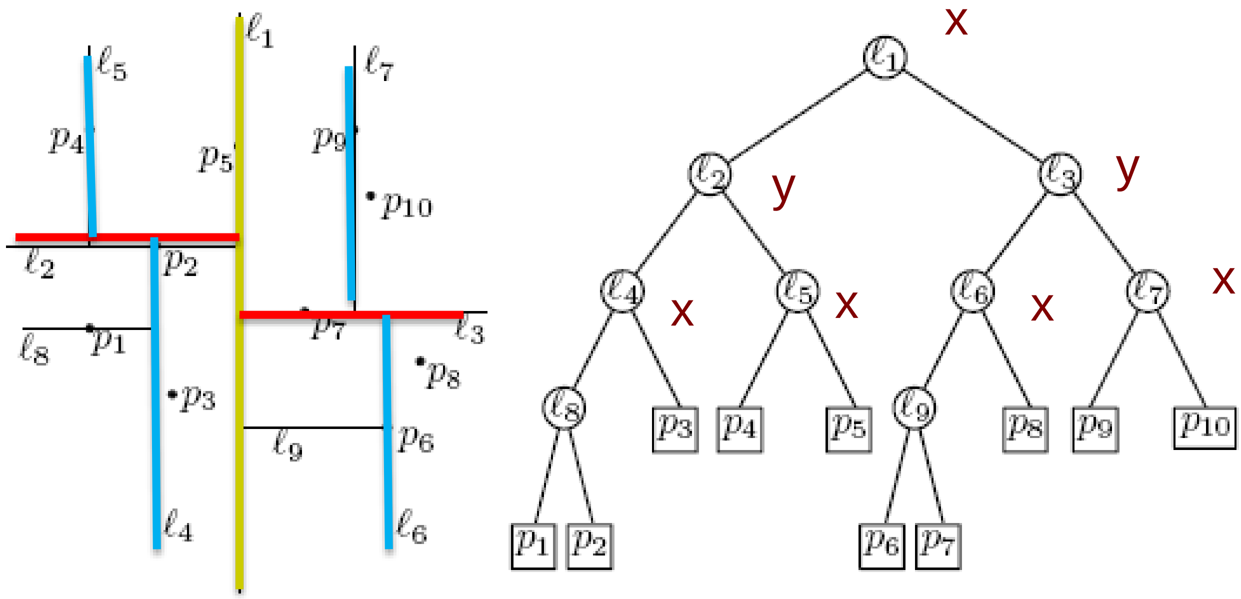
KD-Tree –Example -2

Split by y-coordinate*:* split by a horizontal line that has half the points below or on and half above.



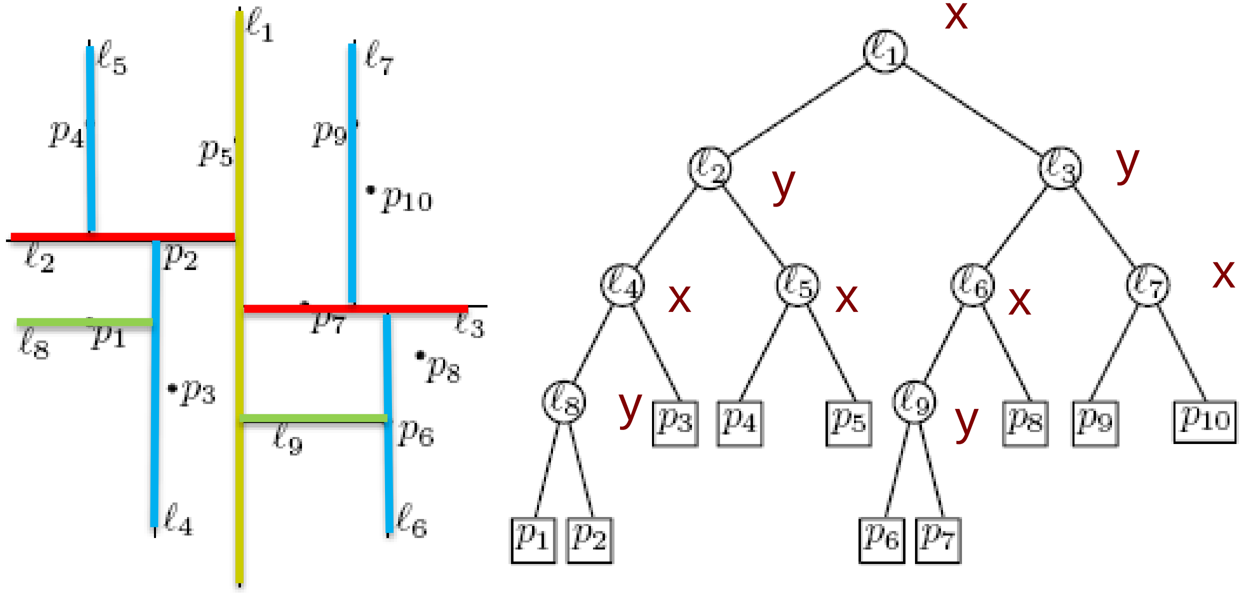
KD-Tree –Example -3

Split by x-coordinate: split by a vertical line that has half the points left or on, and half right.



KD-Tree –Example -4

Split by y-coordinate: split by a horizontal line that has half the points below or on and half above.



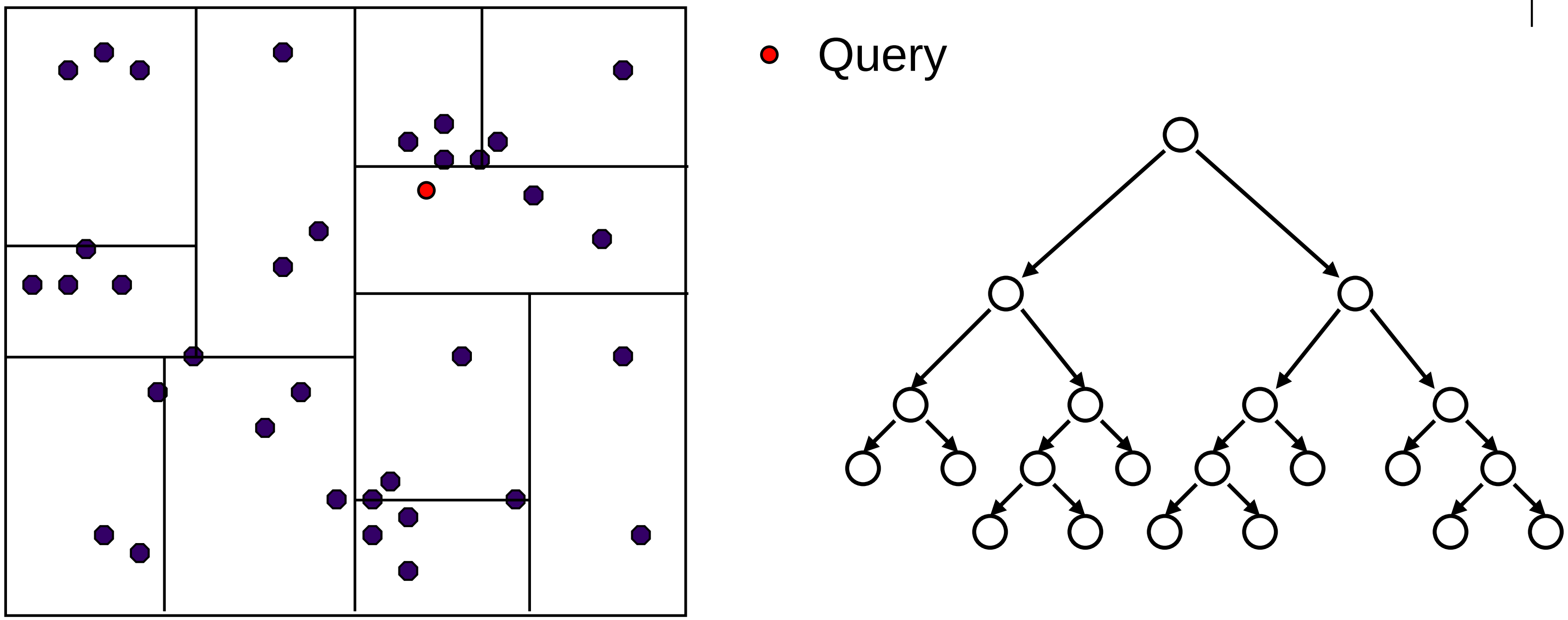
KD-Tree -3

- After the construction, for each node, its left subtree holds all the samples that are less than (or equal to) the node along the splitting axis;
- its right subtree holds all the samples that are larger.

Two key decisions:

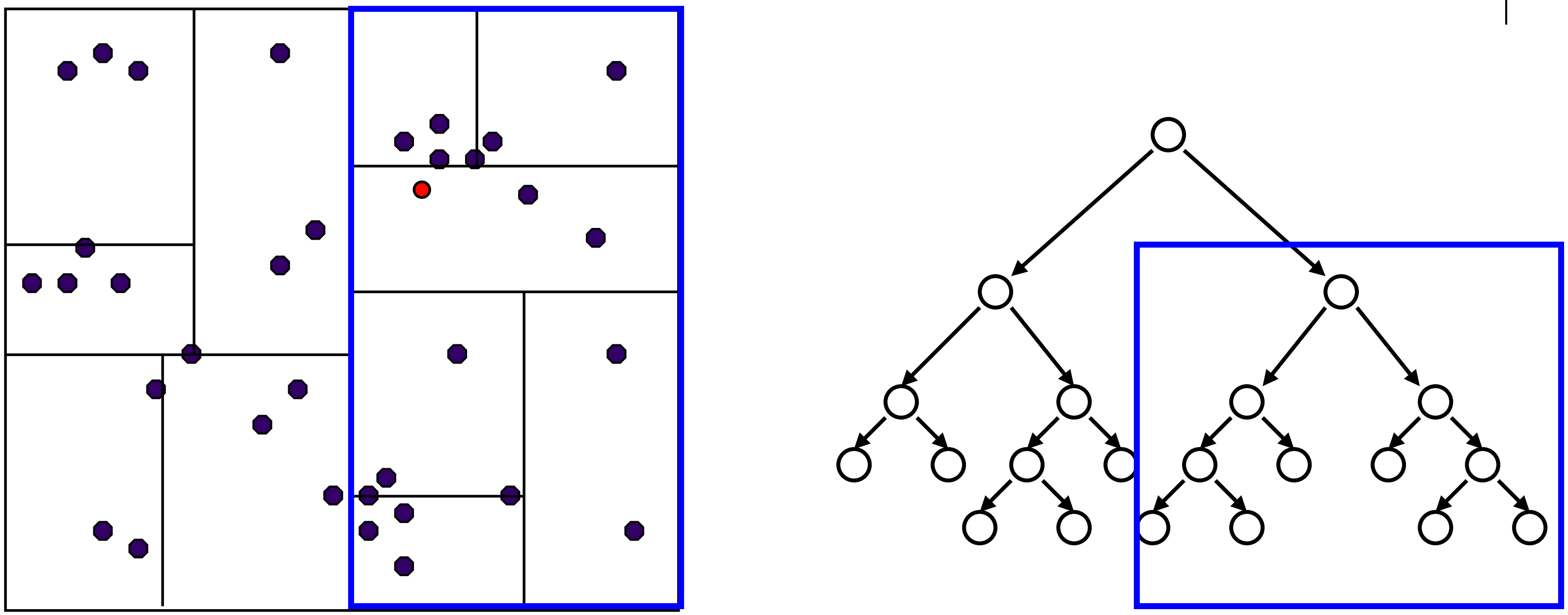
- How to select roots of tree/sub-trees?
  - Chose the point that splits the points in the middle, May use Medium of the points
  - This will lead to a balance tree
- When to end splitting?
  - If a node had no children, the splitting is not required.

Nearest Neighbors Search with KD-Tree



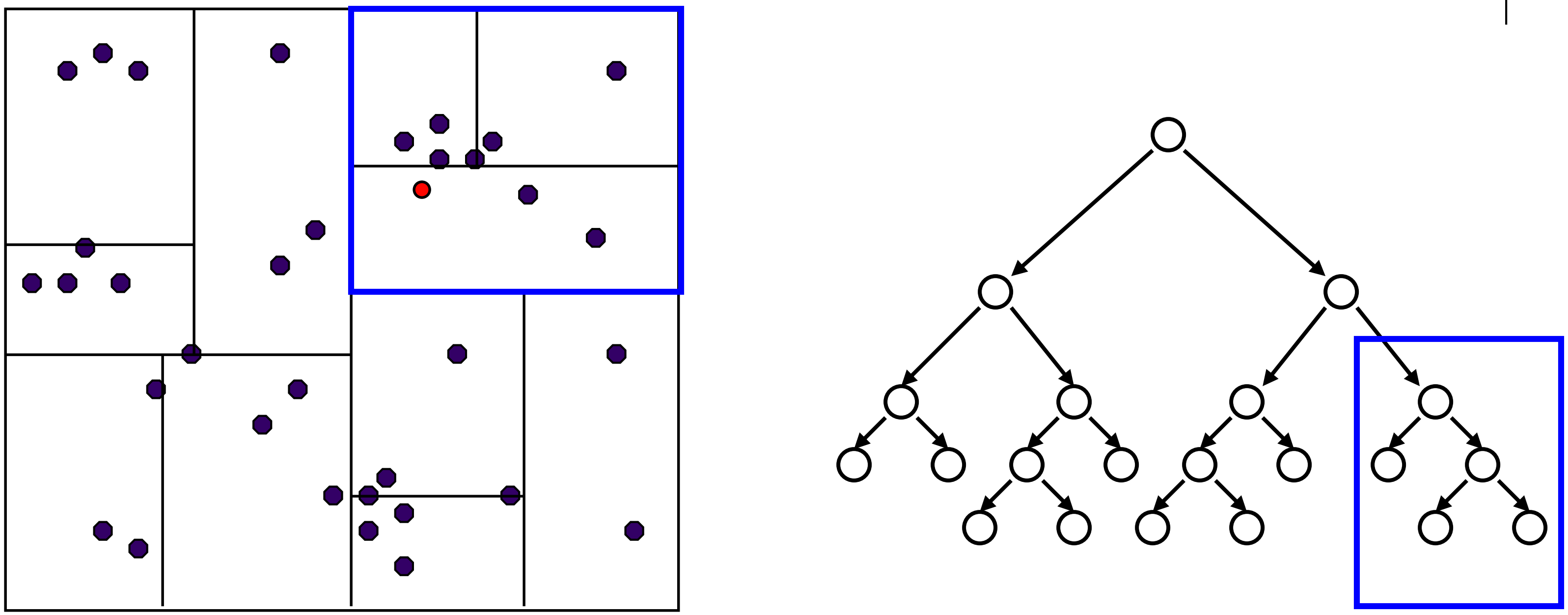
- Examine nearby points first: Traverse the tree, looking for the rectangle that contains the query.

Nearest Neighbors Search with KD-Tree



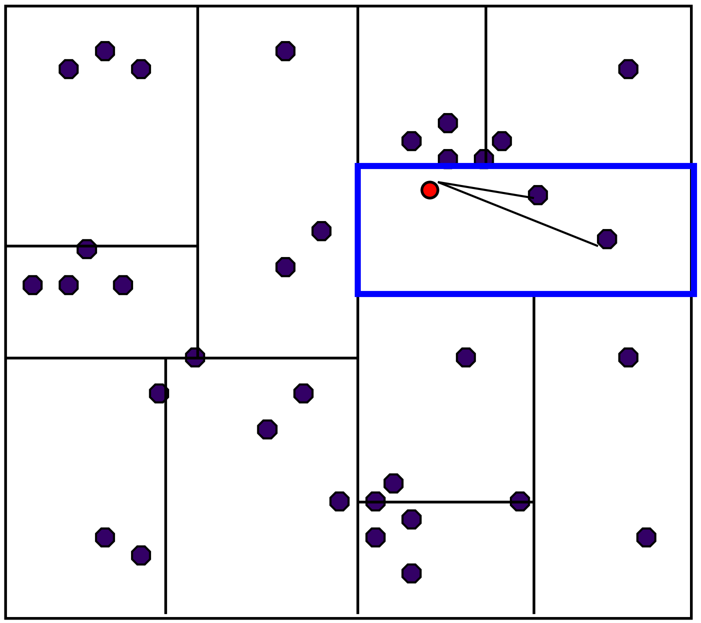
- Explore the branch of the tree that is closest to the query point first.

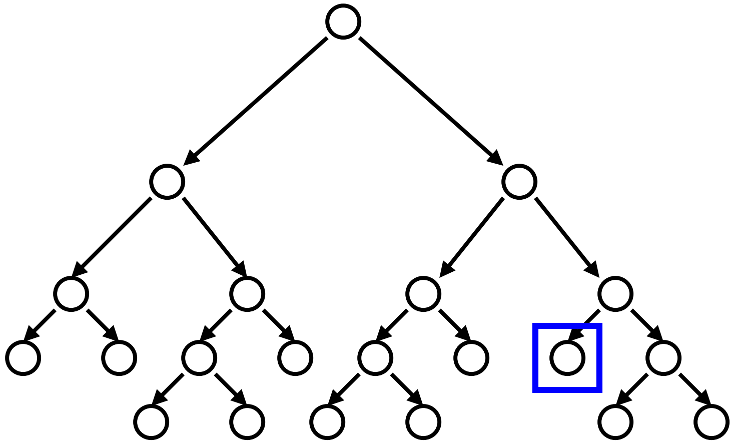
Nearest Neighbors Search with KD-Tree



- Explore the branch of the tree that is closest to the query point first.

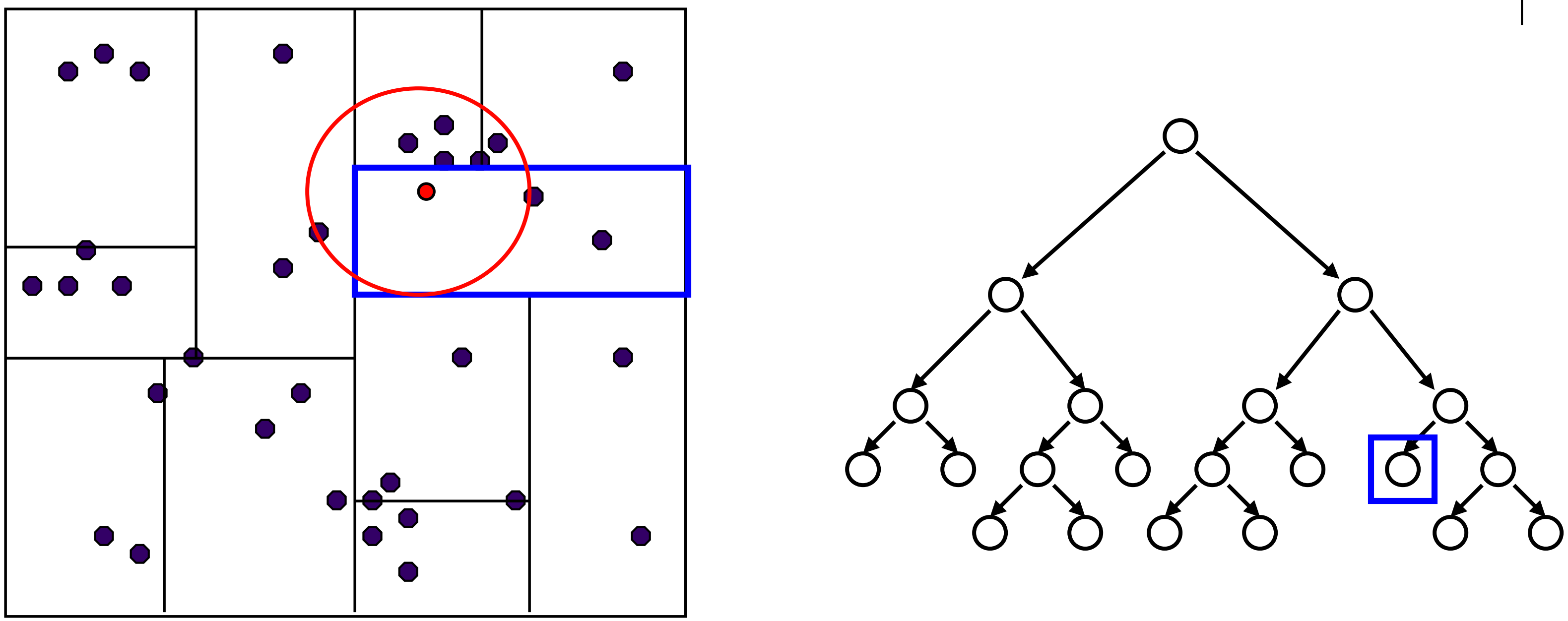
Nearest Neighbors Search with KD-Tree





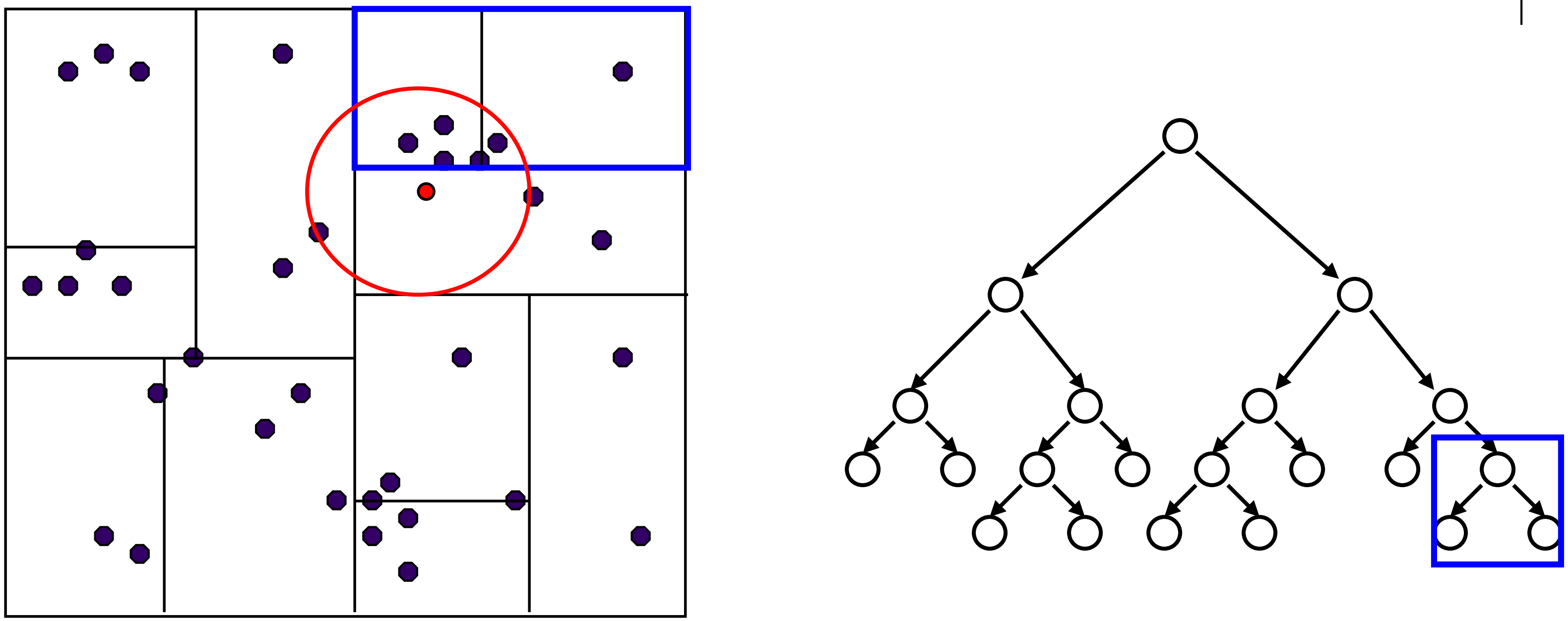
- When we reach a leaf, it saves that node point as the "current best”.
- It then computes the distance to each point in the node; and maintain the kNN distance d_n (the distance that covers the kNN points)

Nearest Neighbors Search with KD-Tree



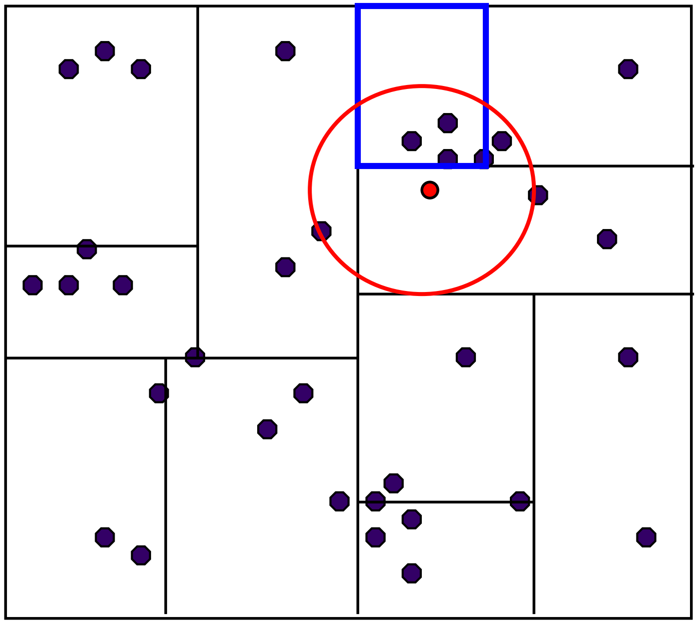
- The algorithm unwinds the tree by checking whether there could be any points on the other splitting planes that are closer than the kNN point.
- This can be done by intersecting the splitting hyperplane with a hypersphere (of radius d_n) around the search point

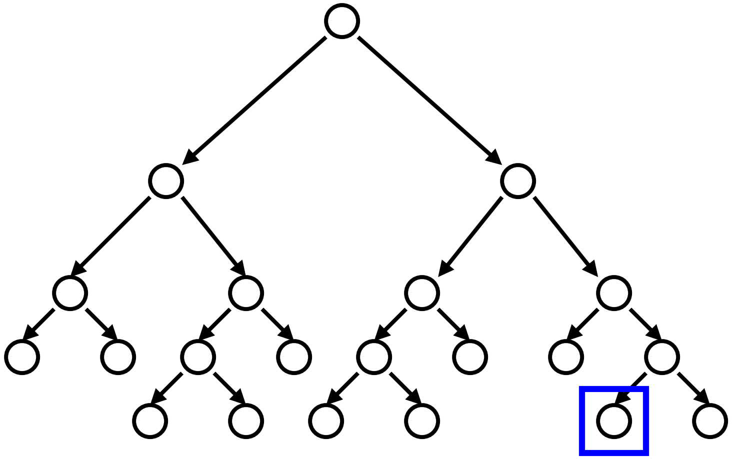
Nearest Neighbors Search with KD-Tree



- If the hypersphere crosses the plane, there could be nearer points on the other side of the plane;
- Otherwise, the algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.

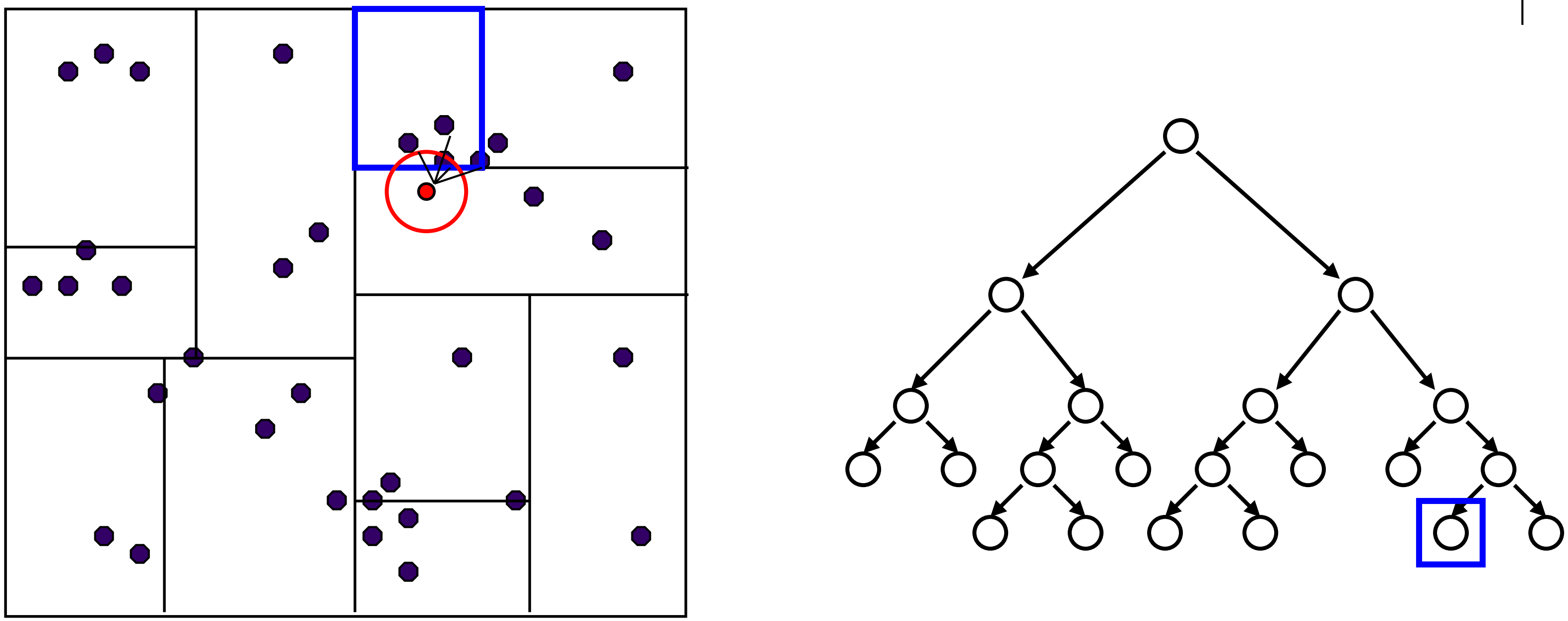
Nearest Neighbors Search with KD-Tree





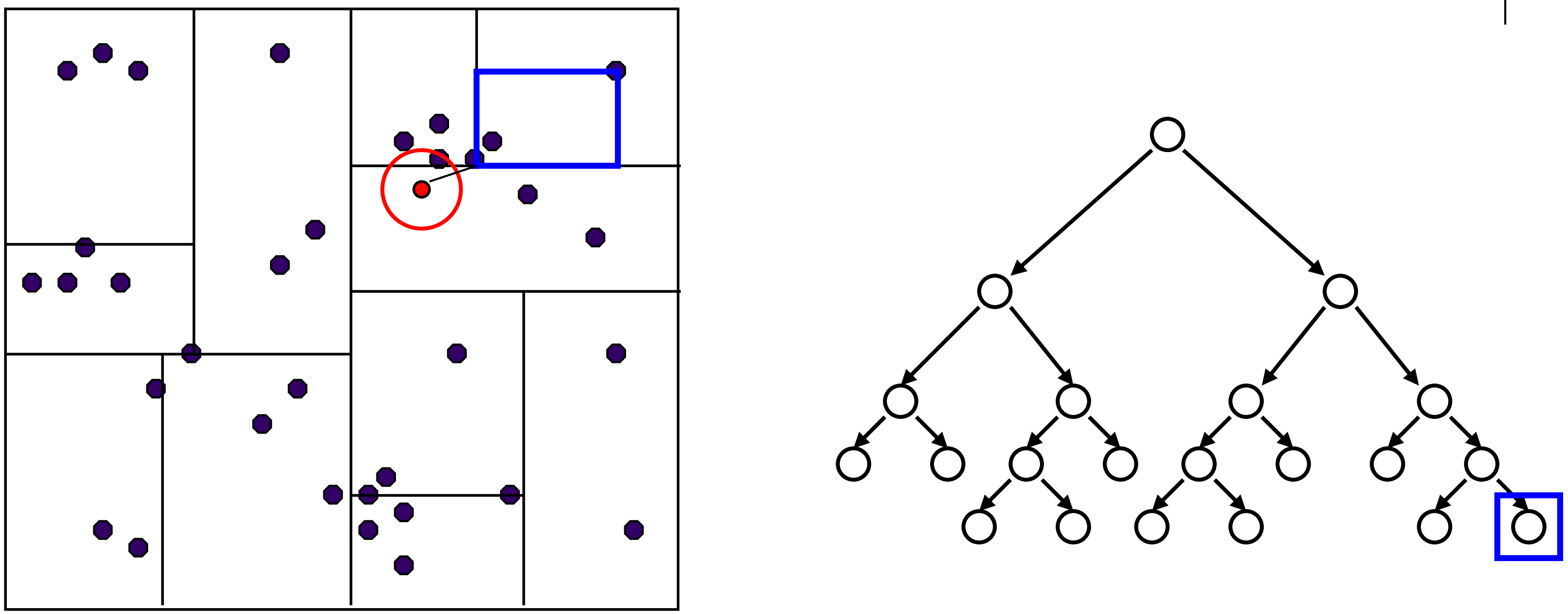
- Each time a new closest node is found, we can update the distance bound for kNN, d_n

Nearest Neighbors Search with KD-Tree



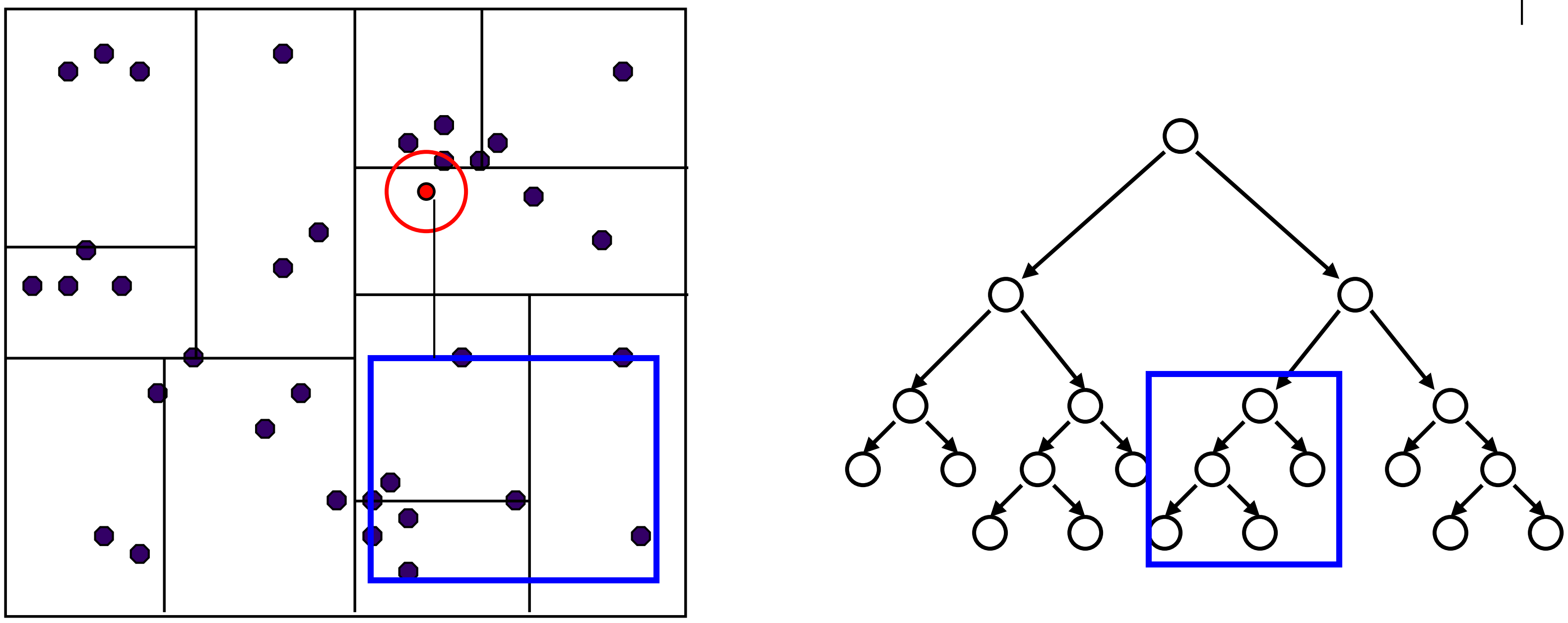
- Each time a new closest node is found, we can update the kNN distance bound, d_n

Nearest Neighbors Search with KD-Tree



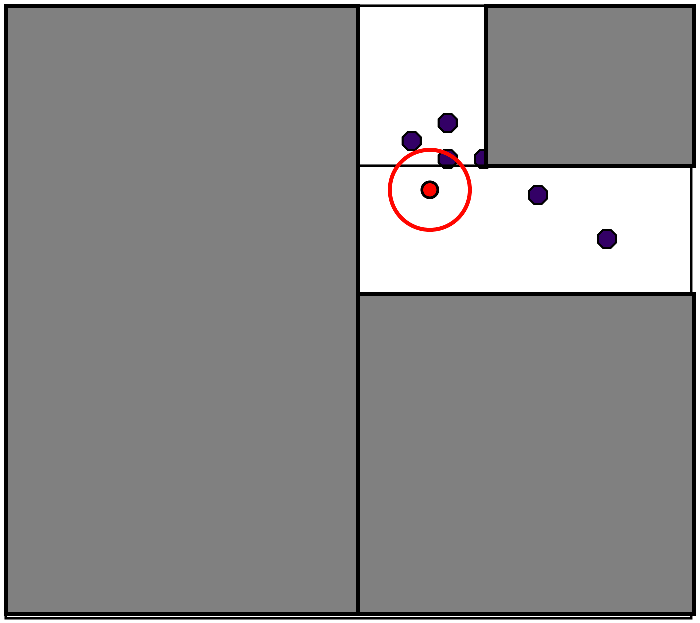
- Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

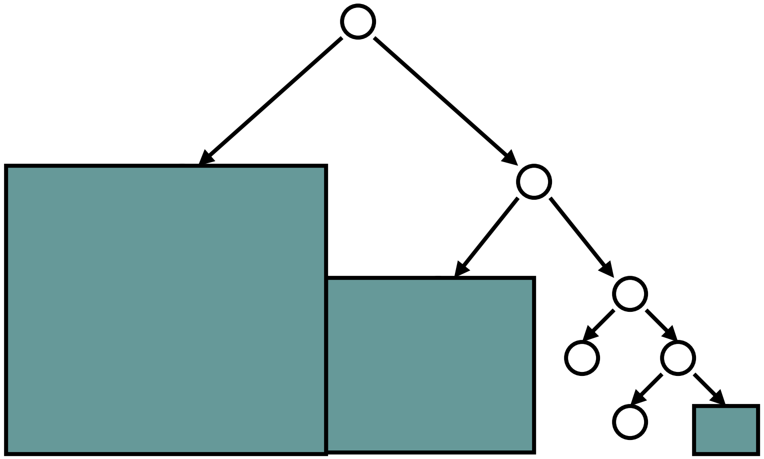
Nearest Neighbors Search with KD-Tree



- Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Nearest Neighbors Search with KD-Tree





- Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

KD-Tree -4

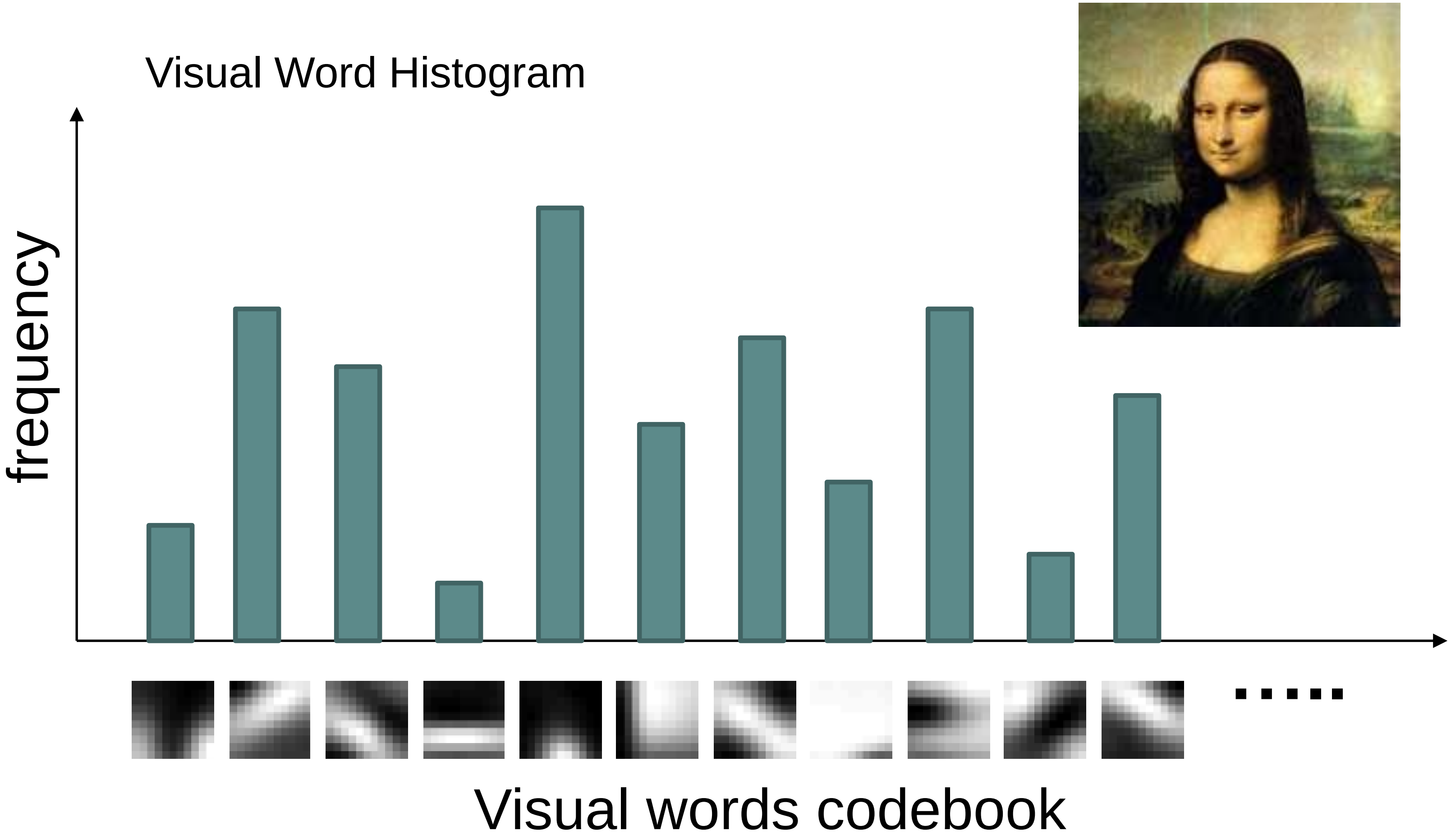
- KD-Tree is very effective for low-dimensional data (i.e., fewer than 10 dimensions)
- It can be extended by providing the *k* nearest neighbours to a point by maintaining *k* current bests instead of just one.
- It may not be effective for high-dimensional data, like visual features:
  - Need to visit many more branches during the backtracking stage
  - Degrades to worst case linear san performance in practice.

Contents

- Introduction to Image Indexing
- Tree-based Indexing
- Inverted File Indexing
- Hashing-based Indexing
- MM Indexing Strategy
- Google Search Architecture
- Summary

Recall Bag-of-Visual Words

- Summarize the entire image based on its distribution (histogram) of visual word occurrences.

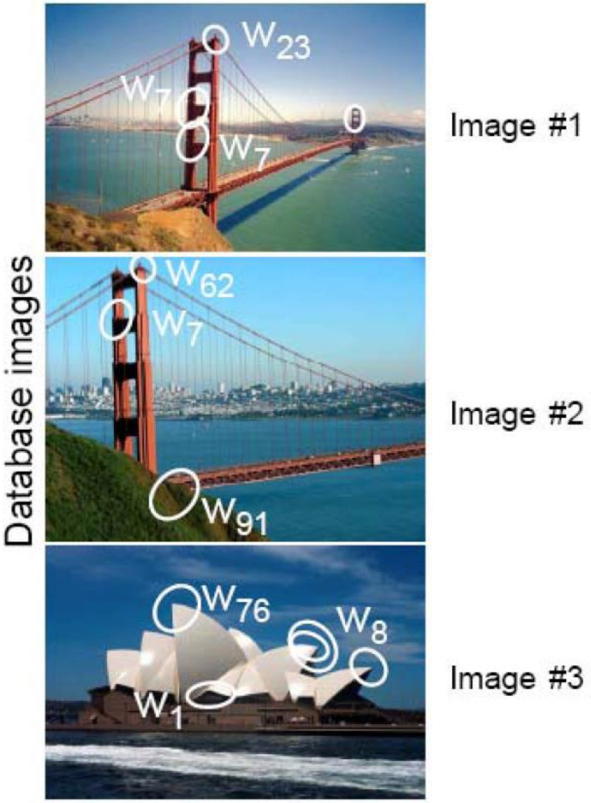


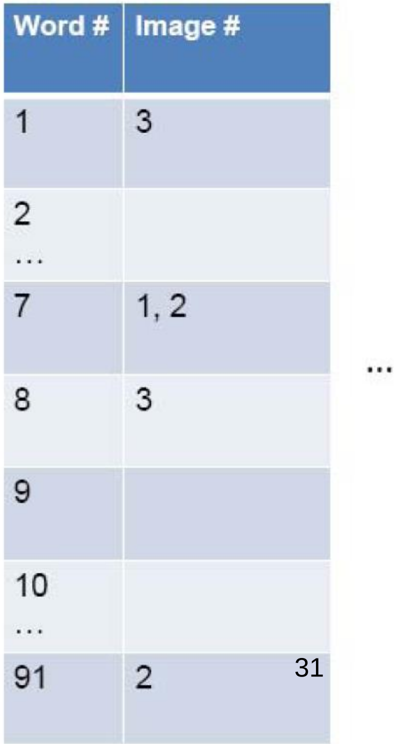
Indexing of Visual Words -1

- As the Bag-of-Visual-Words model quantizes the feature space into discrete "visual words", we can index images easily with an *inverted file*

→ Build an inverted index of all *images* based on *occurrences of visual word*

- Database images are loaded into the index mapping words to image numbers



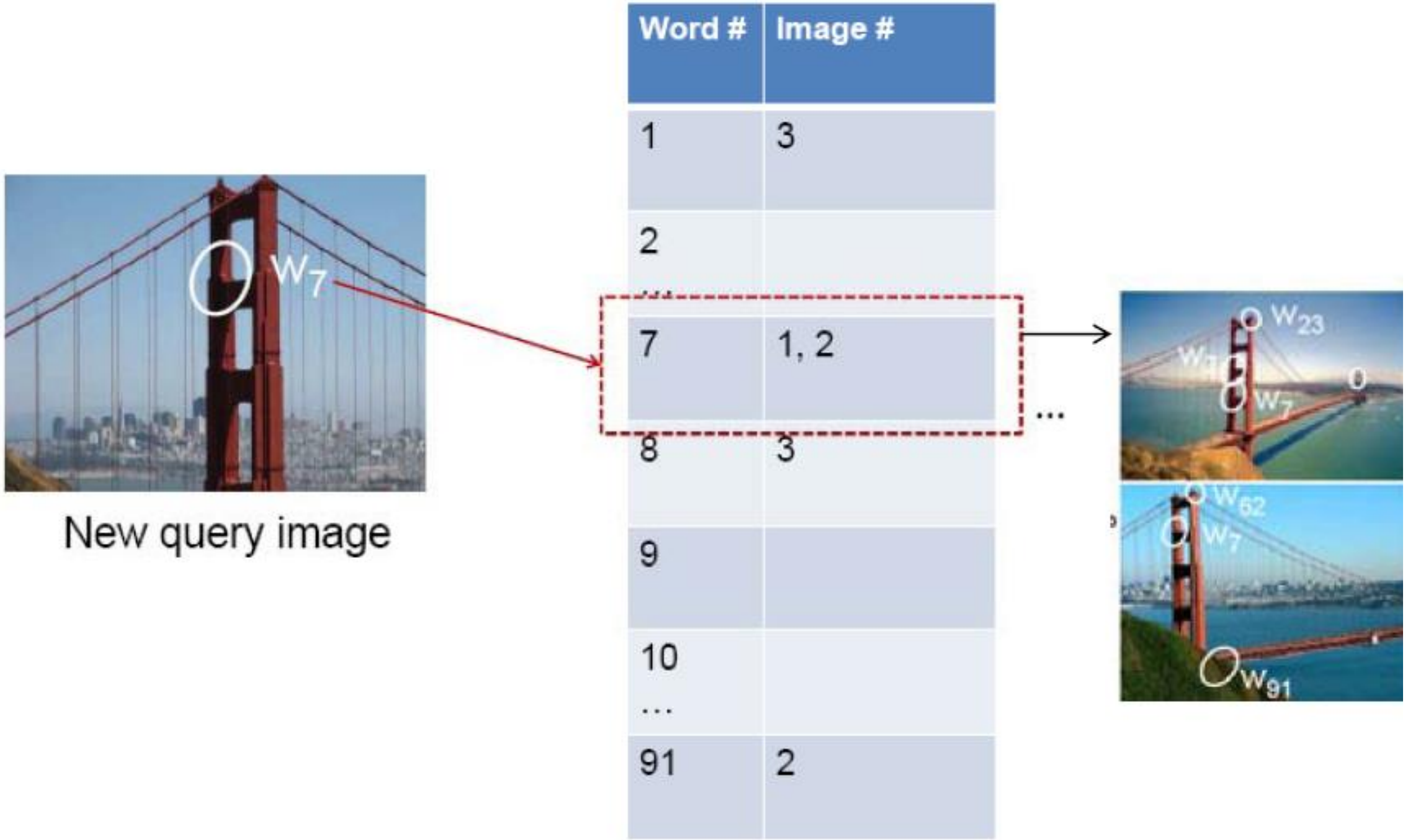


Word #	Image #
1	3
2 …	
7	1, 2
8	3
9	
10 …	
91	2

…

Indexing of Visual Words -2

- Extract visual words in query image
- Map query image to the indices of database images that share a word



New query image

Word #	Image #
1	3
2 …	
7	1, 2
8	3
9	
10 …	
91	2

Indexing of Visual Words -3

- Searching with inverted file

Input: A query image *q* represented by visual words,

suppose *q* has *L visual words*

Access To inverted file

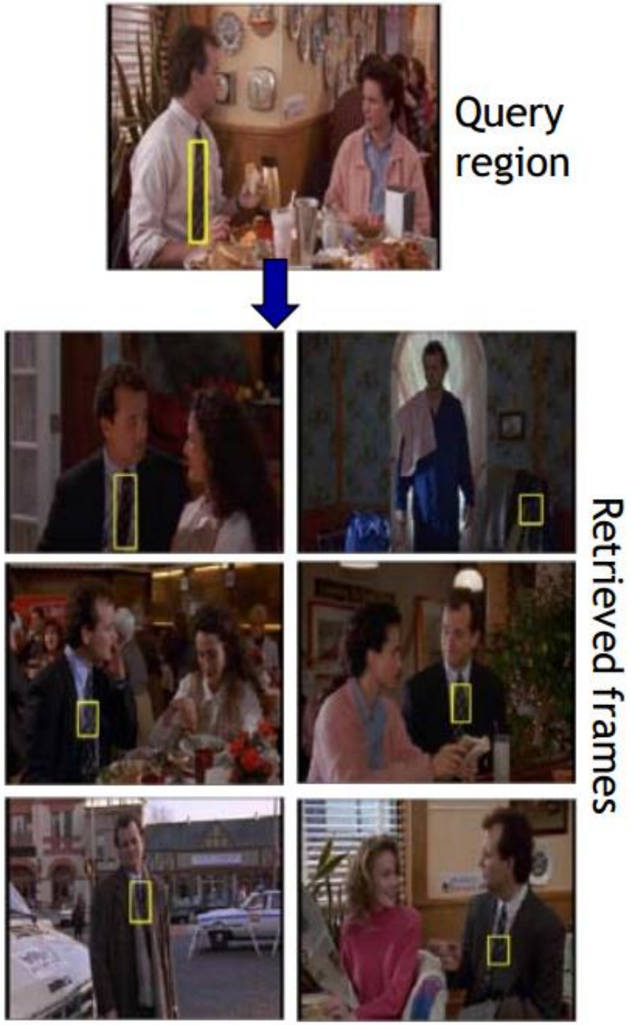
Output: a set of images *S* have the same visual words as the query image

$S = \cap_{i = 1, \ldots, L} \{ \text{list}_i \}$, where list_i is the corresponding index list of images that contain *i*-th visual word.

Indexing of Visual Words -4

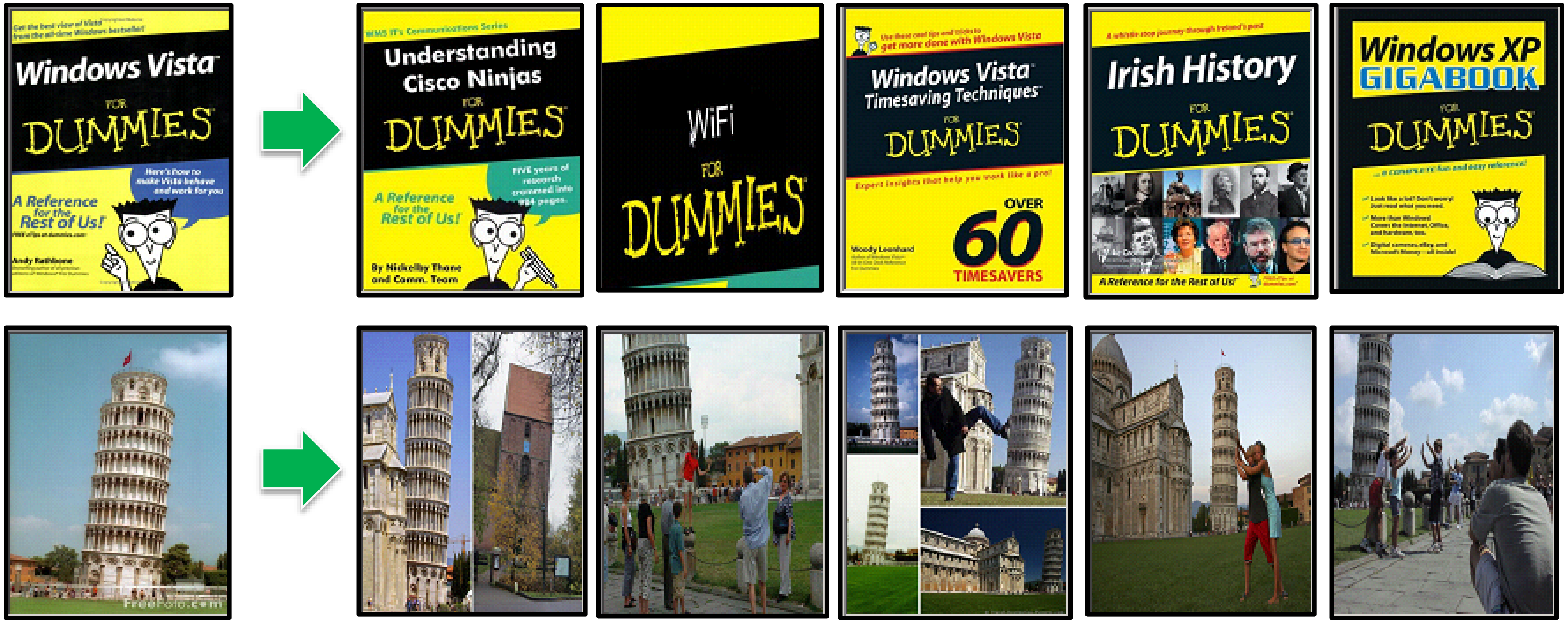
- Collect all words with query region
- Inverted file index to find relevant images (frames)
- Compare word counts
- Spatial verification

- Sivic & Zisserman, ICCV 2003
- Demo: http://www.robots.ox.ac.uk/~vgg/research/vgoogle/index.html



Indexing of Visual Words -5

Other examples of search results


Windows Vista FOR DUMMIES
Understanding Cisco Ninjas FOR DUMMIES
WiFi FOR DUMMIES
Windows Vista Timesaving Techniques FOR DUMMIES
OVER 60 TIMESAVERS
Irish History FOR DUMMIES
Windows XP GIGABOOK FOR DUMMIES

Query

Search results

Indexing of Visual Words -5

- Although it shows encouraging performance, a fundamental difference between an image/video query and a text query limits its usefulness

➔ An image query usually contain more than thousands of visual words, while a text query is usually of 3-5 terms.

  - This results in high computation cost and long query time

- Possible solutions:
  - Remove visual stop words from the query
  - Perform feature selection to select important visual words

Contents

- Introduction to Image Indexing
- Tree-based Indexing
- Inverted File Indexing
- Hashing-based Indexing
- MM Indexing Strategy
- Google Search Architecture
- Summary

Hashing-based Indexing -1

- Basic Idea:
  - Hash high-dimensional data into a low-dimensional Hamming space based on a family of hash functions
  - Hash similar samples into the same bucket based on a family of hash functions

Hamming space is the set of all 2^L binary strings of length L

Hamming distance between two equal length binary strings is the number for which the bits are different.

$$||10\textcolor{orange}{1}1\textcolor{orange}{1}01, 10\textcolor{orange}{0}1\textcolor{orange}{0}01||_H = 2$$

$$||111\textcolor{orange}{0}101, 111\textcolor{orange}{1}101||_H = 1$$

Hashing-based Indexing -2

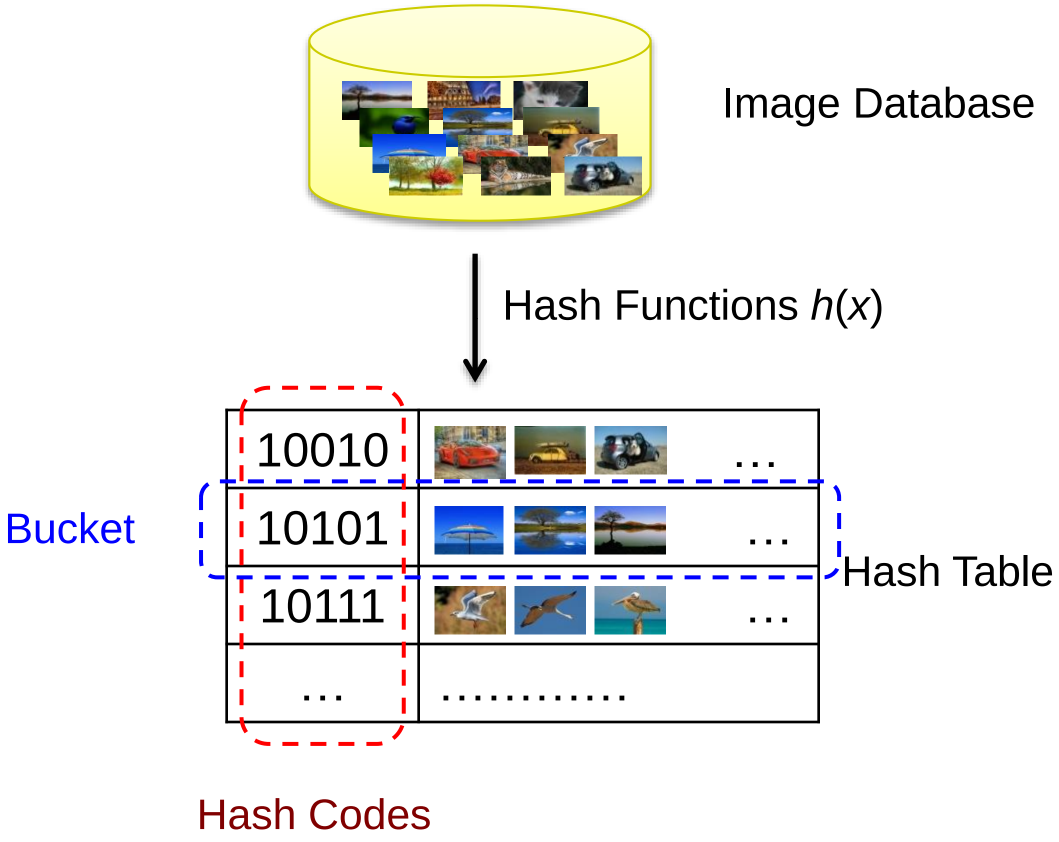


Image Database

Hash Functions *h*(*x*)

10010 …

10101 …

10111 …

… …………

Bucket

Hash Table

Hash Codes

Hashing-based Indexing -3

- Search with Hashing:
  - Given a query, only the samples in the same or adjacent buckets needs to be exhaustively searched
  - Much more efficient than scanning over the entire database

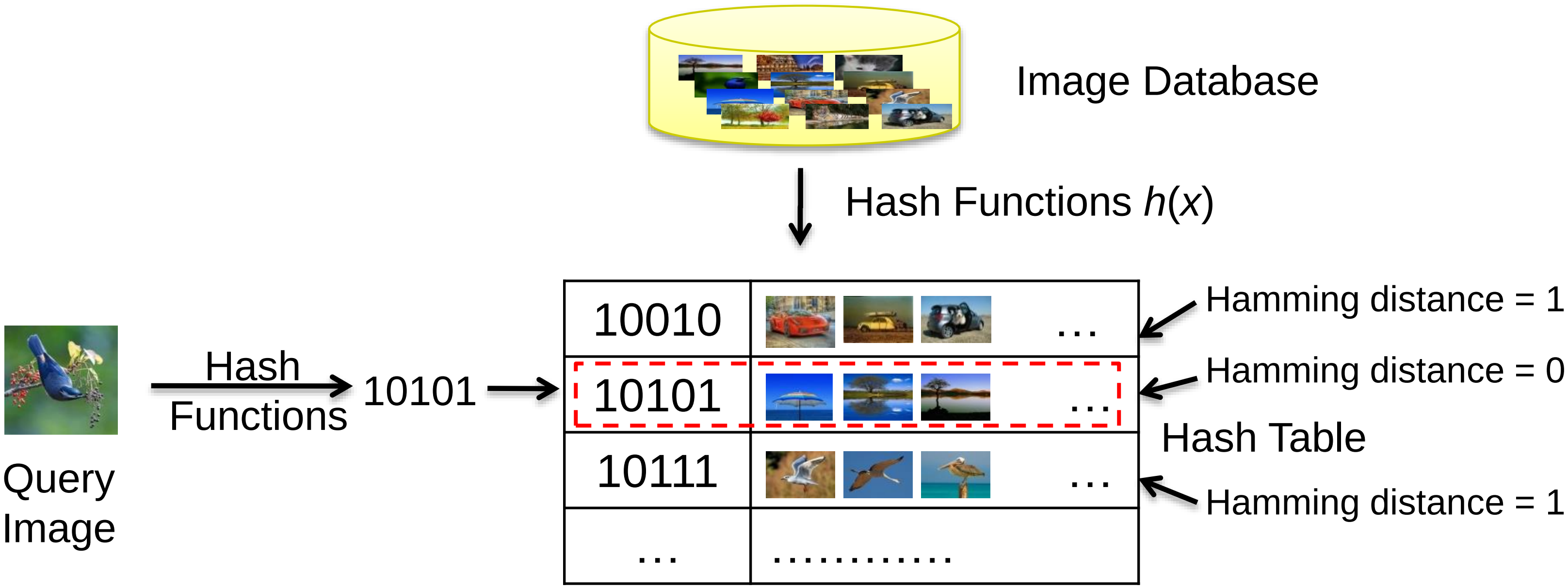


Image Database

Hash Functions $h(x)$

Query Image

Hash Functions

10101

10010	…
10101	…
10111	…
…	…………

Hamming distance = 1

Hamming distance = 0

Hash Table

Hamming distance = 1

Hashing-based Indexing -4

- What is a good hash code?
  - Compact: requires a small number of bits to code the full dataset
  - Effective: maps similar samples to similar binary code words
  - Efficient: Easily computed for a new image
- Design hash functions for generating hash codes
  - Consider a simple and popular hashing approach
  - Locality Sensitive Hashing (LSH) – random projection

Locality Sensitive Hashing -1

- Hash high-dimensional data into hamming space based on random projection

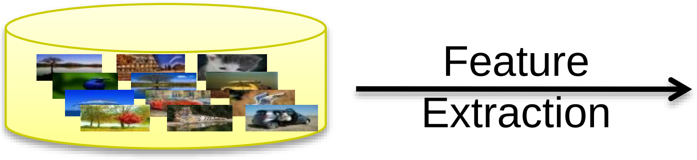


Image Database

Feature Extraction

$X_1=[x_{11}, x_{12}, \dots, x_{1d}]^T$

$X_2=[x_{21}, x_{22}, \dots, x_{2d}]^T$

$\vdots$

$X_N=[x_{N1}, x_{N2}, \dots, x_{Nd}]^T$

Recall that database images are represented by visual feature vectors.

- LSH focuses on approximate nearest neighbor search by hashing similar points together as much as possible using Random Projection

Locality Sensitive Hashing -2

- The basic idea of LSH is to project the data into a low-dimensional binary (Hamming) space; that is, each data point is mapped to a b-bit vector, called the *hash key*
- Each hash function ***h*** must satisfy the locality sensitive hashing property:

$$\Pr[h(x_i) = h(x_j)] = \mathrm{sim}(x_i, x_j)$$

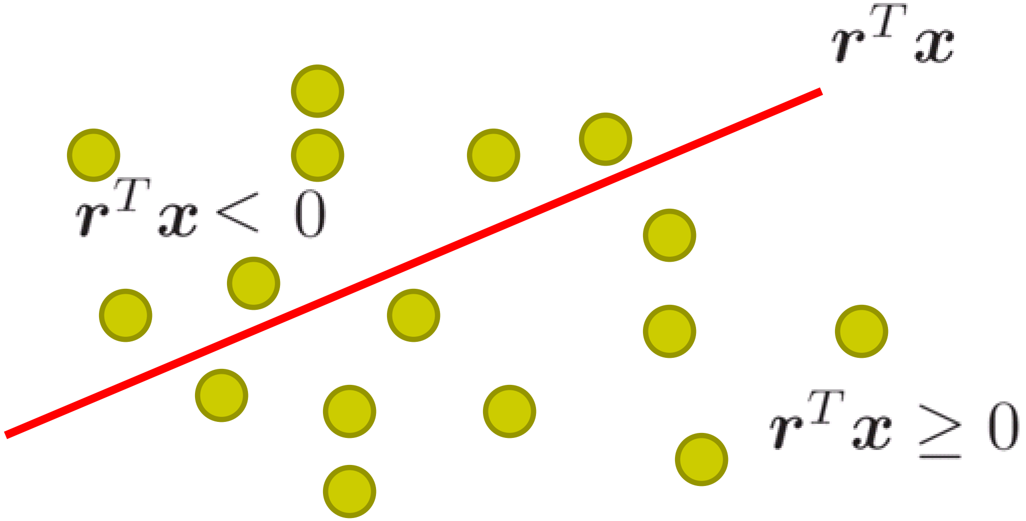
where $\mathrm{sim}(x_i, x_j) \in [0, 1]$ is the similarity function of interest

Locality Sensitive Hashing -3

The hashing function of LSH to produce Hash Code

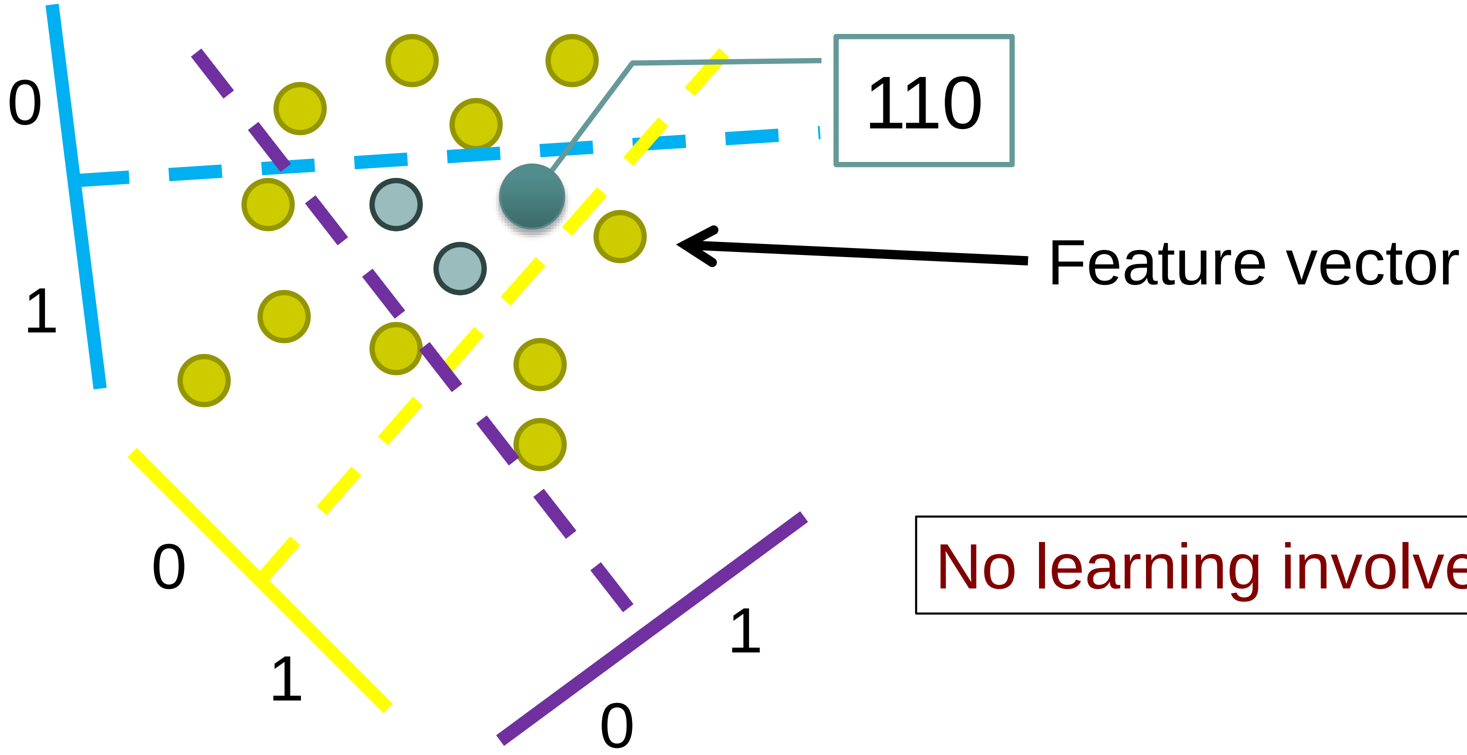
$$h_{\boldsymbol{r}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T\boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$\boldsymbol{r}^T\boldsymbol{x} \geq 0$ is a hyperplane separating the space


$\boldsymbol{r}^T\boldsymbol{x}$
$\boldsymbol{r}^T\boldsymbol{x} < 0$
$\boldsymbol{r}^T\boldsymbol{x} \geq 0$

Locality Sensitive Hashing -4

- Take random projections of data $r^T x$
- Quantize each projection with few bits



No learning involved

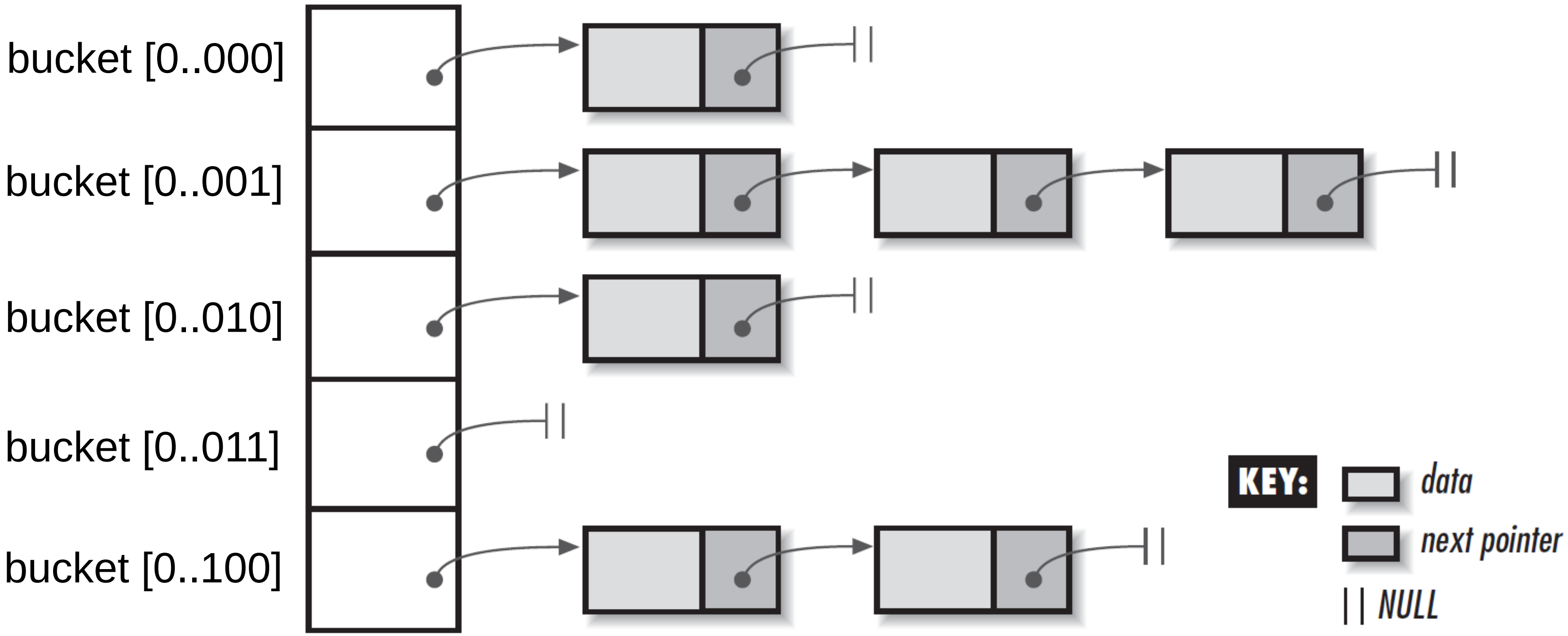
Locality Sensitive Hashing -5

- Indexing of samples

```
Input: A set of points $P$,    $M$ (number of hash tables)
Output: Hash Tables $T_i$, $i = (1, …, M)$
Foreach $i = 1, …, M$
    Initialize hash table $T_i$ by generating random hash functions $g_i$
Foreach $i = 1, …, M$
    Foreach $j=1,…,N_{samples}$
        stroe point $p_j$ on bucket $g_i(p_j)$ of hash table $T_i$
```

Locality Sensitive Hashing -6

- The resulting Hash Tables after indexing



Locality Sensitive Hashing -7

- Search: An approximate Nearest Neighbor Search with LSH

```
Input: A query point $q$, $K$ (number of approximate nearest neighbor)
Access To hash tables $T_i$, $i = (1, …, M)$
Output: $K$ (or less) approximate nearest neighbors
        $S \leftarrow \Phi$
Foreach $i = 1, …, M$
        if $g_i$ matches the $g_q$ of query sample q, then
        $S \leftarrow S$ U {points found in $g_i$ bucket of table $T_i$}
Return the $K$ nearest neighbors of $q$ found in set $S$
```

Locality Sensitive Hashing -8

- Limitations:
  - Need long codes to achieve an acceptable accuracy
  - Need many hash tables to get a good recall
  - But the sizes of L and M are heuristic

- May learn the appropriate hash codes based on machine learning techniques → Spectral Hashing (to read up yourself)

Contents

- Introduction to Image Indexing
- Tree-based Indexing
- Inverted File Indexing
- Hashing-based Indexing
- MM Indexing Strategy
- Google Search Architecture
- Summary

MM Indexing Strategy

- Multiple features may be indexed in different indexing structures
- What indexing structure should be used for which content?
  - KD-Tree: Traditionally used to index global dense features, like color histograms, texture histogram etc
  - Inverted Structure: Most obvious candidate is Bag-of-Visual-Words Feature
  - Hashing Index: For Global and any combined (fused) features
- A similarity measure that combine multiple features:

$$Sim(\underline{Q}, \underline{D}_i) = \sum_{j}^{m} \alpha_j Sim(\underline{Q}^j, \underline{D}_i^j), \quad \sum_{j} \alpha_j = 1; \quad j = 1,..m \ features$$

Current Image Search Engines

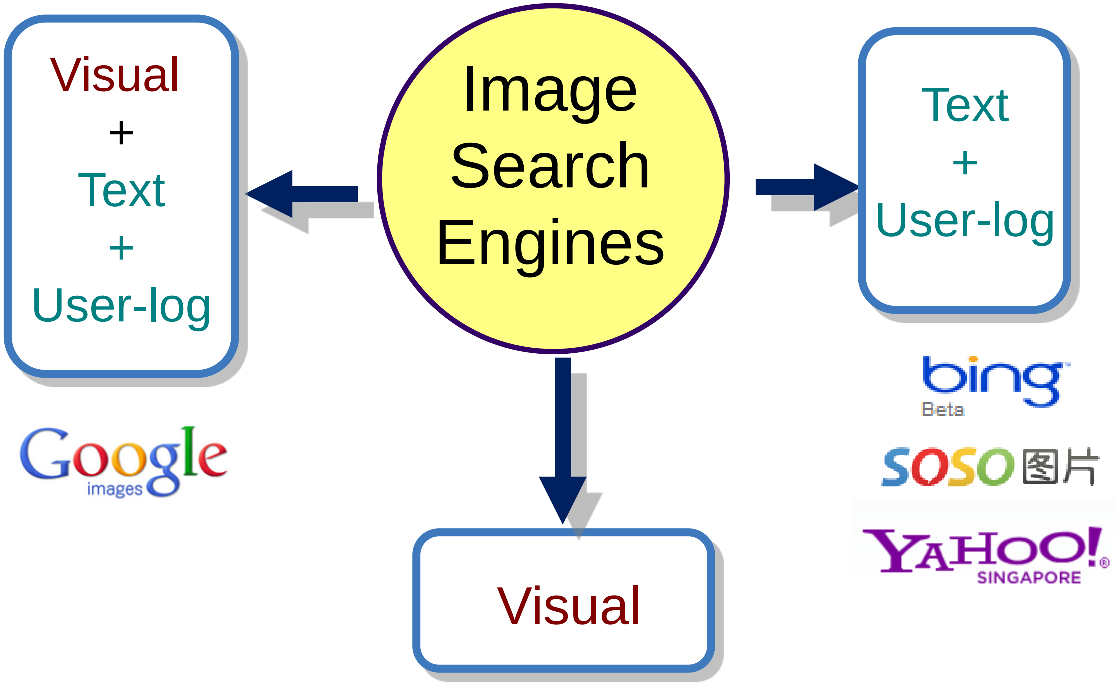


Image Search Engines

Visual + Text + User-log

Text + User-log

Visual

Google images

bing Beta

SOSO 图片

YAHOO! SINGAPORE

Sogou 搜狗 图片

Baidu 图片

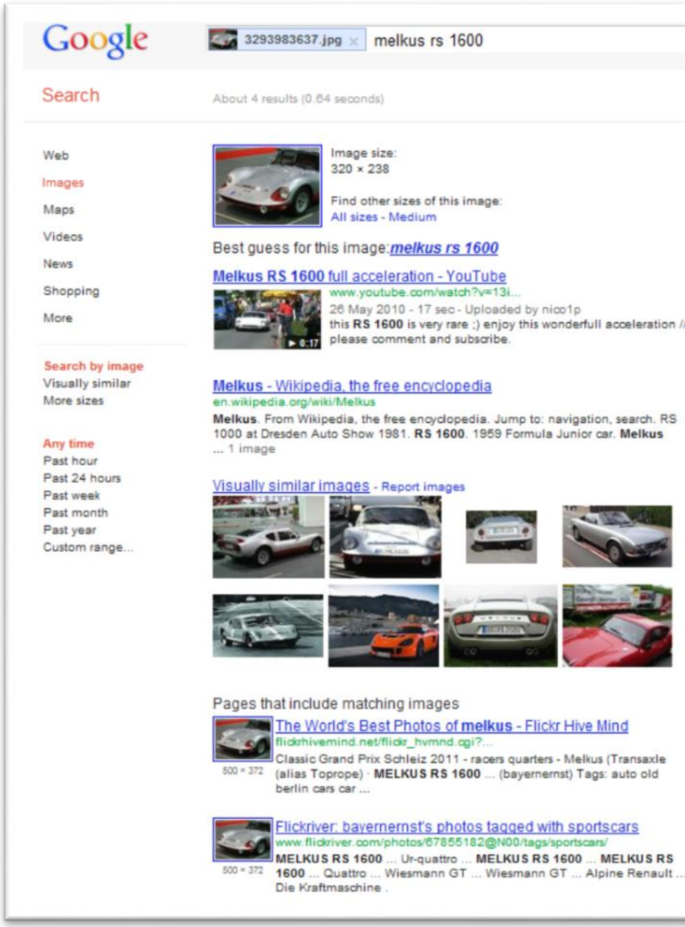
GazoPa β similar image search

imagine 图想 测试版
以图搜图，帮您找到相似宝贝！

Incogna

淘淘搜

TinEye Reverse Image Search



3293983637.jpg × melkus rs 1600

Search

About 4 results (0.64 seconds)

- Web
- Images
- Maps
- Videos
- News
- Shopping
- More

Search by image
- Visually similar
- More sizes

Any time
- Past hour
- Past 24 hours
- Past week
- Past month
- Past year
- Custom range...

Image size:
320 × 238

Find other sizes of this image:
All sizes - Medium

Best guess for this image:*melkus rs 1600*

Melkus RS 1600 full acceleration - YouTube
www.youtube.com/watch?v=13i...
26 May 2010 - 17 sec - Uploaded by nico1p
this **RS 1600** is very rare ;) enjoy this wonderfull acceleration /
please comment and subscribe.

Melkus - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Melkus
Melkus. From Wikipedia, the free encyclopedia. Jump to: navigation, search. RS 1000 at Dresden Auto Show 1981. **RS 1600**. 1959 Formula Junior car. **Melkus** ... 1 image

Visually similar images - Report images

Pages that include matching images

The World's Best Photos of **melkus** - Flickr Hive Mind
flickrhivemind.net/flickr_hvmnd.cgi?...
Classic Grand Prix Schleiz 2011 - racers quarters - Melkus (Transaxle (alias Toprope) · **MELKUS RS 1600** ... (bayernernst) Tags: auto old berlin cars car ...

500 × 372

Flickriver: bayernernst's photos tagged with sportscars
www.flickriver.com/photos/67855182@N00/tags/sportscars/
MELKUS RS 1600 ... Ur-quattro ... **MELKUS RS 1600** ... **MELKUS RS 1600** ... Quattro ... Wiesmann GT ... Wiesmann GT ... Alpine Renault ... Die Kraftmaschine .

500 × 372

Large Scale Image Indexing

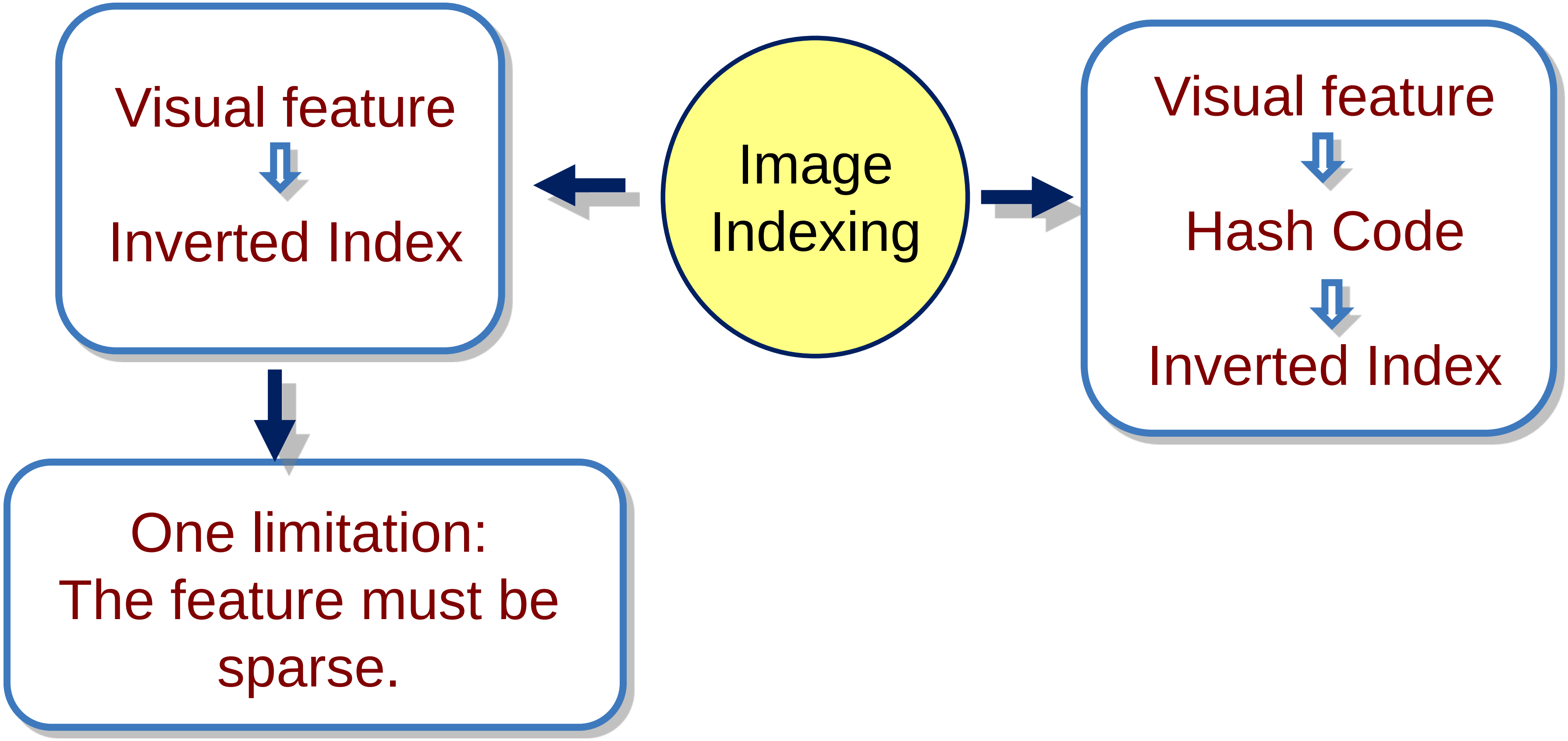


Image Indexing

Visual feature
⇩
Inverted Index

One limitation:
The feature must be sparse.

Visual feature
⇩
Hash Code
⇩
Inverted Index

Framework of a Large-Scale Media Search Engine

Input Image

Feature Extraction

Hash Code Generation

Hash Code Extension

Index by Lucene

Re-ranking

SIFT Feature Extraction

Sparse coding formulation in the

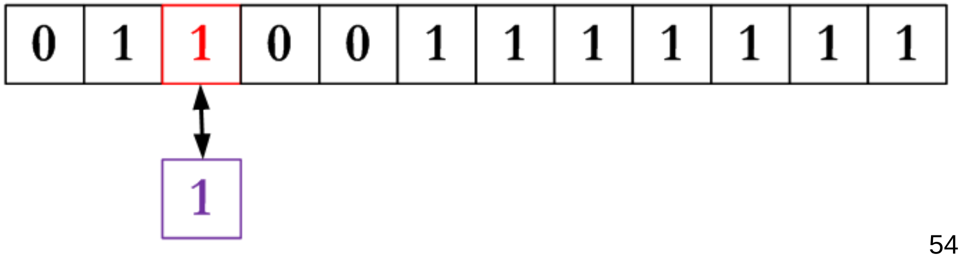
$$\min_{\mathbf{U},\mathbf{V}} \sum^{M} \|\mathbf{x}_m - \mathbf{u}_m\mathbf{V}\|^2 + \lambda|\mathbf{u}_m|$$

Hash Code Generation

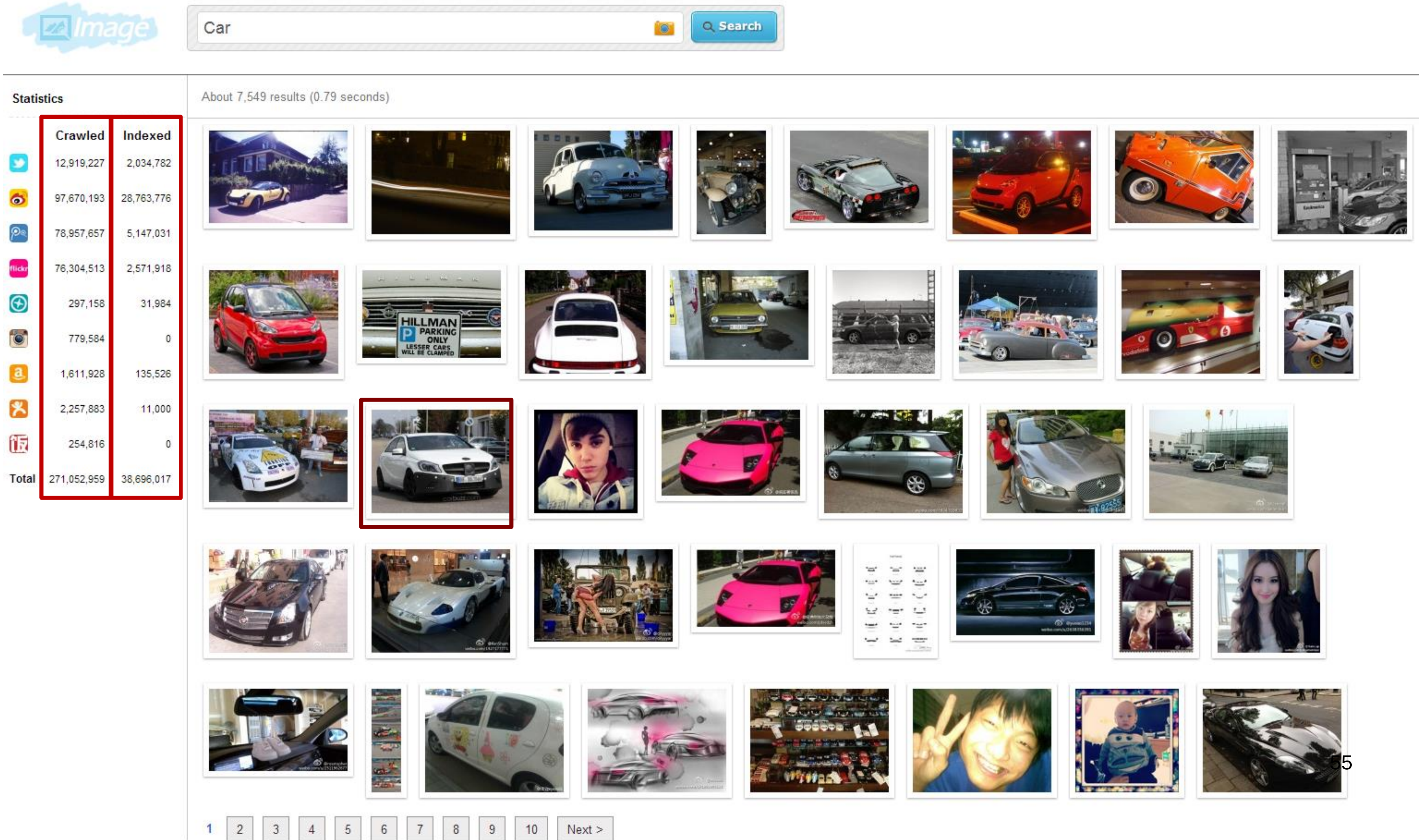
1. Fit Multidimensional Rectangle

Hash Code Extension

For the given image, the words are generated by the hash code and the code with a Hamming distance of 1 or 2. The number of words for each image is $1 + C_{32}^{1} + C_{32}^{2}$.



Query Example: Text query ('Car') (www.nextcenter.org/)

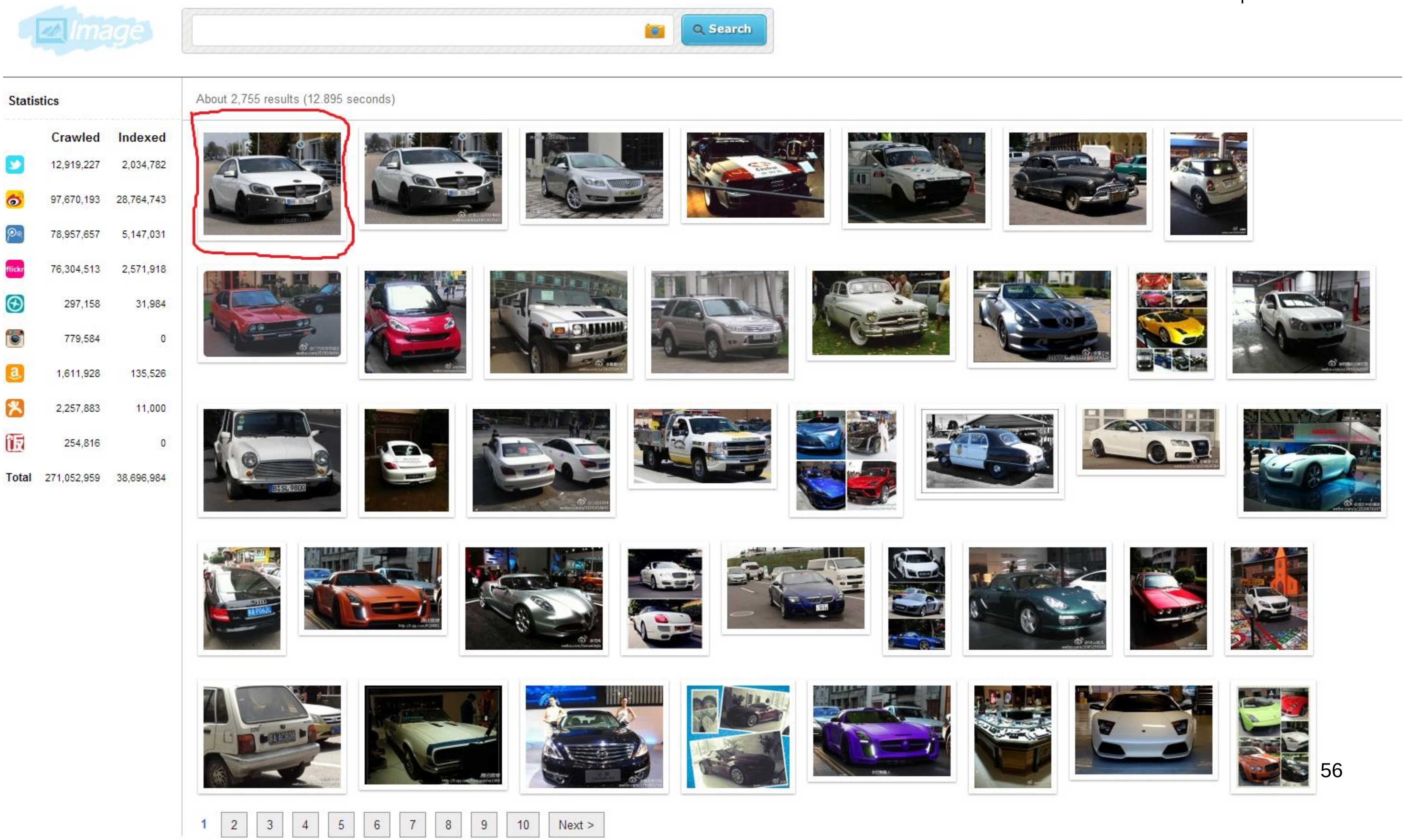


Statistics

	Crawled	Indexed
	12,919,227	2,034,782
	97,670,193	28,763,776
	78,957,657	5,147,031
	76,304,513	2,571,918
	297,158	31,984
	779,584	0
	1,611,928	135,526
	2,257,883	11,000
	254,816	0
Total	271,052,959	38,696,017

About 7,549 results (0.79 seconds)

Query Example: Image query



Statistics

	Crawled	Indexed
	12,919,227	2,034,782
	97,670,193	28,764,743
	78,957,657	5,147,031
flickr	76,304,513	2,571,918
	297,158	31,984
	779,584	0
a	1,611,928	135,526
	2,257,883	11,000
饭	254,816	0
Total	271,052,959	38,696,984

About 2,755 results (12.895 seconds)

Contents

- Introduction to Image Indexing
- Tree-based Indexing
- Inverted File Indexing
- Hashing-based Indexing
- MM Indexing Strategy
- Google Search Architecture
- Summary

Design & Architecture of Google

- Main indexing system: An inverted file and direct file system
- Additional indexing features:
  1. Link information – in the form of PageRank
  2. Anchor Text - additional description of intra-content
  3. Location information - better in handling pseudo phrases
  4. Formatting info
- Features 1 and 2 are anti-Spamming device
- Features 1, 3, 4 are precision device

Link Feature: Page Rank

- One of the key features that contributes to the success of Google search is the Page Rank
- Approach:
  - It generates PageRank based on the entire web graph, rather than just a subset (read up)
  - PageRanks are "pre-computed", and hence provides a static, a priori "importance" estimates for all the pages on the web
  - At query time, these scores are used conjunction with query-specific IR scores to rank query results
  - A possible ranking score is:

$$W(q, \underline{D}_i) = \alpha * w_i(q, \underline{D}_i) + (1-\alpha) * \text{Page-Rank}\ (\underline{D}_i)$$

More about this ranking score later

Anchor Text, Location & Formatting Features

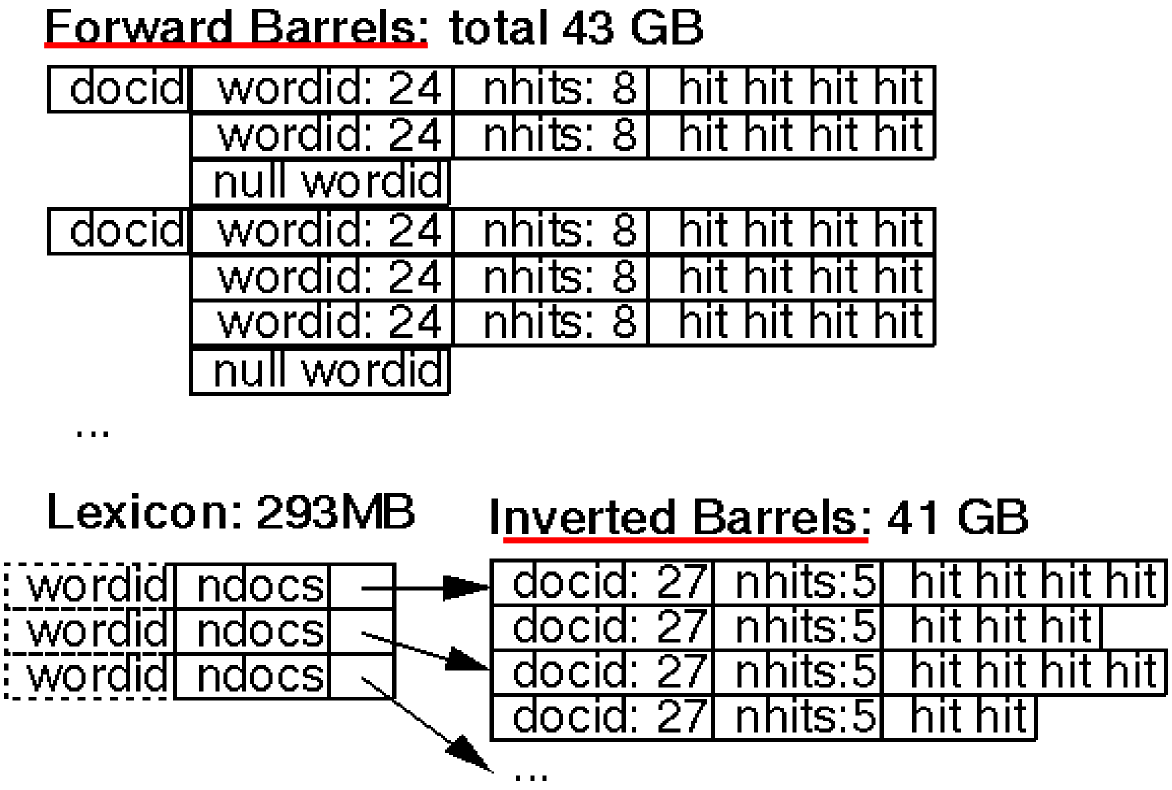
- Google associates anchor text with the page that the link points to:
  - Often provide more accurate description (what other people thinks of this page) of web pages than the pages themselves
  - Makes it harder for pages to appear to be something it is not
  - Can be used to index pages that are hard to describe by text
- Location information: makes extensive use of proximity in search
- Formatting features: words in larger or bolder fonts are weighted higher than others .

Indexing Structures -1

Both forward and Inverted Indices

- Indexer: convert each doc into word occurrences called hits, and stores them in:
  - Forward-index “barrels”
  - 2 Inverted-index “barrels”: 1) titles & anchors; 2) all the rest.

    (search Index 1 first, if not enough hits, then search Index 2)



Forward Barrels: total 43 GB

docid | wordid: 24 | nhits: 8 | hit hit hit hit
wordid: 24 | nhits: 8 | hit hit hit hit
null wordid
docid | wordid: 24 | nhits: 8 | hit hit hit hit
wordid: 24 | nhits: 8 | hit hit hit hit
wordid: 24 | nhits: 8 | hit hit hit hit
null wordid
...

Lexicon: 293MB

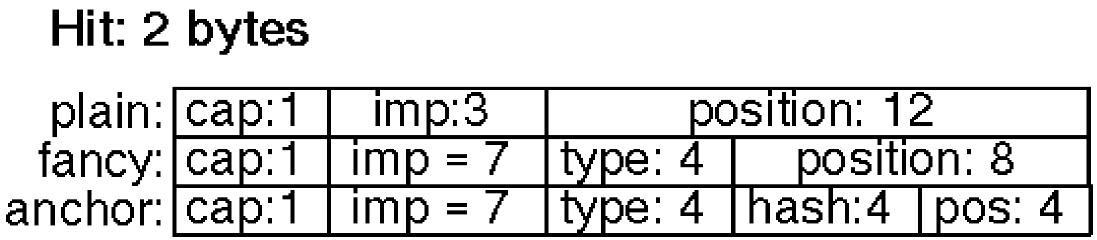
wordid | ndocs
wordid | ndocs
wordid | ndocs

Inverted Barrels: 41 GB

docid: 27 | nhits:5 | hit hit hit hit
docid: 27 | nhits:5 | hit hit hit
docid: 27 | nhits:5 | hit hit hit hit
docid: 27 | nhits:5 | hit hit
...

Indexing Structures -2

Hit Lists

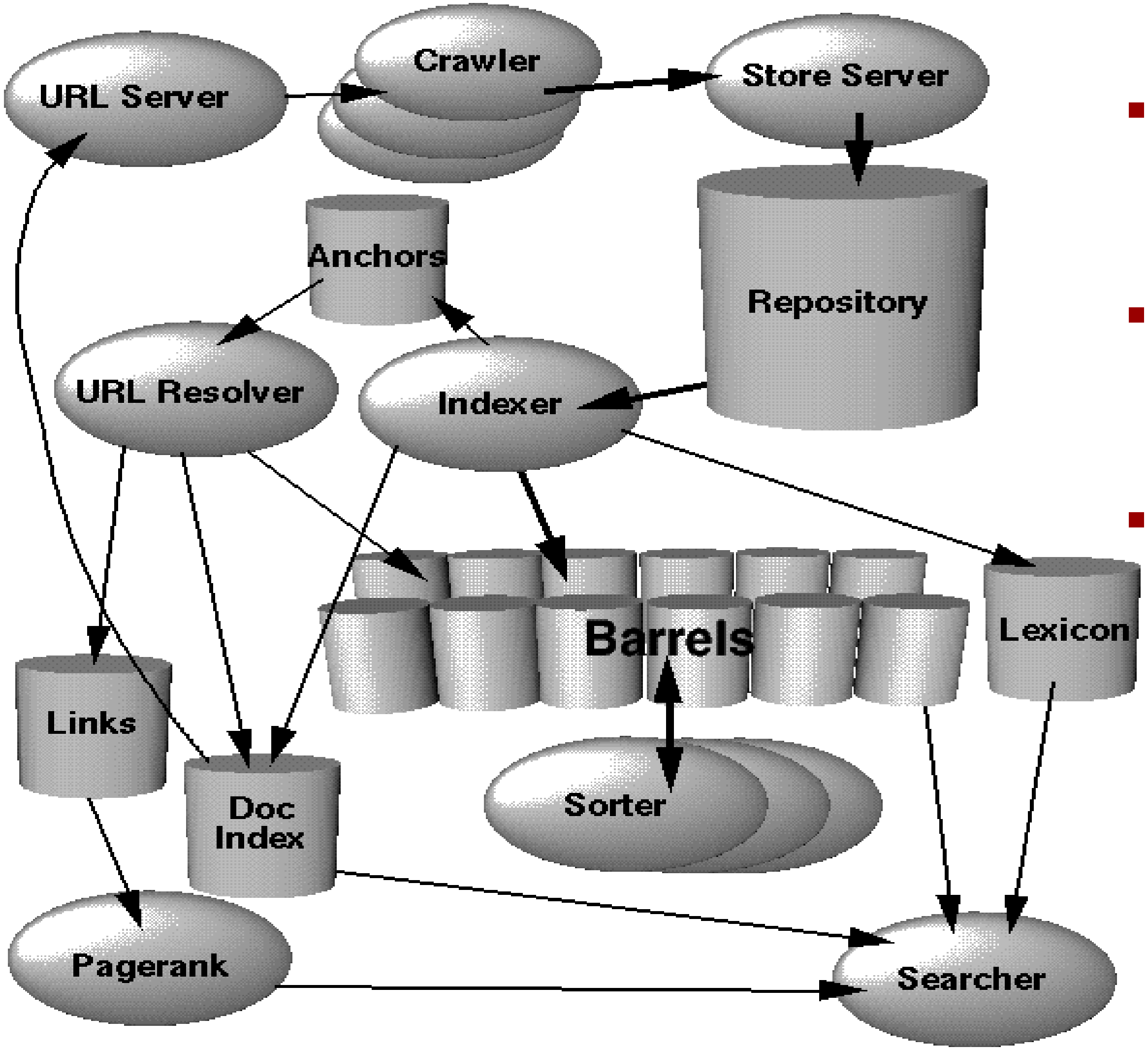


Hit: 2 bytes

plain:	cap:1	imp:3	position: 12		
fancy:	cap:1	imp = 7	type: 4	position: 8	
anchor:	cap:1	imp = 7	type: 4	hash:4	pos: 4

- Word Hit Lists:
  - Encodes list of occurrences of a particular word in a doc including (position, font, cap …)
  - Three lists of hits:
    - Fancy Hit: hits in URL, title, anchor text, meta tag ..
      (imp: font size, relative to normal, type: type of fancy text)
    - Plain Hit: everything else
    - Anchor Hit: store separately for efficiency reason

Overall Architecture



- Web pages are fetched by several distributed crawlers
- Storage Servers compress & store pages in repository
- Indexing..

Ranking of Results

- Google essentially performs keyword-based searches:
- Ranking for single word query
  - Examines hit list of several different types: (title, anchor, URL, plain large font, plain small font …)
  - Generate the weight vector ($\underline{V}_W$) for each pre-defined hit type ($\underline{V}_T$)
  - $\underline{V}_T = (t_{fancy}, t_{plain}, t_{anchor})$ -- pre-defined
  - $\underline{V}_W = (w_{facncy}, w_{plain}, w_{anchor})$
  - $w_i(q, \underline{D}_i) = \underline{V}_W \cdot \underline{V}_T$, for doc_i
  - Final score $W(q, \underline{D}_i) = \alpha * w_i(q, \underline{D}_i) + (1-\alpha) * \text{Page-Rank}(\underline{D}_i)$
- Ranking for multi word query:
  - Consider each word in turn
  - Need to consider proximity of hits when combining the contribute of each word – classify proximity into 10 different value bins ranging from phrase match to "not even close”
  - HOW TO DO IT??

Query Evaluation

- Best effort search:
  - Put a limit on response time, once a certain # of matched documents (currently set at 40,000) are found, the results will be presented to users
  - It is possible that sub-optimal results will be obtained
- Search Performance
  - Able to return results not covered by other search engines – because of PageRank & Proximity
- Query processing timing is dominated by disk IO over NFS

Contents

- Introduction to Image Indexing
- Tree-based Indexing
- Inverted File Indexing
- Hashing-based Indexing
- MM Indexing Strategy
- Google Search Architecture
- Summary

Discussions and Summary -1

- We have discussed various methods for indexing media content.
  - KD-Tree and LSH hashing for indexing “raw” feature vectors: KD-Tree works well for low-dimensional data (e.g., less than 10-D). LSH is effective for high-dimensional data, like visual features.
  - Inverted file for indexing visual words, making it possible to apply text search methodology for media search.

Discussions and Summary -2

- While visual words show good invariant to geometric and photometric changes and are effective for object retrieval, global features (e.g., color, texture) are still useful and effective for retrieving scene images.

- To support various types of user queries, it is necessary to index images based on different kinds of features using corresponding index techniques, like hashing and inverted file.

Next and Future Lessons

- From next lecture onwards, we will look into fundamentals of multimedia
- Next Lesson: Fundamentals in Digital Multimedia
- Following Lessons will look into MM compression, Audio, JPEG and MPEG etc..