Efficient Algorithms for Image Retrieval

Tutorial Image Retrieval

Thomas Deselaers, Henning Müller

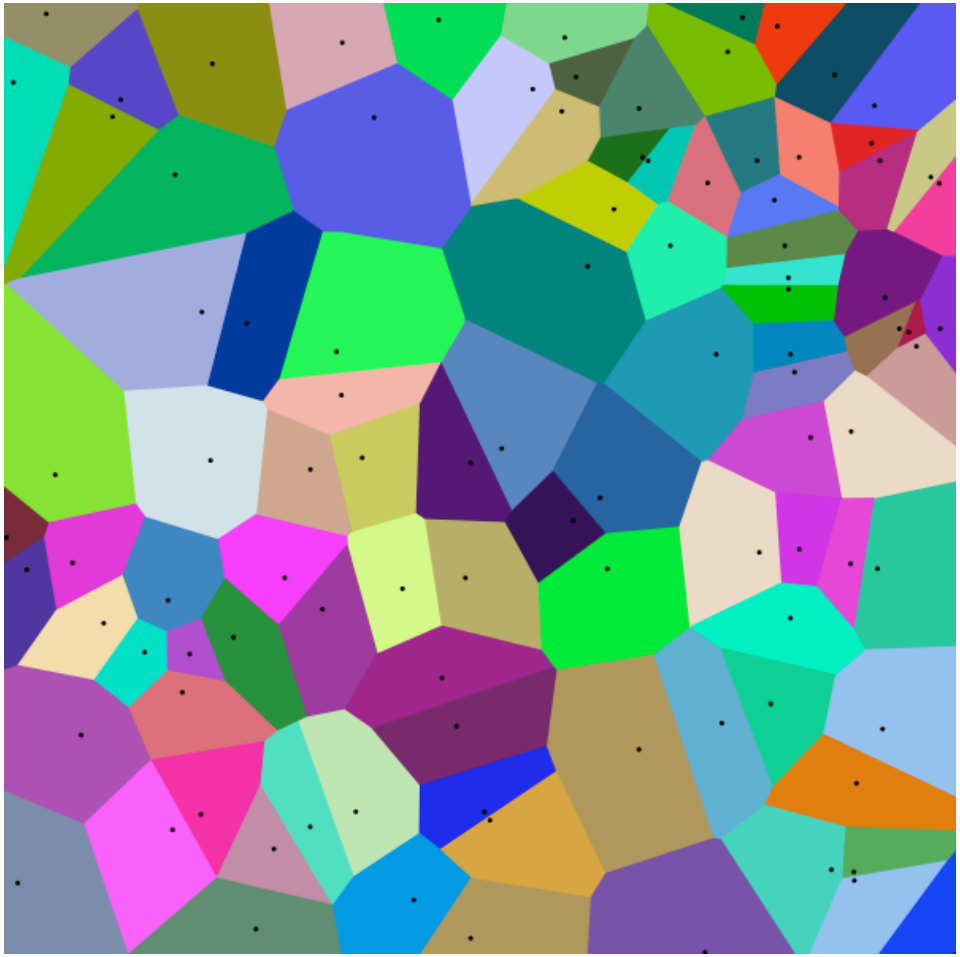
Outline

- Algorithms for the continuous approach
  - Fast nearest neighbour searching
- Algorithms for the discrete approach
  - Inverted document files

Nearest Neighbour

- Voronoi Diagram in a 2D space
- A set of points and the corresponding Voronoi cells
- A Voronoi cell is the area where a point's nearest neighbour is the seed of the cell

Source: Wikipedia

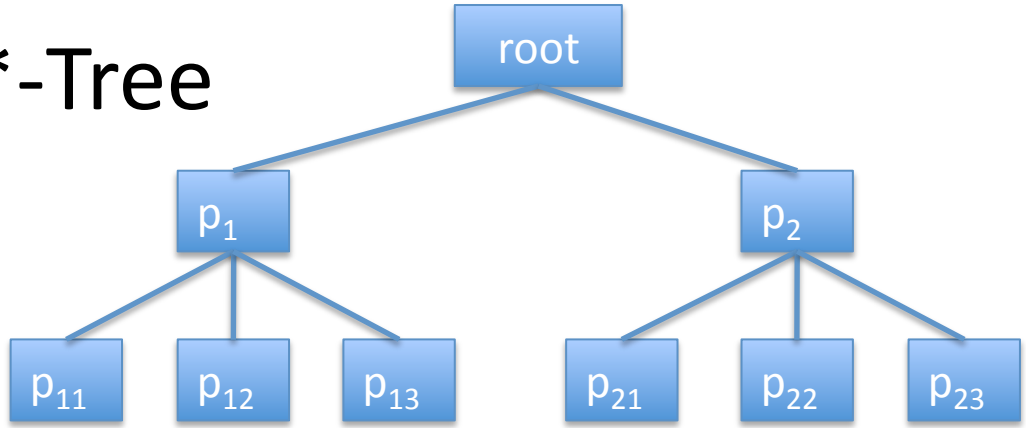


Nearest Neighbour

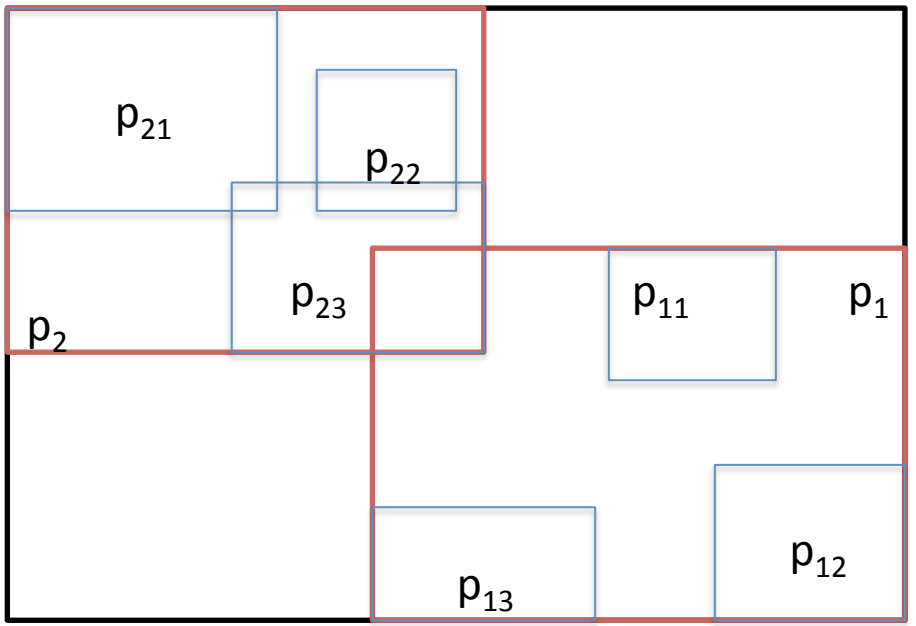
- Standard nearest neighbour search

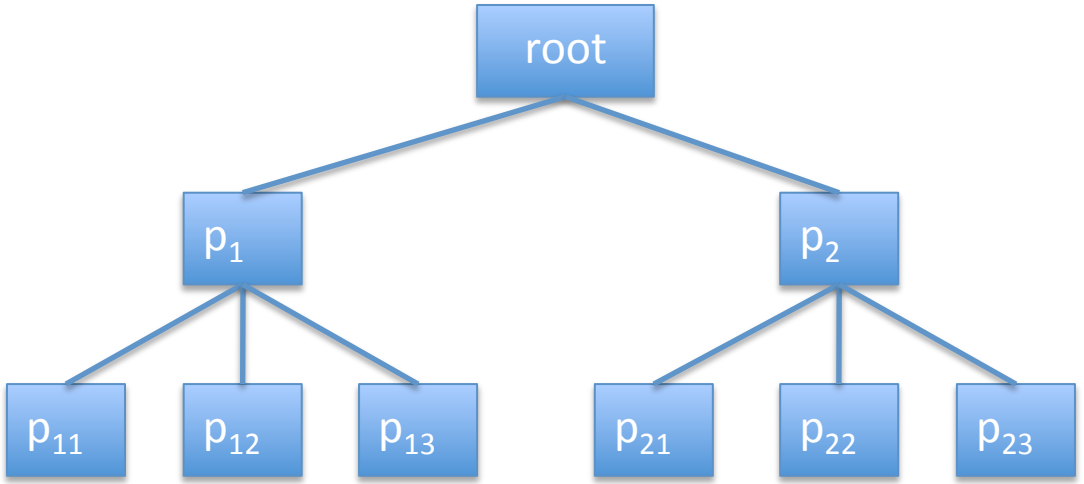
```
X[1] … X[N]: data set
Q: query
mindist=∞
bestN=-1

For n=1:N:
   d=dist(q,X[n])
   if d<mindist:
      mindist=d
      bestN=n
return bestN
```

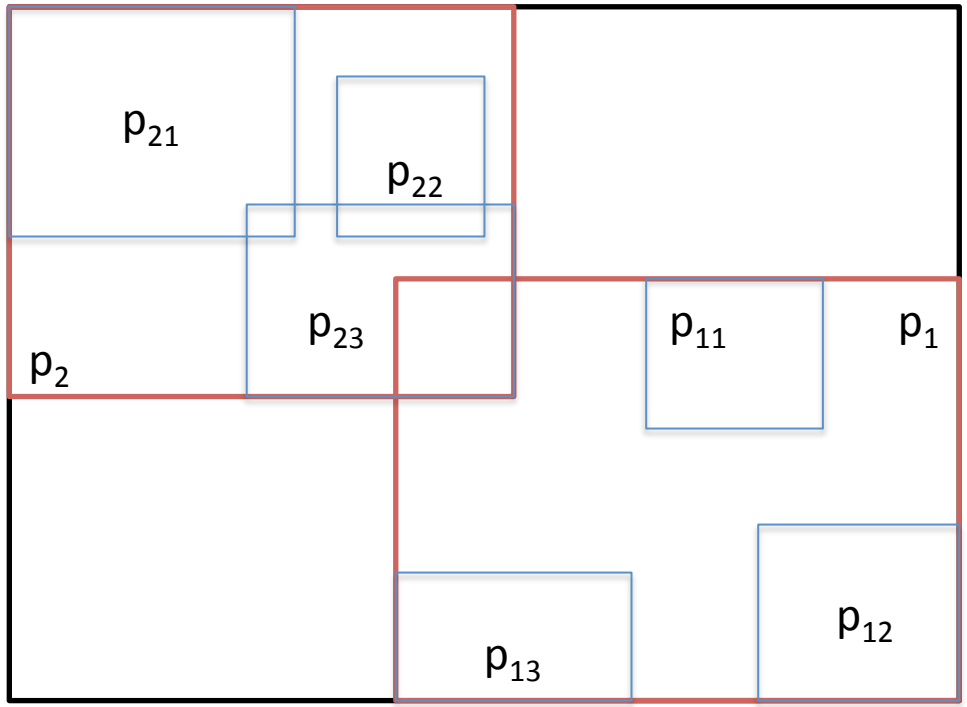
R*-Tree



- Data structure for multi-dimensional data
- Save data in a tree structure
- Directory nodes (blue)
- Data nodes (red)
- Save minimum bounding rectangles in directory nodes

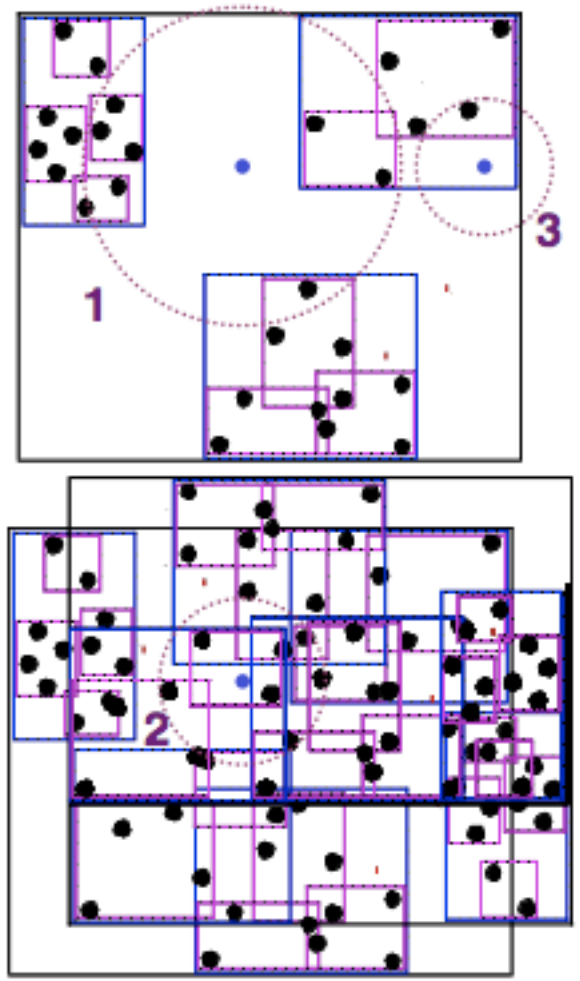



root
p_1
p_2
p_{11}
p_{12}
p_{13}
p_{21}
p_{22}
p_{23}


p_{21}
p_{22}
p_{23}
p_2
p_{11}
p_1
p_{13}
p_{12}

Index Structures: Complexity

- Linear search
  - Complexity O(n/C), very small overhead
- Bad situation for index structures
  - Large range
  - Strongly overlapping regions
  - Few regions are not accessed
  - Common with high dimensionalities
  - Complexity O(n/C), high overhead
- Good situation
  - Small range
  - Small overlaps
  - Many regions do not have to be read
  - Complexity $O(\log_C(n))$

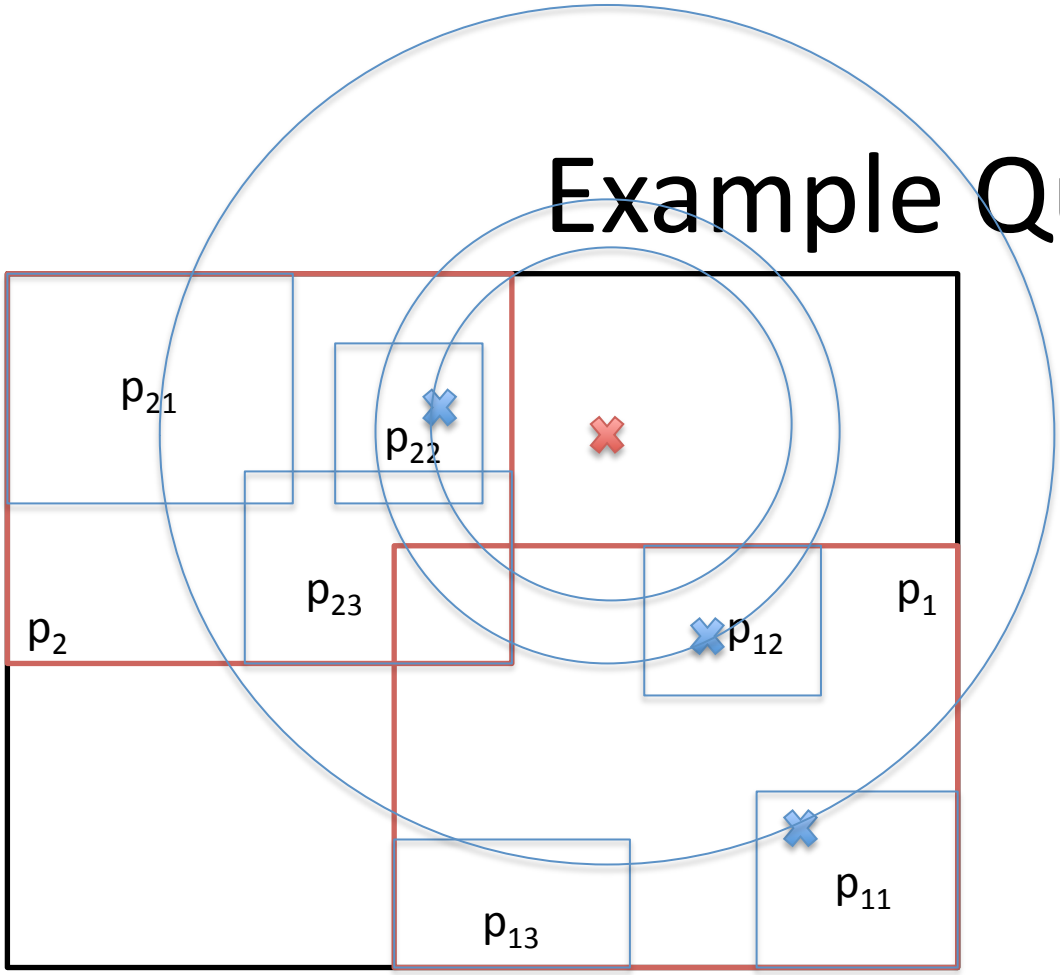


Nearest Neighbour Search with Index

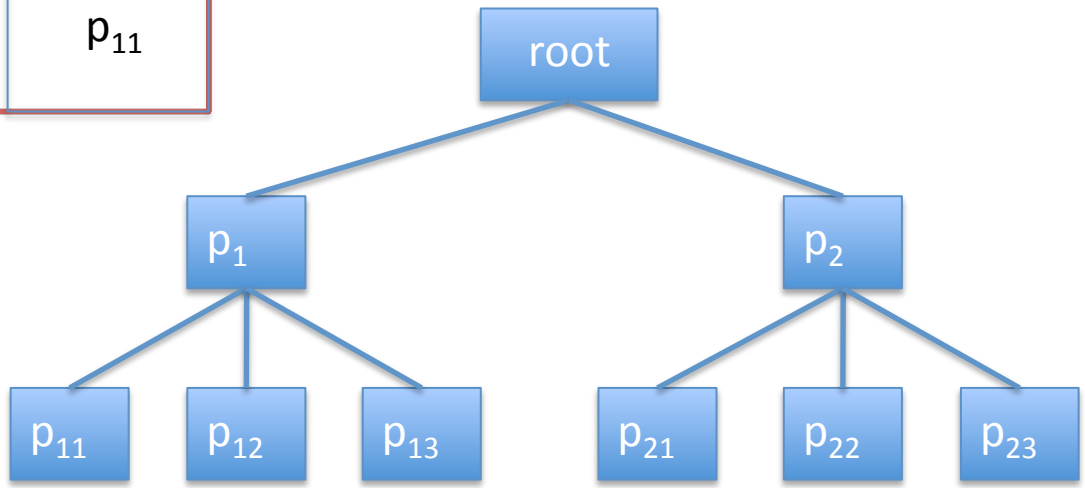
```
Init: resultdist=∞
Function SimpleNNQuery(Point q, Address: page):
    page.load()
    if isDataPage(page):
       for x in page.points:
           d=dist(q,x)
           if d<resultdist:
               resultdist=d
               result=x
    else:
       for p in page.childPages:
           if MINDIST(q,p)<resultdist:
               SimpleNNQuery(q,p);
```

- First path finds arbitrary point
- Search space only slowly reduced
- Many pages unnecessarily read

Example Query



- If the search had started with p_2 no page from p_1 would have been read
- Clearly non-optimal



Nearest Neighbour Search with Index

Best-First Search

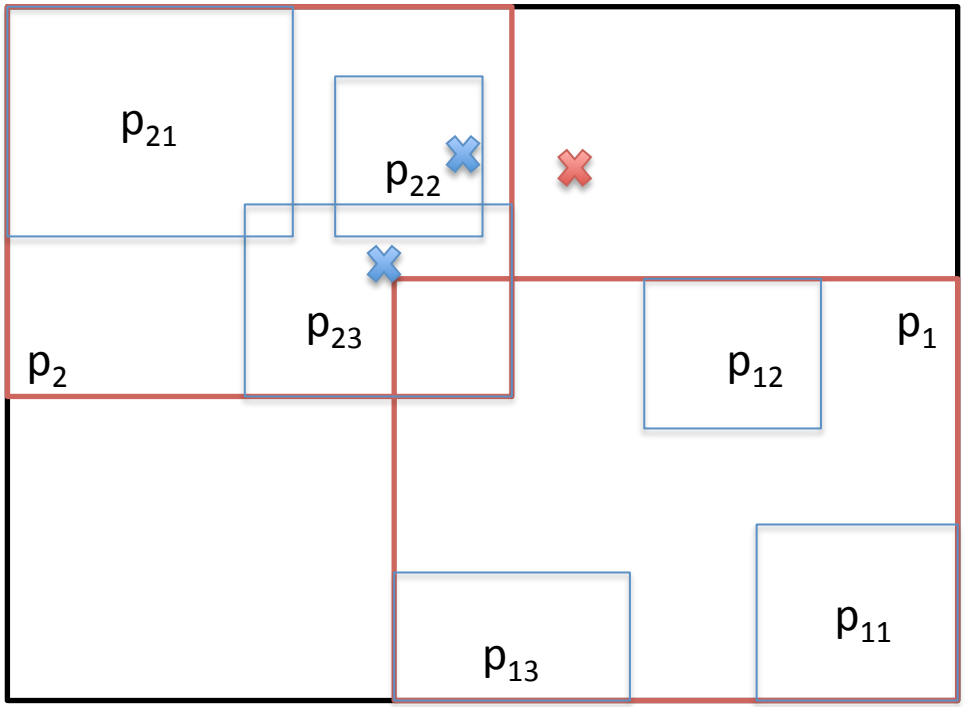
- Avoid recursion
- Instead use priority queue APL (active page list)
  - List which contains directory pages to be processed sorted by priority

- Definition: a page p is active, if and only if
  - p not yet processed
  - Parent of p processed
  - Minimum distance between p and q < current best distance
- Initialisation: APL = [root]
- In each step, process best page from APL
  - Data pages: as before
  - Directory pages: check minimum distance to query

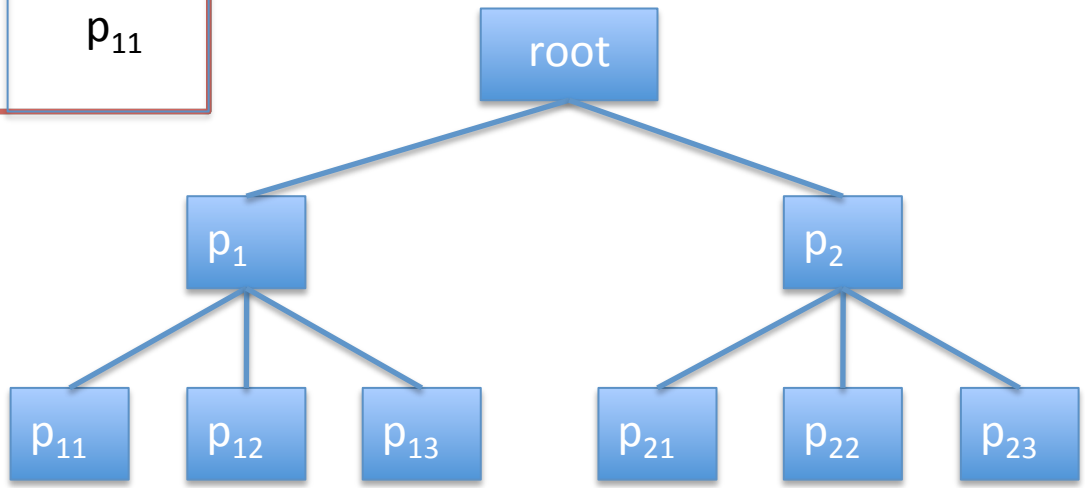
Best First Nearest Neighbour Search with Index

```
Init: apl = [(0.0,root)] // sorted by dist
      resultdist=∞
While apl.notEmpty() and apl[0].dist<resultdist:
    page=apl[0].load()
    delete(apl[0])
    if isDataPage(page):
       for x in page.points:
           d=dist(q,x)
           if d<resultdist:
               resultdist=d
               result=x
    else:
       for p in page.childPages:
           h=MINDIST(q,p)
           if h<resultdist:
               apl.insert((h,p))
```

Example Query with Best First Search



APL: [... 8.2, p_{21})]
resultdist: 7.0



Best First Search is optimal

Here: a draft of the proof

1. Completeness: It will find the correct NN of a query
   - Every correct algorithm has to access all pages that intersect with the NN sphere of q
   - These pages have MINDIST < resultdist
2. It accesses pages in ascending order from the query
   - The APL is sorted by MINDIST and the algorithm will terminate once it is impossible to find any point closer to q than the current result
3. It will not access a single page with MINDIST larger than the true NN distance
   - Child pages cannot have a MINDIST smaller than its parents

Discrete Approach

- Inspired by textual information retrieval
- Each image is represented by a set of binary features (features may be present (possibly multiple times) or absent)
- Feature is either present or absent
  - Similar to words being absent or present in a document
- Images containing the same (informative) features are assumed to be relevant to a query
- Example: GIFT – GNU Image Finding Tool

Discrete Approach

- GIFT uses TF-IDF (text frequency/inverse document frequency) ranking
  - Reduce the impact of features which occur frequently in the data (comparable to “the” in texts)
- TF: frequency a feature *i* has in a document d_j

$$tf(i, d_j) = \frac{n(i, d_j)}{\sum_k n(k, d_j)}$$

- IDF: measures importance of a term

$$idf(i) = \log\left(\frac{|\mathcal{B}|}{|\{d_j : n(i, j) > 0\}|}\right)$$

Discrete Approach

- *tf* captures how often a feature occurs in a document
  - Features that occur often in a document describe this document well
- *idf* captures how relevant a feature is
  - Features that occur rarely in the full database are important
- Important are those features which are often in one image, but seldom over the full data set
  - Images which share seldom features are relevant with respect to each other

Discrete Approach in GIFT

- In GIFT, 4 different feature sets are considered
  - Global colour
  - Local colour
  - Global texture
  - Local texture
- For all local features a tf-idf-like score is calculated and these are fused as a weighted sum
  - Global features are compared with a histogram intersection
- For many cases, the discrete and the continuous approach can be simulated in the respective other
- In GIFT
  - Images have about 1,500 to 2,000 features
  - Similar images share about 400-500

Discrete Approach: Inverted Files

- Store a mapping from content to location
  - E.g. for each feature a list of images that contain this feature
- Allow for efficient searches even for huge amounts of images
  - *idf* for each feature can be pre-calculated
  - *tf* for each feature is stored in the files
  - Allows for searching without accessing the images
- In the continuous approach searching for neighbours is linear to the amount of images, here it is at most linear to the number of features in an image
  - In practice, the features with high impact are processed first and the other features have less influence (Zipf's Law). Therefore, in practice, this approach is very fast

The inverted file

Inverted Files

- Access feature by feature instead of image by image
- Extremely fast access for rare features
- Efficient for sparsely populated spaces