Information Retrieval

# Technical

* Indexing:
  + analyse documents to extract features
  + Organise features into index
* Searching: compare feature with query and evaluate
  + Sequential O(N), Binary search O(Log N), Hash O(1) <~ cannot use
  + Use index (can answer query without access)
* User interaction: formulate query, display result

# Text bases search

* Index elements: keywords (full-text, manually)
  + Stemming ( form of word), Stopwords, Sematic process (synonym)
* Query: keywords + operator
  + Complex query: several keyword (all,many), operator (AND, OR,NOT)
* Inverted index: matrix of document and term ( each term contains list of docs)
  + Sparse matrix
  + Issues when build
    - Minimize database access
    - Build dynamic lexicon (replace word with number and update when new term found
      * Linear table: inefficient,
      * Sorted table with binary search: efficient search not update
      * Hash: efficient search and update, large memory requirement
    - Use in-memory or disk-based temporary storage
    - Minimize total creation time

## Build inverted index

* Read docs by docs, words by words add to list => memory size problem
* Algorithm 1: one pass and linklist
  + Array first\_cell store location of word i^th.
  + Array [doc,next\_cell] store document and location of next item on link list
* Algorithm 2: 2 pass inversion
  + First path find dictionary
  + Second pass: locate exact list size for each word and fill in

## Zipf’s law

* r.f = k (constant) r: rank, f: frequency. Probability of word of rank k is and N is total number of word occurance
  + - I\_1 = D/2. ½ Dictionary term occur 1 time

# Vector space model

* Dictionary have T term. Each term is a dimension. Define weight w of term t in document d
* TF-IDF. Similarity
* Pros: simple, mathematically, weight of term importance, ranked result
* Cons: misses syntactic, sematic, term independence, lack of user control

# Evaluation

* Precision, Recall, Average precision, MAP, ROC curve
* F –measure = 2PR / (P+R)
* Estimate R ( hard to have true relevance documents)
  + Run several engine same query -> different list
  + Find relevance -> union list
  + Find recall

# Relevance Feedback

* Modify query based on relevance judgement
  + Explicit: ask user
  + Implicit: from user action
* Main idea: query expansion and change weight of current term in query
* Rocchio Algorithm

# Latent Semantic Indexing

* SVD (Singular Value Decomposition) on docs-term matrix – technique based on PCA
* X = T S D^T , sim(i,j)
* Effect of projection: remove noise, reserve important inf, reduce dim, works better in practice
* LSI can handle synonym (sematic)

Web searching

* Search program: Robot, spider, worm
  + Based on graph search
  + Use hyperlink to recursively find new docs

# Link analysis

* Web is directed graph:
  + Ch(p): children of the page p
  + Pa(pP): parents of page p
* Use link structure to detect important page => **page Rank: measure of page important**
  + Initial idea: Number of page with point to p R(p) = |pa(p)|
  + Idea 2: Weight by page important R(p) =
  + Idea 3: some pages have more links than other => normalize R(p) =
  + Idea 4: avoid huge number
* How to estimate because of recursive => use iterative process
  + Initialize R\_0(p) for all p
  + Compute unnormalized reestimate
  + Normalize
* Rank sink problem
  + Group of links have cycles cannot get out, absorb all rank in system
  + Random Surfer Model
    - Simulate random on the web
    - Probability (FOLLOW): select random in q with probability 1/|ch(q)|
    - Probability (JUMP): jump to page p with probability E(p)
    - Stationary probability of being in page p
      * ( same as Page Rank)
* Page rank estimation : R\_(n+1) = A \* R\_(n), A is graph adjacency maxtrix.

Document Image Analysis

|  |  |
| --- | --- |
| Step | Task |
| Acquisition | Scanning |
| Preprocessing | Banalization, filtering |
| Page segmentation | Orientation, blocks, lines, words |
| Character Segmentation | Template,model |
| Text Recognition | Linguistic knowledge |
| Docs Understanding | Logical structure |

# Preprocessing

## Binarisation

* Threshold: Separate pixel into 2 classes
  + Low intra-var, High inter variance => max ratio variance

## Smoothing (filtering, noise removal)

* KFill Filter: if core all-non-X change core to X if
  + N: nb of X pixel in neighbour
  + C: nb of connected X region
  + R: nb of corner X pixels
  + (c==1) and (n>3k-4) hoặc c==4 and n=3k-4

# Page Segmentation

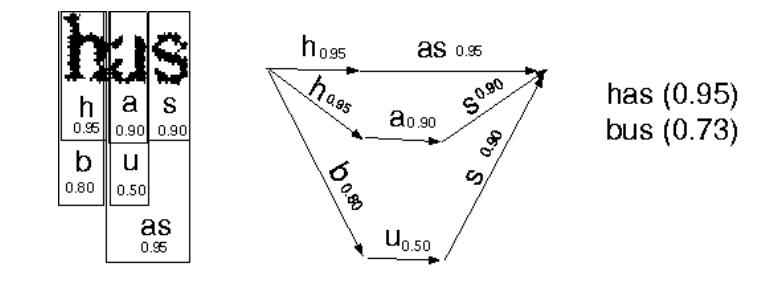
* Task: skew, text separation, lines word or characters
* Technique:
  + Projection: good orientation have **highest variance**
  + Hough Transform: detect lines
    - Point in lines will be in same point in hough transform
    - Each point in hough transform with \theta d(origin,line) parallel lines have some \theta but dif distance
  + Nearest neighbour:
    - P/s: RLSA (Run length smoothing algorithm): fill all sequence of white pixel with length l<= h
    - P/s: Connected component: assign component number to pixel based on neighbour. First pass: label and keep track merge components. Second pass: merge
    - connected component (char or word) -> centroid of connected component -> nearest neighbour component -> find vector from cluster (NN cluster) to cluster to define angle -> draw histogram

# Character recognition

* Normalization: size, italic, thickness, font, handwriting
* Recognition methods:
  + Template matching: feature, distance, sim
  + Statistical approach: model for vector feature distribution
  + Structural approach: description using synaptic rule
    - Describle pattern with primitive and relation
    - Ex: priminative (vertical stick, circle,…) and relation (“A” = WC (loop west, circle))
  + Model-based: Neural, HMM
* Feature: descriptive information
  + Pixel p(x,y), centre of gravity, hist, intersection with given line
* Chain-code stroke representation
  + Stroke: vector x(t)-x(t-1), y(t)-y(t-1)
  + Chain code: classify vector direction (8 possible class)
  + String distance, used matrix (characters str1 vs character str2).

# Post-processing

* Recoginition output: map of alternative with score, confusion matrix between character



* Contextual constraint: linguistic constraint, typographical constraint (numeric field)
  + Linguistic correction: with dictionary, without dictionary (probability), syntax
  + N-gram correction: find best spelling from picture by combine visual and linguistic score
    - Visual:
    - Linguistic :

# Page layout analysis

* Document structure = Logical structure (title,pararagraph,.. <~ position rule) + Physical structure ( blocks,lines, word <~ properties)

Image retrieval

Text annotation: old, expensive method

Content based method: recent

* Principle: feature vector, distance measurement

# Feature vector:

## Global feature

* Color **histogram:** quantize color space in N color
  + RGB: 2 bits per color => 4^3 (64) colors (bins need to represent it)
  + HSV: Colour – colour intensity – illumination
  + Pros: easy, fast, invariance (translation, rotation, scale, moving).
  + Cons: Too fine( large vector, sensitive), Too coarse(lack of discrimination), spial information

## Local feature

* ‘Grid based histogram:
  + Spatial inf, split into a grid and calculate hist
* Texture
  + Sobel edge detector
  + MPEG-7 edge: 5 types of edge (4 directional +1) + 16 regions in image => 80 bins
  + Gabor filter
* Region – segmentation
* Bag of visual word

## Bag of visual word

* The model:
  + Locate feature (location, size)
  + Extract description of visual content
  + Learn visual volcabulary
  + Quantize feature using volcabulary
  + Represent images by histogram of visual words

### Locate feature

* Salient (important) point: Harris corner detector (corner = points large variance in all directions)
* Dense sampling: use regular grids over image (those grid present important region)

### Extract visual content

* Descriptor: SIFT, SURF ( 4x4 grid of cells of 4x4 pixel) => 128 dimension ( 16 x 8 direction)

### Clustering feature vector (vocabulary)

* Clustering feature vector into K-clusters and quantize image ( many feature vector) into vector k-bins

### Similarity

* Manhattan, Euclidean
* Mahalantobis:
  + N samples X=(x\_1,x\_2,…x\_d)
  + Find average, covariance matrix …
* Combining multiple feature
  + Several feature vector and several distance measurement
* Feature distance normalization: (Gaussian form distance variable)
  + Compute distance values for many image pairs
  + Estimate mean value
  + Transform your distance
* Feature weighting from RF
  + Adapt weight for feature distance
  + (L nb of images smallét d(F(i),Q(i)), L\_i is same with feature i)

Image Textual annotation

Generate keyword from image content and use text-based search

* Train a classifier for each keyword
* Refinement: annotation by region (image segmentation and label region)
* Continuous relevance model: automatically segmentation and find probable keywordss
  + [keyword notation . visual similarity)
  + . Smoothing (in case 0 prob)

High Dimensional Indexing

Search algorithm like linear in BOW complexity O(Nd) d is dim of vector space, N is data base

# Space Partitioning: quad-tree

* Split n point in subcubes so that each cubes contains < threshold
* Search:
  + Find a smallest cube contain P
  + Explore the neighbour cube so that that interact with sphere (P,dmin), d set to small value
  + Update d and find more neighbour
* Curse of dimension
  + D is large enough, a linear scan perform better than partitioning method

# Locality Sensitive Hashing

* Hash function h(x) is locality-sensitive if any pair of point
* Hash query point search NN in the buckets contain the query point
  + Search NN among point by H value but can miss NN because hash function not good
* Use several hash function to reduce recall
  + Pr[h(x)=h(y)] = \alpha.
  + Probability I miss NN with h hash function (1-\alpha)^h
* Function: random projection quantization
  + if x close to y so pr[h(x)

Video Indexing

* Scene determination: combines shot into scene (higher level temporal fragments that correspond to the story-telling parts of the video.
* Shot segmentation continuous take from one camera (between shots is transition)

# Basic

## Hard cut detection

* Measure distance between 2 consecutive frame, cut when greater than threshold d(I\_i,I\_i+1) >= threshold

## Gradual Transition Detection (RMIT)

* Sliding window ( compare pre and post frame)
* Compute prepostratio .
* Peak ratio => end shot, minimal at beginning transition and rise maximum when the end of transition

## Challenge

* Similar environment
* Fast movement of large object
* Suddenly change illumination

# Camera motion

* Panning (left – right), tilting (up –down) , zooming
* Determine
  + Find movement in image
  + Model field with parametric model
  + Decide motion from parametric values
* Parametric model
  + Motion vector and
  + Pan , Title zoom
  + x + , zoom a2=a6>0 (in) other = 0
  + Estimate
    - At time t: point and motion vector
* Mosaic: camera motion can be used to comulate into single picture

# Keyframe selection

## Fix spacing

* Every 5 sec
* Simple, not adaption video content

## Difference based:

* Select a new keyframe when difference > threshold

## Shot based selection:

* Split video into shot
* Select zero, one, several frame (candidate: first, last or middle)
* Cluster centroid: images in shot -> feature -> frame closest to centroid

# Scene Segmentation

* Sequence of shots with similar topic

## Step1 : Time constrained clustering

* Idea: visually similarity shots, close in time => belong in same scene
* Shot S\_i is composed of f\_bi … f\_ei
* Time constrained vidual distance betwwen shot S\_i and S\_j
  + b,e (begin and end of each shots)

## Step 2: Transition graph

* Build graph ( clusters are nodes, edges between cluster which contain consecutive shot)
* Cut edge is an edge that can split into 2 disconnected graph
* Scenes are islands separated by cut edges