Singular Value Decomposition

- For an m x n matrix A of rank r there exists a factorization as follows:

$$\underset{m \times n}{A} = \underset{m \times m}{U} \underset{m \times n}{\Sigma} \underset{n \times n}{V^t}$$

- The columns of U are orthonormal eigenvectors of AA^t
- The columns of V are orthonormal eigenvectors of $A^t A$
- Eigenvalues $\lambda_1, \cdots, \lambda_r$ of AA^t are eigenvalues of $A^t A$
- Singular values $\sigma_i = \sqrt{\lambda_i}$ and

Low-rank Approximation

- SVD can be used to compute optimal **low-rank approximations**.
- Approximation problem: Find $\boldsymbol{A_k}$ of rank $\boldsymbol{k}$ such that

$$A_k = \min_{X:\mathrm{rank}(X)=k} \|A \quad X\|_F$$

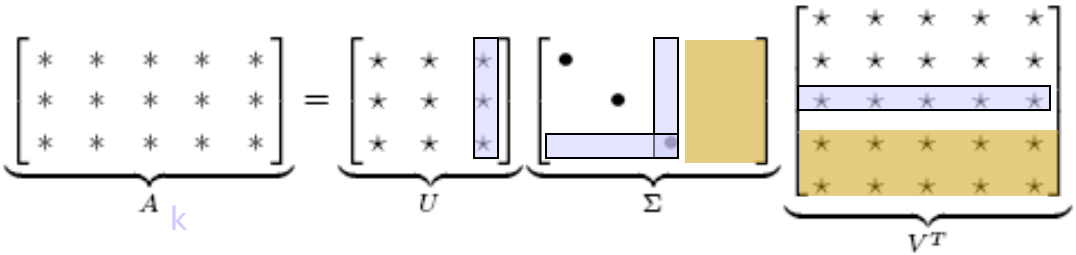
Frobenius norm

$$\|\mathrm{A}\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}\,.$$

Low-rank Approximation

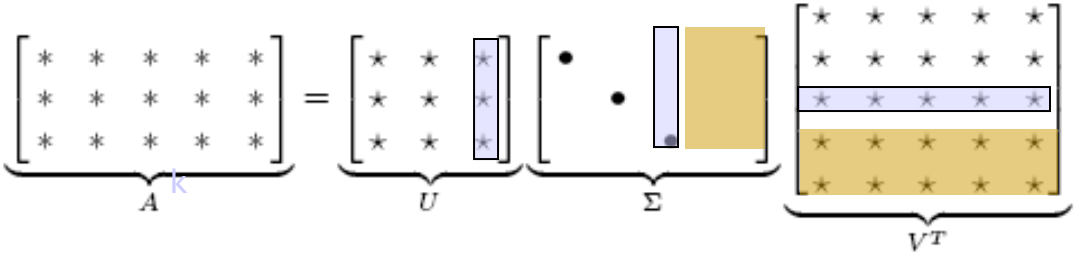
- Solution via SVD

$$A_k = U\,\mathrm{diag}(\ _1, \ldots,\ _k, \underbrace{0, \ldots, 0}_{\text{set smallest } r\text{-}k \text{ singular values to zero}})\,V^T$$



Reduced SVD

- If we retain only k singular values, and set the rest to 0, then we don't need the matrix parts in color
- Then Σ is $k \times k$, U is $M \times k$, V^T is $k \times N$, and A_k is $M \times N$
- This is referred to as the reduced SVD
- It is the convenient (space-saving) and usual form for computational applications
- It's what Matlab gives you



Approximation Error

- How good (bad) is this approximation?
- It's the best possible, measured by the Frobenius norm of the error:

$$\min_{X:\mathrm{rank}(X)=k} \|A \quad X\|_F = \|A \quad A_k\|_F = \quad {}_{k+1}$$

where the σ_i are ordered such that $\sigma_i \geq \sigma_{i+1}$.

Suggests why Frobenius error drops as *k* increases.

Approximation Error

- Maximize $||Ax||$ such that $x^t\, x = 1$
- Maximize $x^t A^t Ax$ such that $x^t\, x = 1$
- Maximize $x^t\ Bx$ such that $x^t\, x = 1$
- If x is an orthonormal eigenvector, then $Bx = \lambda x$
- So, Max $\lambda x^t\, x$ such that $x^t\, x = 1$
- Which is Max λ where λ is an eigenvalue of B
- So, the maximum λ among all the eigenvalues
- So, maximum value of $||A - A_k||_2$ is the largest singular value which is σ_{k+1}.

SVD Low-rank Approximation

- Whereas the term-doc matrix A may have M=50000, N=10 million (and rank close to 50000)
- We can construct an approximation A_{100} with rank 100.
  - <u>Of all rank 100 matrices, it would have the lowest Frobenius error.</u>
- Great … but why would we??
- Answer: *Latent Semantic Indexing*

Latent Semantic Indexing via the SVD

What it is?

- From term-doc matrix A, we compute the approximation $A_{k.}$
- There is a row for each term and a column for each doc in A_k
- Thus docs live in a space of $k<<r$ dimensions
  - These dimensions are not the original axes
- But why?

Vector Space Model: Pros

- **Automatic** selection of index terms
- **Partial matching** of queries and documents *(dealing with the case where no document contains all search terms)*
- **Ranking** according to **similarity score** *(dealing with large result sets)*
- **Term weighting** schemes *(improves retrieval performance)*
- Various extensions

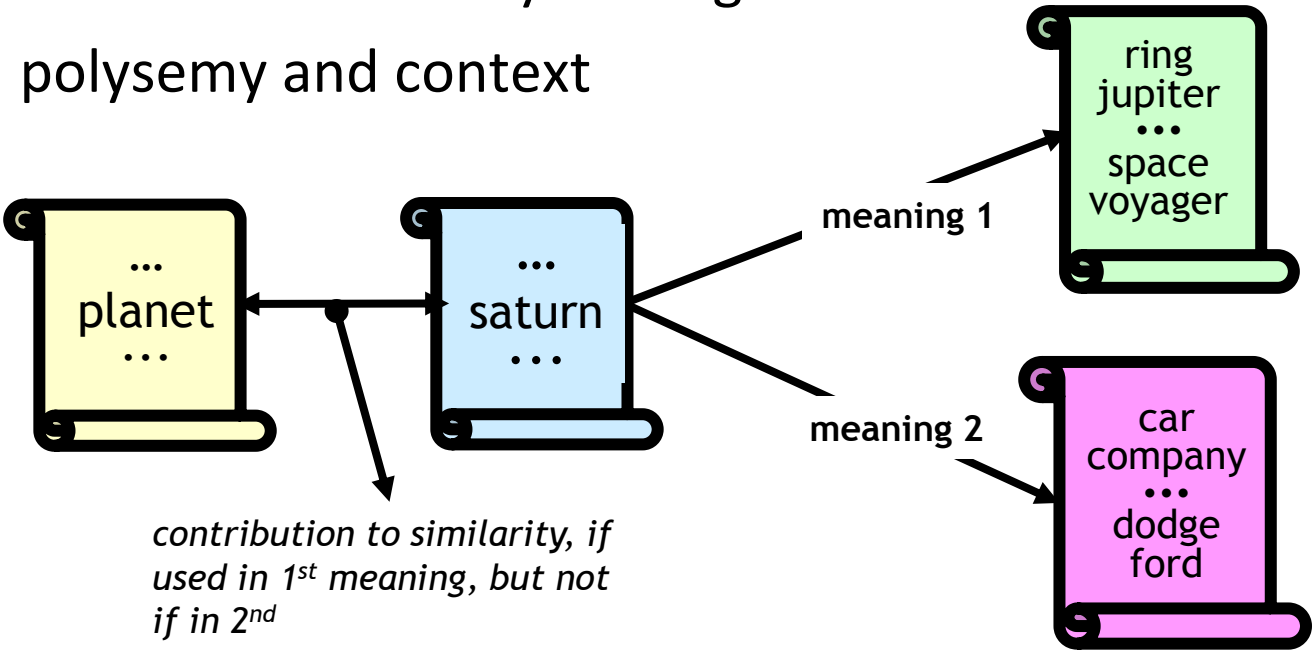
Problems with Lexical Semantics

- **Synonymy**: Different terms may have an **identical or a similar meaning** (weaker: words indicating the same topic).
- No associations between words are made in the vector space representation.

$$\mathrm{sim}_{\mathrm{true}}(d, q) > \cos(\angle(\vec{d}, \vec{q}))$$

Polysemy and Context

- Document similarity on single word level: polysemy and context



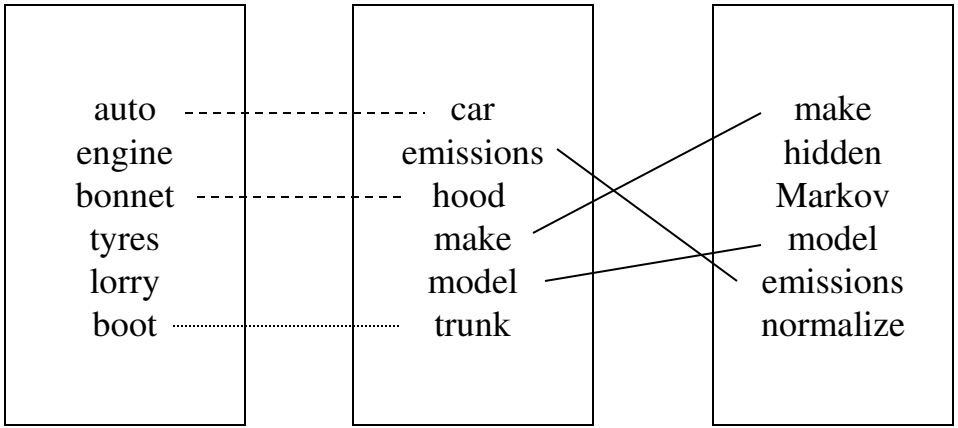
Problems with Lexical Semantics

- Ambiguity and association in natural language
  - **Polysemy**: Words often have a **multitude of meanings** and different types of usage *(more severe in very heterogeneous collections).*
  - The vector space model is unable to discriminate between different meanings of the same word.

- $sim_{true}(\mathrm{d}, \mathrm{q}) < \cos(\measuredangle\,(d, q))$

The Problem

- Example: Vector Space Model



Synonymy

Will have small cosine

but are related

Polysemy

Will have large cosine

but not truly related

- Perform a ***low-rank approximation*** of ***document-term matrix*** (typical rank **100-300**)
- General idea
  - Map documents (*and* terms) to a **low-dimensional** representation.
  - Design a mapping such that the low-dimensional space reflects **semantic associations** (latent semantic space).
  - Compute document similarity based on the **inner product** in this **latent semantic space**

Latent Semantic Indexing (LSI)

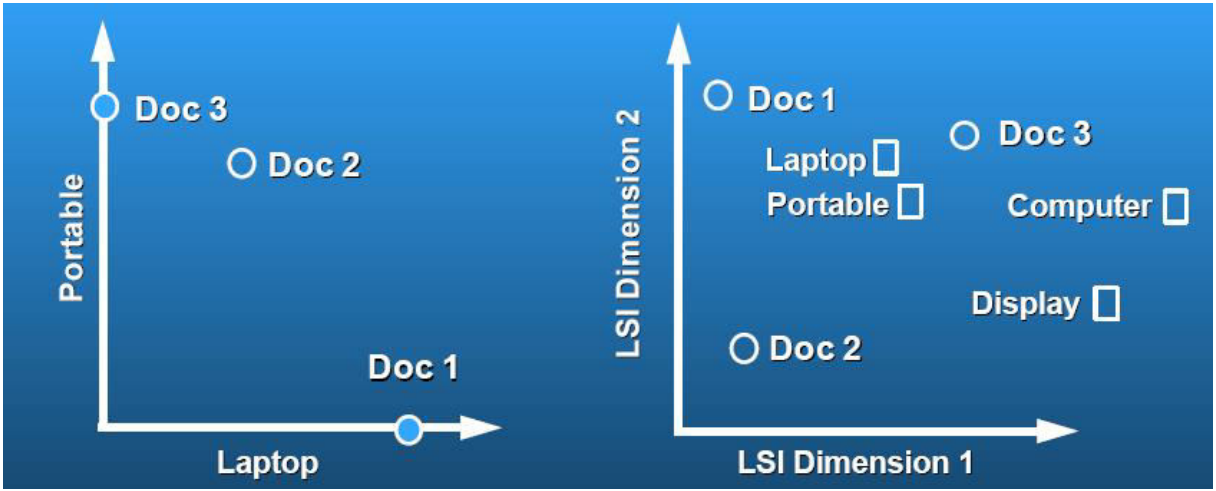
- Perform a **low-rank approximation** of **document-term matrix** (typical rank **100–300**)
- General idea
  - Map documents (*and* terms) to a **low-dimensional** representation.
  - Design a mapping such that the low-dimensional space reflects **semantic associations** (latent semantic space).
  - Compute document similarity based on the **inner product** in this **latent semantic space**

Goals of LSI

- LSI takes documents that are semantically similar (= talk about the same topics), but are not similar in the vector space (because they use different words) and re-represents them in a reduced vector space in which they have higher similarity.
- Similar terms map to similar location in low dimensional space
- Noise reduction by dimension reduction

Latent Semantic Analysis

- **Latent semantic space**: illustrating example



courtesy of Susan Dumais

Performing the maps

- Each row and column of *A* gets mapped into the *k*-dimensional LSI space, by the SVD.
- is not only the mapping with the Claim – this best (Frobenius error) approximation to *A*, but in fact *improves* retrieval.
- A query *q* is also mapped into this space, by
  - Query NOT a sparse vector. $q_k = q^T U_k \Sigma_k^{-1}$

LSA Example

- A simple example term-document matrix

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

LSA Example

- Example of $C = U\Sigma V^t$:
- The matrix U

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

LSA Example

- Example of $C = U\Sigma V^t$:
- The matrix Σ

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

LSA Example

- Example of $C = U\Sigma V^t$:
- The matrix V^T

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

LSA Example: Reducing the dimension

U	1	2	3	4	5
ship	−0.44	−0.30	0.00	0.00	0.00
boat	−0.13	−0.33	0.00	0.00	0.00
ocean	−0.48	−0.51	0.00	0.00	0.00
wood	−0.70	0.35	0.00	0.00	0.00
tree	−0.26	0.65	0.00	0.00	0.00

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00

Original matrix C vs. reduced $C_2 = U\Sigma_2 V^T$

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

Latent Semantic Analysis

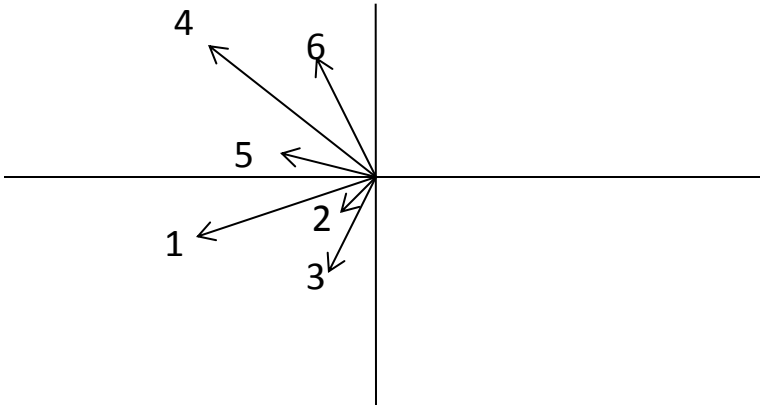
- **Latent semantic space**: illustrating example

	d1	d2	d3	d4	d5	d6
Ship	1	0	1	0	0	0
Boat	0	1	0	0	0	0
Ocean	1	1	0	0	0	0
Voyage	1	0	0	1	1	0
Trip	0	0	0	1	0	1

- The reduced dimensional data in 2-d is given below

	d1	d2	d3	d4	d5	d6
SV1	-1.62	-0.6	-0.44	-0.97	-0.7	-0.26
SV2	-0.46	-0.84	-0.3	1.0	0.35	0.65

LSA



- d2 is similar to d3
  - d2: boat, ocean; d3: ship
- d4 is similar to d5 and d6 ($\cos\theta$)
  - d4: voyage, trip; d5: voyage; d6: trip

Why the reduced dimension matrix is better?

- Similarity of d2 and d3 in the original space: 0.
- Similarity of d2 and d3 in the reduced space: $0.52 * 0.28 + 0.36 * 0.16 + 0.72 * 0.36 + 0.12 * 0.20 + -0.39 * -0.08 \approx 0.52$

- Typically, LSA increases recall and hurts precision

Empirical Evidence

- Experiments on TREC 1/2/3 – Dumais
- Lanczos SVD code (available on netlib) due to Berry used in these experiments
  - Running times of ~ one day on tens of thousands of docs [still an obstacle to use!]
- Dimensions – various values 250-350 reported. Reducing *k* improves recall.
  - (Under 200 reported unsatisfactory)
- Generally expect recall to improve – what about precision?

Empirical evidence

- Precision at or above median TREC precision
  - Top scorer on almost 20% of TREC topics
- Slightly better on average than straight vector spaces
- Effect of dimensionality:

Dimensions	Precision
250	0.367
300	0.371
346	0.374

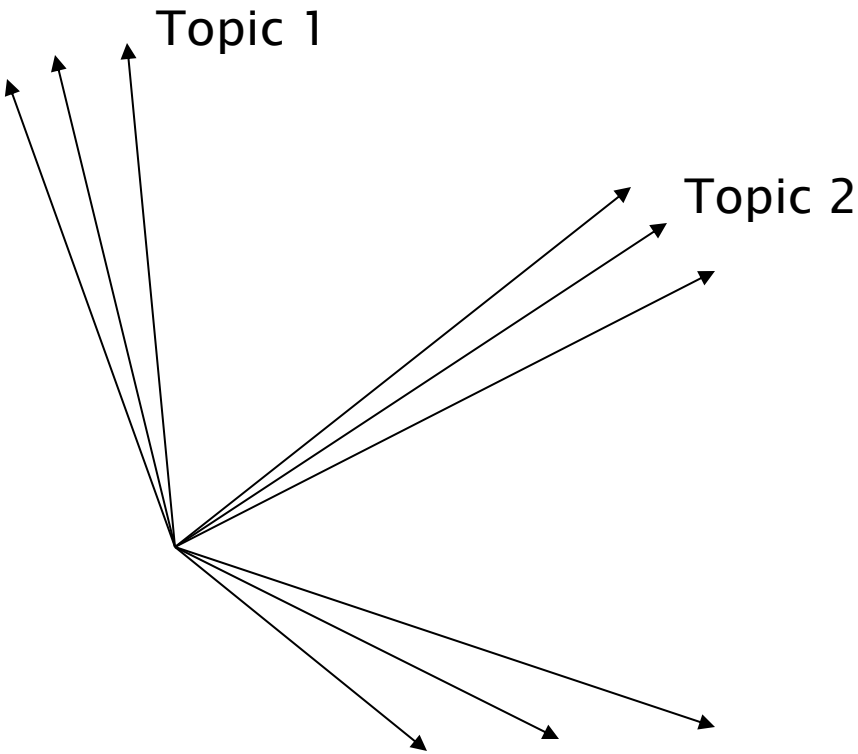
Failure modes

- Negated phrases
  - TREC topics sometimes negate certain query/terms phrases – precludes simple automatic conversion of topics to latent semantic space.
- Boolean queries
  - As usual, freetext/vector space syntax of LSI queries precludes (say) “Find any doc having to do with the following 5 companies”
- See Dumais for more.

But why is this clustering?

- We've talked about docs, queries, retrieval and precision here.
- What does this have to do with clustering?
- Intuition: Dimension reduction through LSI brings together "related" axes in the vector space.

Simplistic picture



Some wild extrapolation

- The "dimensionality" of a corpus is the number of distinct topics represented in it.
- More mathematical wild extrapolation:
  - if A has a rank k approximation of low Frobenius error, then there are no more than k distinct topics in the corpus.

LSI has many other applications

- In many settings in pattern recognition and retrieval, we have a feature-object matrix.
  - For text, the terms are features and the docs are objects.
  - Could be opinions and users …
  - This matrix may be redundant in dimensionality.
  - Can work with low-rank approximation.
  - If entries are missing (e.g., users' opinions), can recover if dimensionality is low.
- Powerful general analytical technique
  - Close, principled analog to clustering methods.

Principal Components

- Most Popular Feature Extraction Scheme in
  - Signal Processing
  - Pattern Recognition
  - Machine Learning
- Each new feature is a linear combination of the original features
- Let $X_1, X_2, \ldots, X_n$ be n D-dimensional patterns.
- Any such D-dimensional vector is represented as

$$X = \sum_{j=1}^{D} \beta_j P_j \qquad\qquad Y = \sum_{j=1}^{d} \beta_j P_j$$

Where d < D ; β_j and P_j are the weight and basis vectors;
X and Y are the actual and approximate representations.

Characterization of Error

- Error $= E\{(X-Y)^t(X-Y)\}$

  $= E\{\sum_{j=d+1}^{D} \beta_j^2\}$ $\qquad P_i, P_j$ are orthonormal

- Error = $E\{\sum_{d+1}^{D} \beta_j \beta_j^t\} = E\{\sum_{d+1}^{D} P_j^t X X^t P_j\}$

This is because $\beta_j = \beta_j^t = X^t P_j$

$$P_j^t X = P_j^t \sum_{i=1}^{D} \beta_i P_i = \beta_j$$

- So, Error = $\sum_{j=d+1}^{D} P_j^t \Omega P_j$

Eigenvectors of the Covariance Matrix

The covariance matrix $\Omega = E\{XX^t\}$

By looking at the eigenvectors of the covariance matrix, we have

$$\boldsymbol{\Omega P_j} = \lambda_j \boldsymbol{P_j}$$

So, error = $\sum_{j=d+1}^{D} P_j^t \Omega P_j = \sum_{j=d+1}^{D} P_j^t P_j \lambda_j = \sum_{d+1}^{D} \lambda_j$

So, error can be minimized by choosing the least eigenvalues for the entries in the sum above. So, the eigenvalues are

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq \lambda_{d+1} \geq \cdots \geq \lambda_D$$

Example

Consider six 2-dimensional patterns:

(1,2), (2,1), (2,2), (6,7), (7,6), (6,6)

$$\mu = \frac{1}{6}\sum_{i=1}^{6} X_i = \frac{1}{6}(24,24) = (4,4)$$

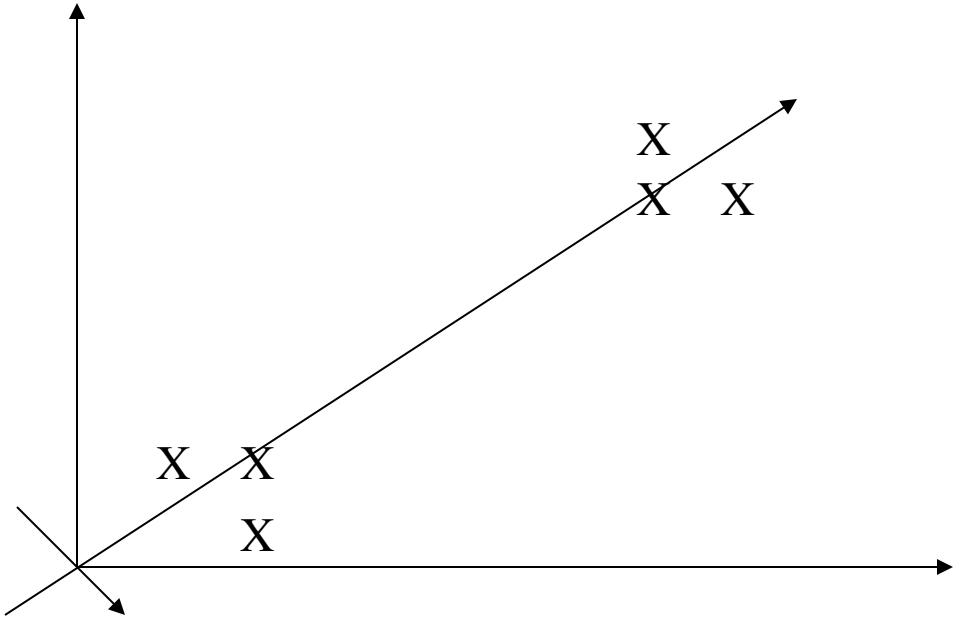
$$\Omega = \frac{1}{6}\sum_{1}^{6}(X_i - \mu)(X_i - \mu)^t = \begin{pmatrix} \frac{17}{3} & \frac{16}{3} \\ \frac{16}{3} & \frac{17}{3} \end{pmatrix}$$

The eigenvalues and eigenvectors are:

$$\lambda_1 = 11; \lambda_2 = \frac{1}{3}$$

$$P_1 = (1,1); P_2 = (1,-1)$$

The two eigenvector directions are shown below:



Let $A = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$ $\quad A^t A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$

$$\lambda_1 = 3; \lambda_2 = 1$$

Thus *M*=3, *N*=2. Its SVD is

$$\begin{bmatrix} 0 & 2/\sqrt{6} & 1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{6} & 1/\sqrt{3} \\ 1/\sqrt{2} & 1/\sqrt{6} & -1/\sqrt{3} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{3} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

Let D = $\begin{pmatrix}1\\2\end{pmatrix}\begin{pmatrix}2\\1\end{pmatrix}\begin{pmatrix}2\\2\end{pmatrix}\begin{pmatrix}6\\7\end{pmatrix}\begin{pmatrix}7\\6\end{pmatrix}\begin{pmatrix}6\\6\end{pmatrix}$

$$\mu = \begin{pmatrix}4\\4\end{pmatrix}$$

$$X = \begin{bmatrix} -3 & -2 \\ -2 & -3 \\ -2 & -2 \\ 2 & 3 \\ 3 & 2 \\ 2 & 2 \end{bmatrix}$$

$$X^t X = \begin{bmatrix} 34 & 32 \\ 32 & 34 \end{bmatrix}$$

$$\lambda_1 = 66; \lambda_2 = 2$$

If normalized then the eigenvalues are 11 and 1/3

And the eigenvectors are $\begin{pmatrix}1\\1\end{pmatrix}$ $\begin{pmatrix}1\\-1\end{pmatrix}$

$$\begin{bmatrix} 34 & 32 \\ 32 & 34 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 66 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

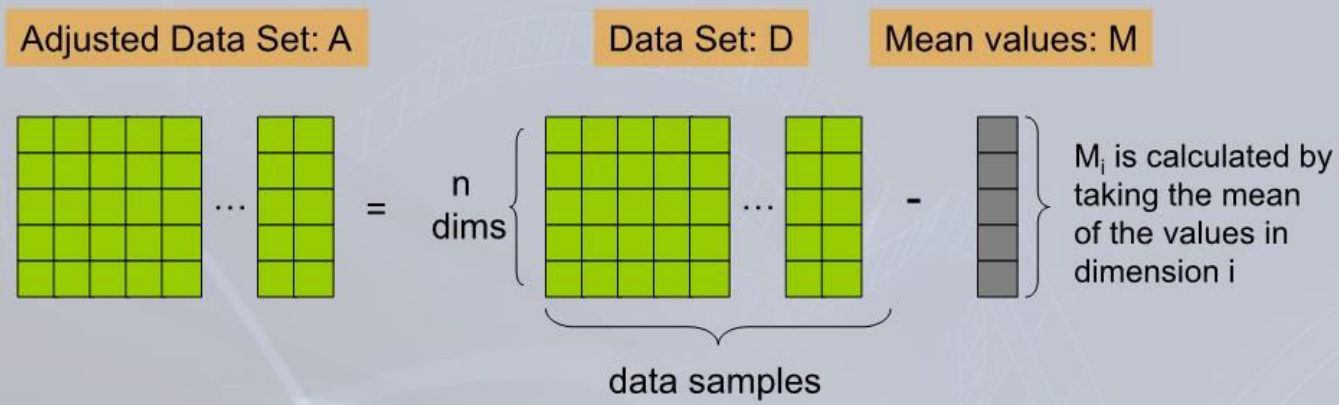
$$Ax = \lambda x$$

$$A^2 x = A(Ax) = A(\lambda x) = \lambda Ax = \lambda^2 x$$

SVD may be used in PCA.

Principal Component Analysis (PCA)

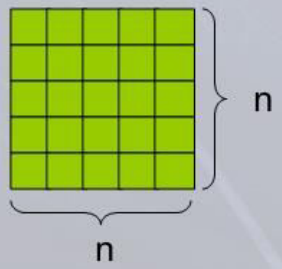
Steps in PCA: #1 Calculate Adjusted Data Set



Principal Component Analysis (PCA)

Steps in PCA: #2 Calculate Co-variance matrix, C, from Adjusted Data Set, A

Co-variance Matrix: C



$C_{ij} = cov(i,j)$

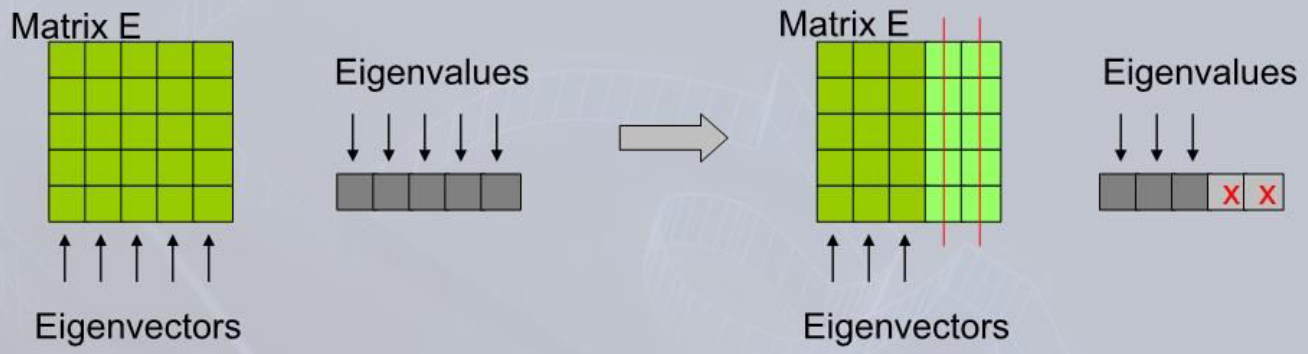
$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

Note: Since the means of the dimensions in the adjusted data set, A, are 0, the covariance matrix can simply be written as:

$$C = (A A^T) / (n-1)$$

Principal Component Analysis (PCA)

Steps in PCA: #3 Calculate eigenvectors and eigenvalues of C



If some eigenvalues are 0 or very small, we can essentially discard those eigenvalues and the corresponding eigenvectors, hence reducing the dimensionality of the new basis.

Principal Component Analysis (PCA)

Steps in PCA: #4 Transforming data set to the new basis

$$F = E^T A$$

where:
- F is the transformed data set
- E^T is the transpose of the E matrix containing the eigenvectors
- A is the adjusted data set

Note that the dimensions of the new dataset, F, are less than the data set A

To recover A from F:

$$(E^T)^{-1}F = (E^T)^{-1}E^T A$$
$$(E^T)^T F = A$$
$$EF = A$$

* E is orthogonal, therefore $E^{-1} = E^T$

PCA using SVD

Recall: In PCA we basically try to find eigenvalues and eigenvectors of the covariance matrix, C. We showed that $C = (AA^T) / (n-1)$, and thus finding the eigenvalues and eigenvectors of C is the same as finding the eigenvalues and eigenvectors of AA^T

Recall: In SVD, we decomposed a matrix A as follows:

$$A = U S V^T$$

and we showed that:

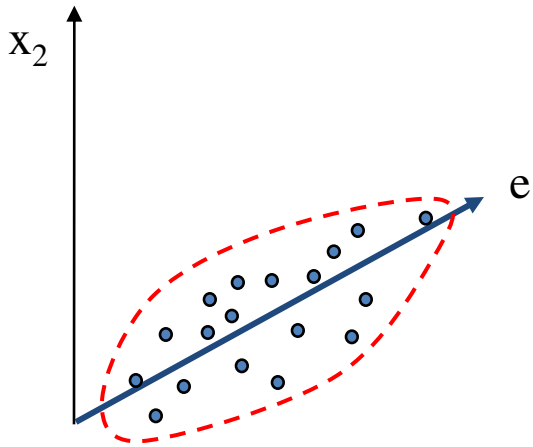
$$AA^T = U S^2 U^T$$

where the columns of U contain the eigenvectors of AA^T and the eigenvalues of AA^T are the squares of the singular values in S

Thus SVD gives us the eigenvectors and eigenvalues that we need for PCA

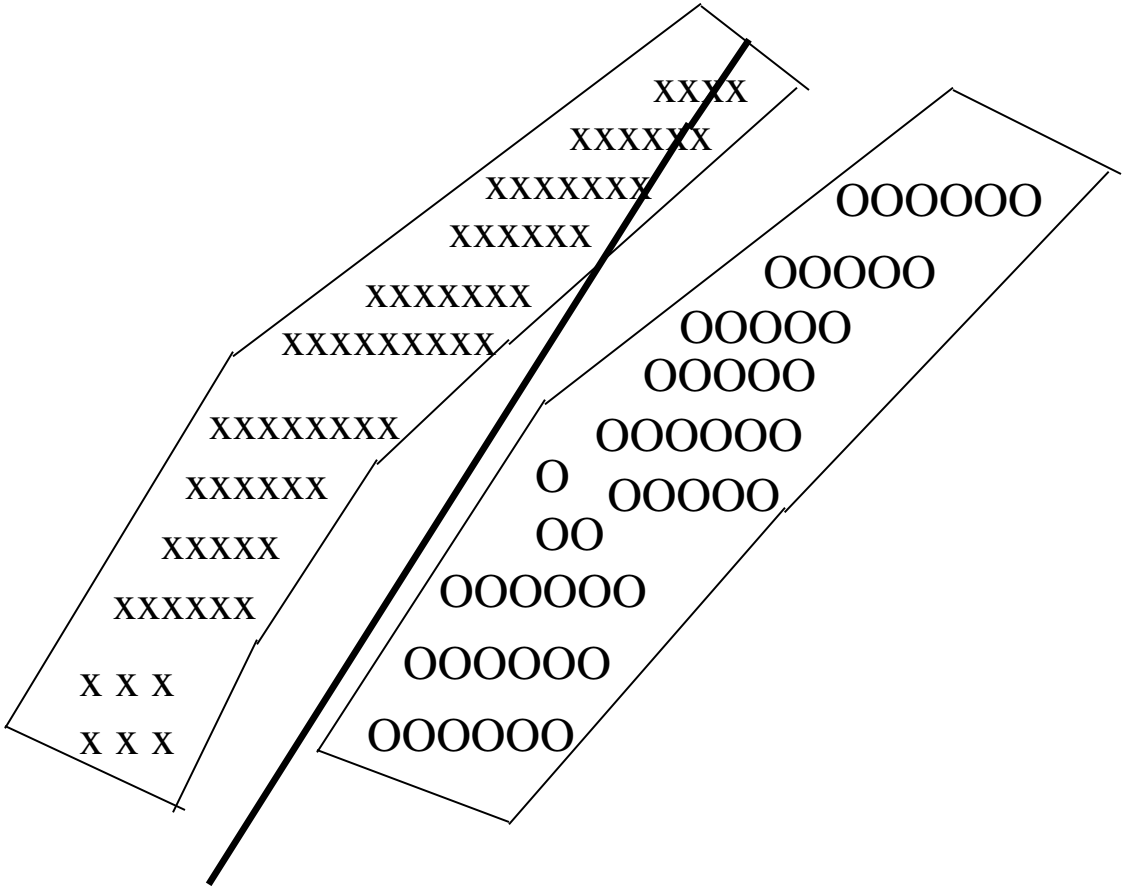
Dimensionality Reduction: PCA

- Find the eigenvectors of the covariance matrix
- The eigenvectors define the new space



Prof. Vipin Kumar

Dimensions = 206


xxxx
xxxxxx
xxxxxxx
xxxxxx
xxxxxxx
xxxxxxxxx
xxxxxxxx
xxxxxx
xxxxx
xxxxxx
x x x
x x x
OOOOOO
OOOOO
OOOOO
OOOOO
OOOOOO
O
OOOOO
OO
OOOOOO
OOOOOO
OOOOOO

Non-negative Matrix Factorization (NMF)

$$\mathbf{V} \approx \mathbf{WH}$$

$$\begin{cases} \mathbf{V} \in R^{n \times m} \\ \mathbf{W} \in R^{n \times r} \\ \mathbf{H} \in R^{r \times m} \end{cases}$$

$$\min \left\| \mathbf{V} - \mathbf{WH} \right\|^2$$

V: Data matrix.
W: Mixing matrix
H: Encoding matrix

The only constraint is **NON-NEGATIVIY** of the matrices

Only **ADDITIVE**,not subtractive, combinations are allowed

- W links documents to topics.
- H links topics to words.

Equivalance of NMF and K-means Clustering

- Objective function of K-means clustering is equivalent to:

$$\min J_{K-means} = \sum_{k=1}^{K} \sum_{i \in C_k} \left\| \mathbf{x}_i - \mathbf{w}_k \right\|^2$$

$\mathbf{x}_i$:ith sample, $\mathbf{w}_k$:kth cluster centroid, K: # of clusters

- The cth row of the encoding matrix **H** would have non-zero elements for the samples which belong to cth cluster.
- Columns of the mixing matrix **W** can be considered as cluster mean vectors

# of 1s is *t*

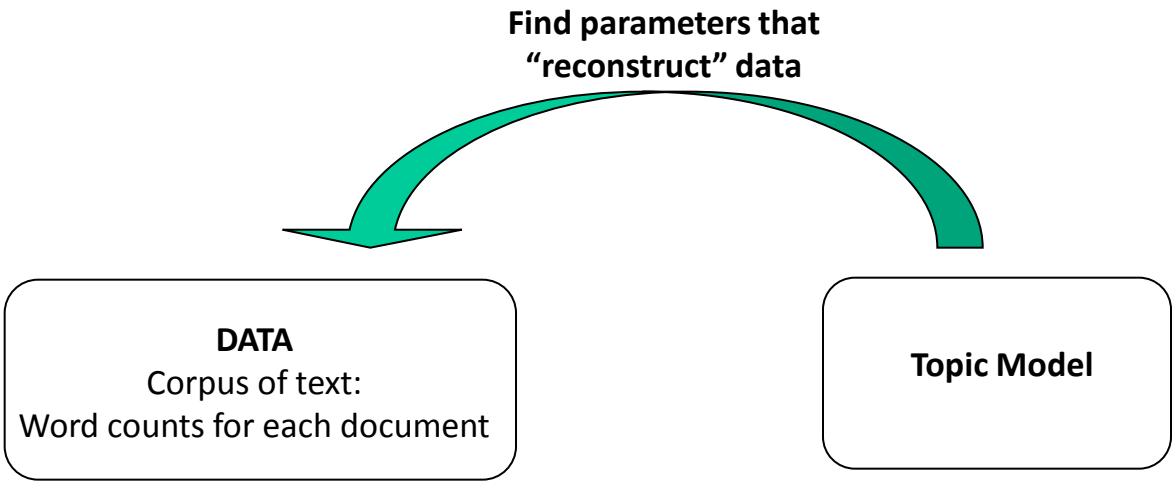
$$\mathbf{h}_c = [000...00\underbrace{11.....111}100...000]/t$$

$$\mathbf{W} = \mathbf{V}\mathbf{H}^T$$

$$\begin{bmatrix} 4 & 19 \\ 4 & 19 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 & 6 & 6 & 7 \\ 1 & 2 & 1 & 6 & 7 & 6 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Approximation

Model is Generative



Probabilistic Topic Models

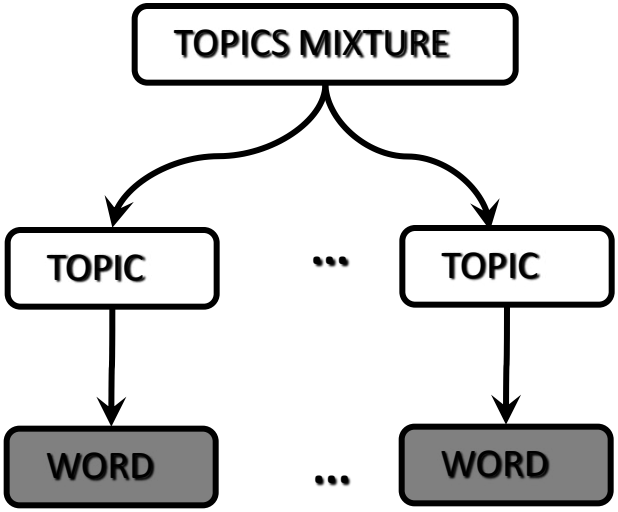
- Each document is a probability distribution over topics (distribution over topics = gist)

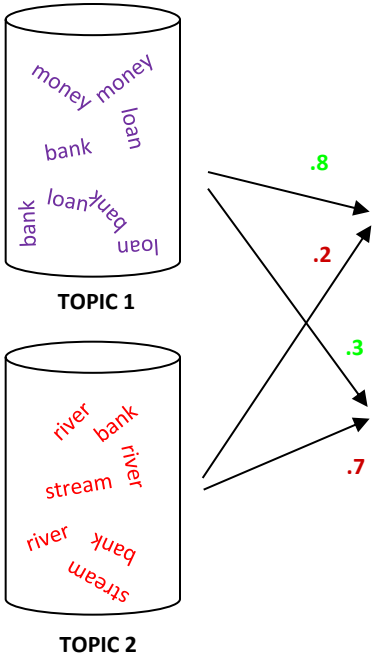
- Each topic is a probability distribution over words

1. for each document, choose
   a mixture of topics

2. For every word slot, sample a topic [1..T] from the mixture

3. sample a word from the topic





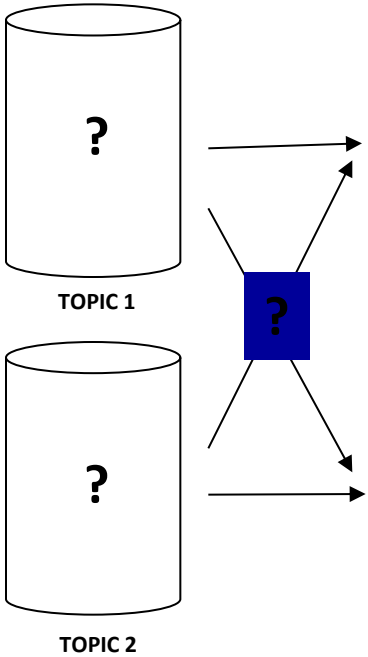
DOCUMENT 1: money[1] bank[1] bank[1] loan[1] river[2] stream[2] bank[1] money[1] river[2] bank[1] money[1] bank[1] loan[1] money[1] stream[2] bank[1] money[1] bank[1] bank[1] loan[1] river[2] stream[2] bank[1] money[1] river[2] bank[1] money[1] bank[1] loan[1] bank[1] money[1] stream[2]

DOCUMENT 2: river[2] stream[2] bank[2] stream[2] bank[2] money[1] loan[1] river[2] stream[2] loan[1] bank[2] river[2] bank[2] bank[1] stream[2] river[2] loan[1] bank[2] stream[2] bank[2] money[1] loan[1] river[2] stream[2] bank[2] stream[2] bank[2] money[1] river[2] stream[2] loan[1] bank[2] river[2] bank[2] money[1] bank[1] stream[2] river[2] bank[2] stream[2] bank[2] money[1]

Mixture components

Mixture weights

Bayesian approach: use priors



DOCUMENT 1: money? bank? bank? loan? river? stream? bank? money? river? bank? money? bank? loan? money? stream? bank? money? bank? bank? loan? river? stream? bank? money? river? bank? money? bank? loan? bank? money? stream?

DOCUMENT 2: river? stream? bank? stream? bank? money? loan? river? stream? loan? bank? river? bank? bank? stream? river? loan? bank? stream? bank? money? loan? river? stream? bank? stream? bank? money? river? stream? loan? bank? river? bank? money? bank? stream? river? bank? stream? bank? money?

Example: Topics From an Educational Corpus

(TASA: (Touchstone Applied Science Associates) Corpus- Lang., Arts, Social Studies, Health, Sciences)

- 37K docs, 26K words
- 1700 topics, e.g.:

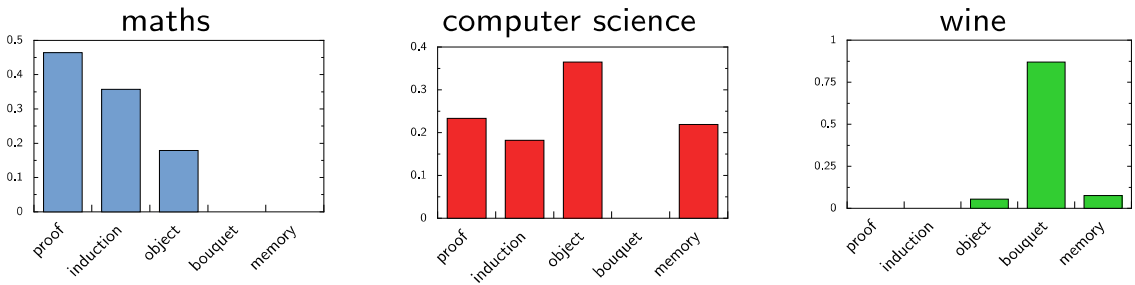
PRINTING	PLAY	TEAM	JUDGE	HYPOTHESIS	STUDY
PAPER	PLAYS	GAME	TRIAL	EXPERIMENT	TEST
PRINT	STAGE	BASKETBALL	COURT	SCIENTIFIC	STUDYING
PRINTED	AUDIENCE	PLAYERS	CASE	OBSERVATIONS	HOMEWORK
TYPE	THEATER	PLAYER	JURY	SCIENTISTS	NEED
PROCESS	ACTORS	PLAY	ACCUSED	EXPERIMENTS	CLASS
INK	DRAMA	PLAYING	GUILTY	SCIENTIST	MATH
PRESS	SHAKESPEARE	SOCCER	DEFENDANT	EXPERIMENTAL	TRY
IMAGE	ACTOR	PLAYED	JUSTICE	TEST	TEACHER
PRINTER	THEATRE	BALL	EVIDENCE	METHOD	WRITE
PRINTS	PLAYWRIGHT	TEAMS	WITNESSES	HYPOTHESES	PLAN
PRINTERS	PERFORMANCE	BASKET	CRIME	TESTED	ARITHMETIC
COPY	DRAMATIC	FOOTBALL	LAWYER	EVIDENCE	ASSIGNMENT
COPIES	COSTUMES	SCORE	WITNESS	BASED	PLACE
FORM	COMEDY	COURT	ATTORNEY	OBSERVATION	STUDIED
OFFSET	TRAGEDY	GAMES	HEARING	SCIENCE	CAREFULLY
GRAPHIC	CHARACTERS	TRY	INNOCENT	FACTS	DECIDE
SURFACE	SCENES	COACH	DEFENSE	DATA	IMPORTANT
PRODUCED	OPERA	GYM	CHARGE	RESULTS	NOTEBOOK
CHARACTERS	PERFORMED	SHOT	CRIMINAL	EXPLANATION	REVIEW

PLSI – a Probabilistic Approach

- Models each word in a document as a sample from a mixture model.

$$p(w) = \sum_{i=1}^{k} p(z_i)p(w|z_i) \qquad \text{s.t.} \quad \sum_{i=1}^{k} p(z_i) \stackrel{!}{=} 1$$

- Introduces the concept of a *topic*.



- Each word generated from one topic, different words can be generated by different topics.
- Problems: (1) *not well-defined* on documents level, (2) overfitting ($kV + kM$ parameters).

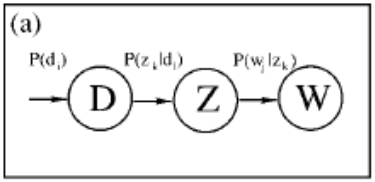
PLSI

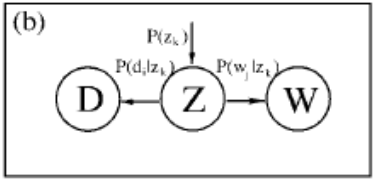
- Assume a multinomial distribution of topics (z)

$$mult(m_1, m_2, \cdots, m_k \mid \mu, n) = \binom{n}{m_1, m_2, \cdots, m_k} \Pi_{k=1}^{K} \mu_k^{m_k}$$

$$P(d_i, w_j) = P(d_i)P(w_j \mid d_i), \quad P(w_j \mid d_i) = \sum_{k=1}^{K} P(w_j \mid z_k)P(z_k \mid d_i).$$

$$P(d_i, w_j) = \sum_{k=1} P(z_k)P(d_i \mid z_k)P(w_j \mid z_k),$$





Question: How to determine z ?

Relation with LSI

- Relation

$$\mathbf{P} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$$

$$P(d,w) = \sum_{z \in Z} P(z)P(d \mid z)P(w \mid z)$$

$$\hat{\mathbf{U}} = (P(d_i \mid z_k))_{i,k} \qquad \hat{\mathbf{\Sigma}} = \mathbf{diag}(P(z_k))_{k} \qquad \hat{\mathbf{V}} = (P(w_j \mid z_k))_{j,k}$$

- Difference:
  - LSI: minimize L-2 norm
  - pLSI: maximize log-likelihood of training data

Problem of PLSI

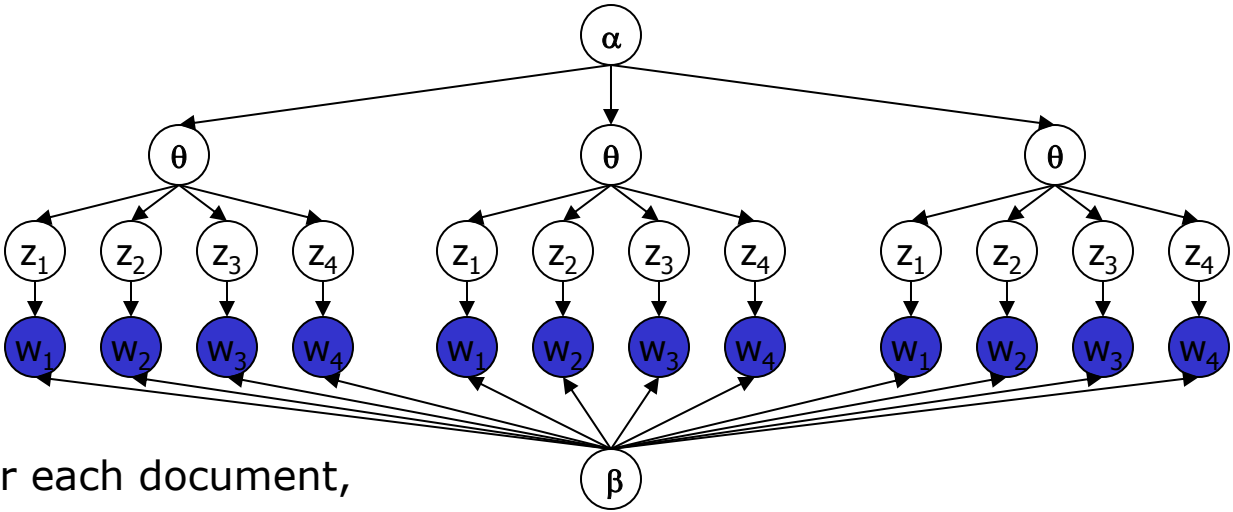
- It is not a proper generative model for document:
  - Document is generated from a mixture of topics
- The number of topics may grow linearly with the size of the corpus
- Difficult to generate a new document

Dirichlet Distributions

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} \theta_i^{\alpha_i - 1}$$

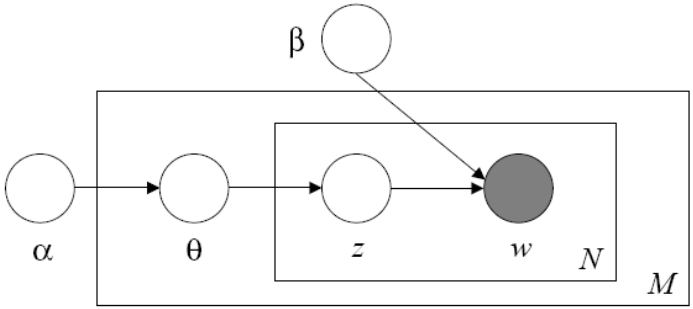
- Useful Facts:
  - This distribution is defined over a (k-1)-simplex. That is, it takes k non-negative arguments which sum to one. Consequently it is a natural distribution to use over multinomial distributions.
  - In fact, the Dirichlet distribution is the conjugate prior to the multinomial distribution. (This means that if our likelihood is multinomial with a Dirichlet prior, then the posterior is also Dirichlet!)
  - The Dirichlet parameter α_i can be thought of as a prior count of the ith class.

The LDA Model



- For each document,
- Choose $\theta \sim \text{Dirichlet}(\alpha)$
- For each of the N words wn:
  - Choose a topic z_n» Multinomial(θ)
  - Choose a word w_n from $p(w_n|z_n,\beta)$, a multinomial probability conditioned on the topic z_n.

The LDA Model

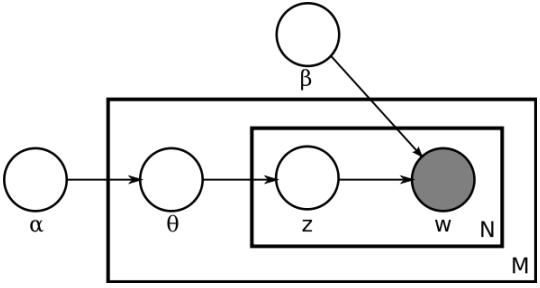


For each document,

- Choose θ» Dirichlet(α)
- For each of the N words w_n:
  - Choose a topic z_n» Multinomial(θ)
  - Choose a word w_n from $p(w_n|z_n,\beta)$, a multinomial probability conditioned on the topic z_n.

LDA (Latent Dirichlet Allocation)

- Document = mixture of topics (as in pLSI), but according to a Dirichlet prior
  - When we use a uniform Dirichlet prior, pLSI=LDA
- A word is also generated according to another variable β: $\beta_{ij} = p(w^j = 1 \mid z^i = 1)$



$$p(\mathbf{w} \mid \alpha, \beta) = \int p(\theta \mid \alpha) \left(\prod_{n=1}^{N} \sum_{z_n} p(z_n \mid \theta) p(w_n \mid z_n, \beta) \right) d\theta.$$

$$p(D \mid \alpha, \beta) = \prod_{d=1}^{M} \int p(\theta_d \mid \alpha) \left(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn} \mid \theta_d) p(w_{dn} \mid z_{dn}, \beta) \right) d\theta_d.$$

Performance of LDA on Reuters Data

About the experiments

- 100-topic LDA trained on a 16'000 documents corpus of news articles by Reuters (the news agency).
- Some standard stop words removed.

Top words from some of the $p(w|z)$

"Arts"	"Budgets"	"Children"	"Education"
new	million	children	school
film	tax	women	students
show	program	people	schools
music	budget	child	education
movie	billion	years	teachers
play	federal	families	high
musical	year	work	public

Knowledge is used in

- Refining query
- Collecting the documents
- Representing query and documents
- Indexing documents
- *Finding similarity and in ranking*
- *Grouping and Classification*

(B1 "00573"...) (B2 "00631"...) (B3 "005113"...) (B4 "004"...) (B5 "'00514"...) (B6 "005133"...)
(B7 "00513"...) (B8 "005131"...) (B9 "00435"...) (B10 "'005133"...) (Bll "005133"...) (B12 "0051".
(B13 "004"...) (B14 "006454"...) (B15 "00654"...) (B16 "00637"...) (B17 "0066"...) (B18 "00642".
(B19 "0063"...) (B20 "0063"...)(B21 "00637"...) (B22 "0063“…) (B23 "0063"...) (B24 "00435"...
(B25 "00633"...)(B26 "0063"...)(B27 "00633“…) (B28 "005133"...) (B29 "004125"...) (B30 "00637
(B31 "005265"...) (B32 "0046“…)(B33 "005133"...)(B34 "00516"...)(B35 "01M35"...) (B36 "00435"
(B37 "0051“…)(B38 "005133"...)(B39 "005133"...)(B40 "00574"...)(B41 "00435" . .)(B42 "005756
(B43 "00422"...)(B44 "004"...)(B45 "004"...)(B46"0063“. ..)(B47 "00435“…)(B48 "0046", . .)
(B49 "00543"...)(B50 "00513"...)(B51 "0046"..) (B52 "00435“…)(B53 "0051"...)(B54 "005131"..
(B55 "00543"...)(B56 "0043Y' . .)(B57 "004165“...) (B58 "005"...) (B59 "0057"...) (B60 "0046"..
(B61 "0051"..) (B62 "00516". ,) (B63 "00511"...) (B64 "0063"...) (B65 "004"...)(B66 "004"..)
(B67 "00575"...) (B68 "00424"...) (B69 "00436",..) (B70 "00421"...)(B71 "00574" . ,) (B72 "004".
(B73 "00633"...)(B74 "00512028"...)(B75 "005133"...)(B76 "00435" . .)(B77 "00435"...)(B78 "0042
(B79 "005131"...) (B80 "004"...)(B81 "00574"' . .) (B82 "00543".. ,) (B83 "00543"...) (B84 "0066'
(B85 "00435")(B86 "00513Y'...) (B87 "00526"...) (B88 "005265"...) (B89 "0063"., .) (B90 "00435'
(B91 "00512"'...) (B92 "00468".. ,) (B93 "00436"...) (B94 "0051",..) (B95 "0066"...)(B96 "00411".
(B97 "0051"...) (B98 "00574".. ,) (B99 "0046"...) (B100 "00513"...)(BI01 "00574"...) (BI02 "00435
(B103 "00435"...)(B104 "005453"...)(B105 "0057"'...)(B106 "'00435"...)(B107 "00411". , .)
(B108 "00512"...)

• (rank library1 library2 library3 “PROGRAMMING”)
Rank 1 Library LIBRARY1 with 17 books
Rank 2 Library LIBRARY2 with 15 books
Rank 3 Library LIBRARY3 with 9 books

• (rank library1 library2 library3 “ARTIFICIAL INTELLIGENCE”)
Rank 1 Library LIBRARY1 with 12 books
Rank 2 Library LIBRARY3 with 9 books
Rank 3 Library LIBRARY2 with 4 books

• (rank library1 library2 library3 "DATA PROCESSING")
Rank 1 Library LIBRARY2 with 31 books
Rank 2 Library LIBRARYI with 19 books
Rank 3 Library LIBRARY3 with 16 books

• Instead of simply taking counts, now we can use the number of *citations to these books using Google Scholar* to get a better characterization of the impact of the collection in a library in a given area.
• Further, one can use Amazon.com to get significant related books for strengthening an area in a given library.
• One can use the Table of Contents to get books that complement each other or at different levels.

Wikipedia, the free encyclopedia - Mozilla Firefox

y Bookmarks Tools Help

http://en.wikipedia.org/wiki/Pattern_recognition

pattern recognition

mize Links Free Hotmail Windows Marketplace Windows Media Windows

Log in / create

Article Discussion Read Edit View history Search

Pattern recognition

From Wikipedia, the free encyclopedia

For other uses, see Pattern recognition (disambiguation).

In machine learning, **pattern recognition** is the assignment of some sort of output value (or *label*) to a given input value (or *instance*), according to some specific algorithm. An example of pattern recognition is classification, which attempts to assign each input value to one of a given set o *classes* (for example, determine whether a given email is "spam" or "non-spam"). However, pattern recognition is a more general problem that encompasses other types of output as well. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence); and parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence.

Pattern recognition algorithms generally aim to provide a reasonable answer for all possible inputs and to do "fuzzy" matching of inputs. This is opposed to *pattern matching* algorithms, which look for exact matches in the input with pre-existing patterns. A common example of a pattern-ma algorithm is regular expression matching, which looks for patterns of a given sort in textual data and is included in the search capabilities of many text editors and word processors. In contrast to pattern recognition, pattern matching is generally not considered a type of machine learning, a pattern-matching algorithms (especially with fairly general, carefully tailored patterns) can sometimes succeed in providing similar-quality output to the sort provided by pattern-recognition algorithms.

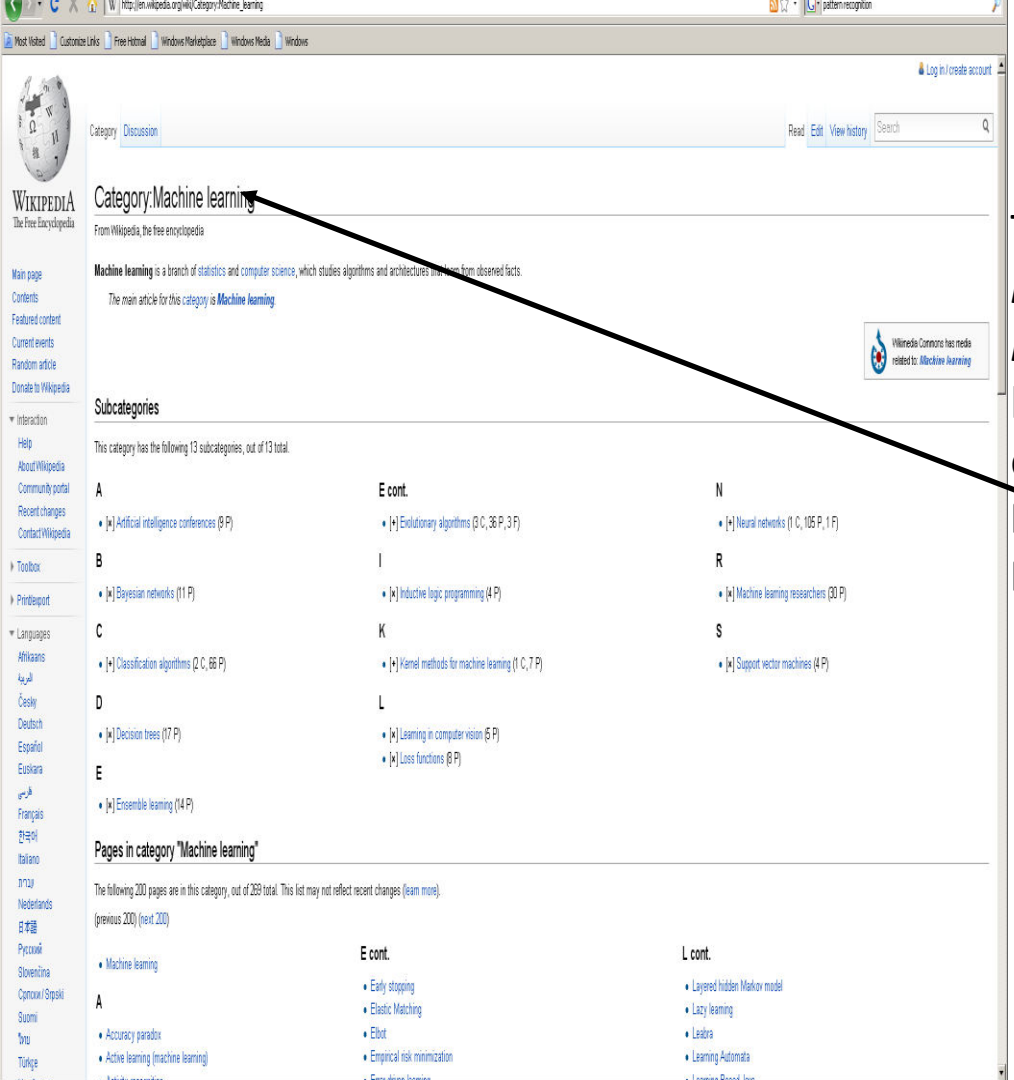
Pattern recognition is studied in many fields, including psychology, psychiatry, ethology, cognitive science, traffic flow and computer science.

Contents [hide]

1 Overview
2 Problem statement (supervised version)
3 Uses
4 Algorithms
4.1 Classification algorithms (supervised algorithms predicting categorical labels)
4.2 Clustering algorithms (unsupervised algorithms predicting categorical labels)
4.3 Regression algorithms (predicting real-valued labels)
4.4 Categorical sequence labeling algorithms (predicting sequences of categorical labels)
4.5 Real-valued sequence labeling algorithms (predicting sequences of real-valued labels)
4.6 Parsing algorithms (predicting tree structured labels)
4.7 General algorithms for predicting arbitrarily-structured labels
4.8 Ensemble learning algorithms (supervised meta-algorithms for combining multiple learning algorithms together)
5 See also
6 References
7 Further reading
8 External links

Overview

Pattern recognition is generally categorized according to the type of learning procedure used to generate the output value. *Supervised learning* assumes that a set of *training data* (the *training set*) has been provided, consisting of a set of instances that have been properly labeled by hand v



Category:Machine learning

From Wikipedia, the free encyclopedia

Machine learning is a branch of statistics and computer science, which studies algorithms and architectures that learn from observed facts.

The main article for this category is ***Machine learning***

Subcategories

This category has the following 13 subcategories, out of 13 total.

A
- [+] Artificial intelligence conferences (9 P)

B
- [+] Bayesian networks (11 P)

C
- [+] Classification algorithms (2 C, 66 P)

D
- [+] Decision trees (17 P)

E
- [+] Ensemble learning (14 P)

E cont.
- [+] Evolutionary algorithms (3 C, 36 P, 3 F)

I
- [+] Inductive logic programming (4 P)

K
- [+] Kernel methods for machine learning (1 C, 7 P)

L
- [+] Learning in computer vision (5 P)
- [+] Loss functions (8 P)

N
- [+] Neural networks (1 C, 105 P, 1 F)

R
- [+] Machine learning researchers (30 P)

S
- [+] Support vector machines (4 P)

Pages in category "Machine learning"

The following 200 pages are in this category, out of 369 total. This list may not reflect recent changes (learn more).

(previous 200) (next 200)

- Machine learning

A
- Accuracy paradox
- Active learning (machine learning)

E cont.
- Early stopping
- Elastic Matching
- Elbot
- Empirical risk minimization

L cont.
- Layered hidden Markov model
- Lazy learning
- Leabra
- Learning Automata

The concept ***Pattern Recognition*** belongs to two categories: **Machine Learning**, **Formal sciences**

The different meanings that **Pattern Recognition** may refer to are listed in its disambiguation page.

Pattern recognition (disambiguation) - Wikipedia, the free encyclopedia - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://en.wikipedia.org/wiki/Pattern_recognition_(disambiguation)

Most Visited | Customize Links | Free Hotmail | Windows Marketplace | Windows Media | Windows

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Article | Discussion | Read | Edit | View history

Pattern recognition (disambiguation)

From Wikipedia, the free encyclopedia

Pattern recognition is a field about machine learning.

Pattern recognition may also refer to:

- Pattern recognition (psychology)
- *Pattern Recognition* (novel), a novel by William Gibson
- *Pattern Recognition*, an album by Sea Scouts
- "Pattern Recognition", a song by Sonic Youth from *Sonic Nurse*

This disambiguation page lists articles associated with the same title.
If an internal link led you here, you may wish to change the link to point directly to the intended article.

Categories: Disambiguation pages

This page was last modified on 12 April 2011 at 11:50.

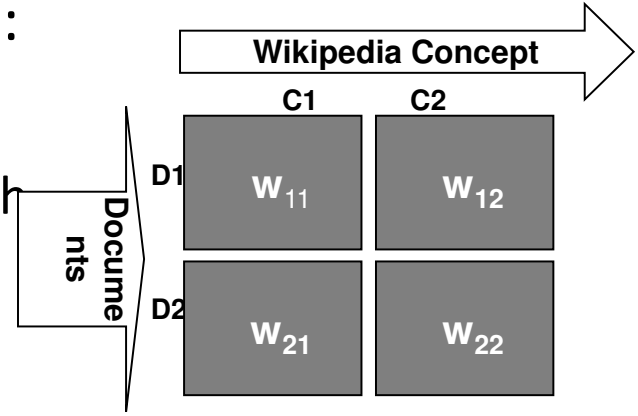
Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details.
Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Contact us

Privacy policy | About Wikipedia | Disclaimers

Concept Mapping

- Match documents to Wikipedia concepts.
- Result: A document-concept Matrix
- Matching Schemes:
  - Exact Match
  - Relatedness Match



d1

Pattern Recognition, or **PR** for short, is a combination of **computer science**, **classification**, and **philosophy**. …

Exact match

Wikipedia concept dictionary

Pattern Recognition, PR
…
computer science
…
classification
…
philosophy
…

document-concept Vector

	Wiki Concepts C							
	Pattern Recognition	**…**	**computer science**	**…**	**classification**	**…**	**philosophy**	**…**
d1	2	…	1	…	1	…	1	…

Concept Mapping Schemes: Exact Match

- Each document is scanned to find Wikipedia concepts (article titles).
- The searched Wikipedia concepts are used to comprise the concept vector of the corresponding document.
- Synonymous phrases to the same concept are grouped together through the redirect links in Wikipedia.

Concept Mapping Schemes: Exact Match

- **A dictionary** is constructed, with each entry corresponding to a topic covered by Wikipedia.
- Each entry includes not only the **preferred Wikipedia concept** which is used as the title of the article, but also all the **redirected concepts** representing the same topic.
- Based on the dictionary, both preferred concepts and redirected concepts are retrieved from documents.
- Only preferred concepts are used to build the **concept vector for each document**. The **weight** of each preferred concept equals to the frequency of itself plus the frequencies of all the redirected concepts appearing in a document.
- The document-concept TFIDF matrix is further calculated based on the document-concept frequency matrix.

Pattern Recognition, or **PR** for short, is a combination of **computer science**, **classification**, and **philosophy**. …

term vector of document d1

terms *W*

	w_1	w_2	…	w_N
d1	tfidf11	tfidf12		tfidf1N

×

Wiki article terms *W*

Wikipedia term concept matrix

Wiki concepts C

	c_1	c_2	…	c_M
w_1	$tfidf_{11}$	$tfidf_{12}$	…	$tfidf_{1N}$
w_2	$tfidf_{21}$	$tfidf_{22}$	…	$tfidf_{2N}$
…	…	…	…	…
w_N	$tfidf_{N1}$	$tfidf_{N2}$	…	$tfidf_{NM}$

document-concept Vector

Wiki concepts C

	c_1	c_2	…	c_M
d1	r_{11}	r_{12}		r_{1N}

Concept Mapping: Relatedness Match

Step 1: A **Wikipedia term-concept matrix** is constructed from Wikipedia article collection.

		Wiki concepts C			
		c_1	c_2	...	c_N
Wiki article terms *W*	w_1	val_{11}	val_{12}	...	val_{1N}
	w_2	val_{21}	val_{22}	...	val_{2N}
	...	...	...	...	...
	w_N	val_{N1}	val_{N2}	...	val_{NN}

- The values in the matrix are TFIDF scores, which denote the relatedness of each term to each Wikipedia concept article.
- For each word, only top *k* concepts with highest *TFIDF* scores are selected. (For example, k = 5)

Concept Mapping: Relatedness Match

Step 2: The word-concept matrix is used as a bridge to build **Document-Wikipedia concept Matrix**. The relatedness of a Wikipedia concept c_k to a given document d_j is calculated as:

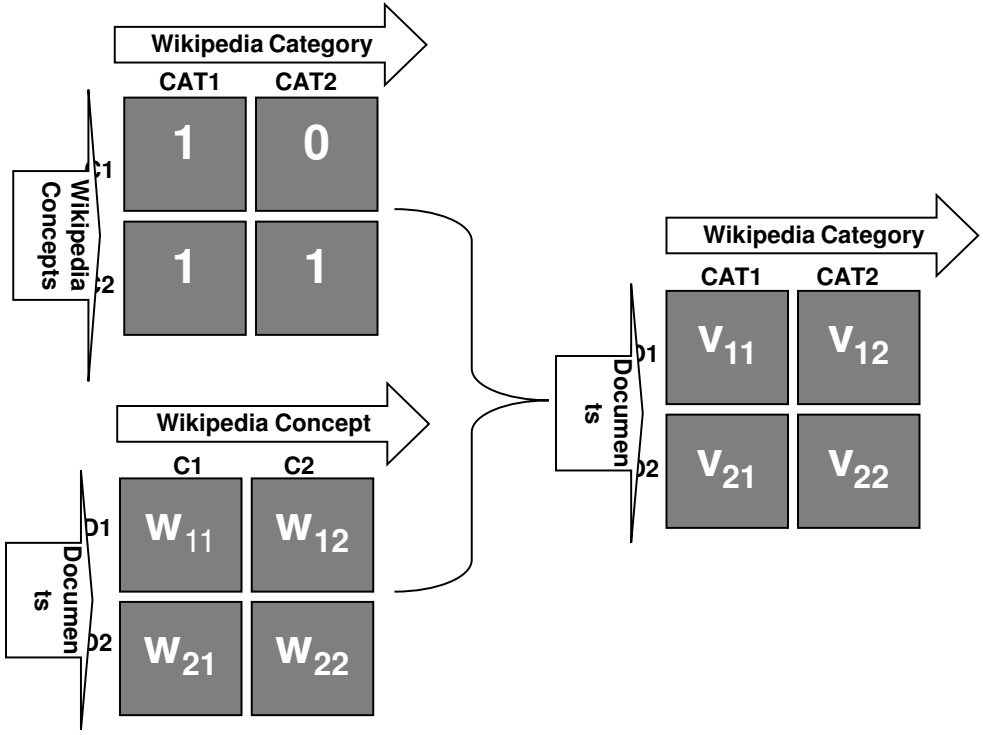
$$r_k^{d_j} = \sum tfidf_{d_j}^{w_i} \, tfidf_{c_k}^{w_i}$$

$tfidf_{d_j}^{w_i}$: the TFIDF score of word w_i in document d_j ; representing the importance of w_i to d_j

$tfidf_{c_k}^{w_i}$: the TFIDF score of word w_i in concept c_k ; representing the importance of w_i to c_k

- For each document, only the top *M* concepts with highest relatedness score are selected. (For example, M is 200)

Category Mapping



Concept Mapping
(Exact Match or Relatedness Match)

The hierarchical categorization system of Wikipedia

document-concept Vector

Wiki concepts C

	c_1	c_2	…	c_M
d1	r_{11}	r_{12}		r_{1N}

Wiki Concept-Category Matrix

Wiki categories *CAT*

Wiki Concepts C	cat_1	cat_2	…	cat_k
c_1	1	0	…	0
c_2	0	0	…	1
…	…	…	…	…
c_N	0	1	…	0

document-category Vector

Wiki categories *CAT*

	cat_1	cat_2	…	cat_k
d1	v_{11}	v_{12}		v_{1k}

Category Mapping

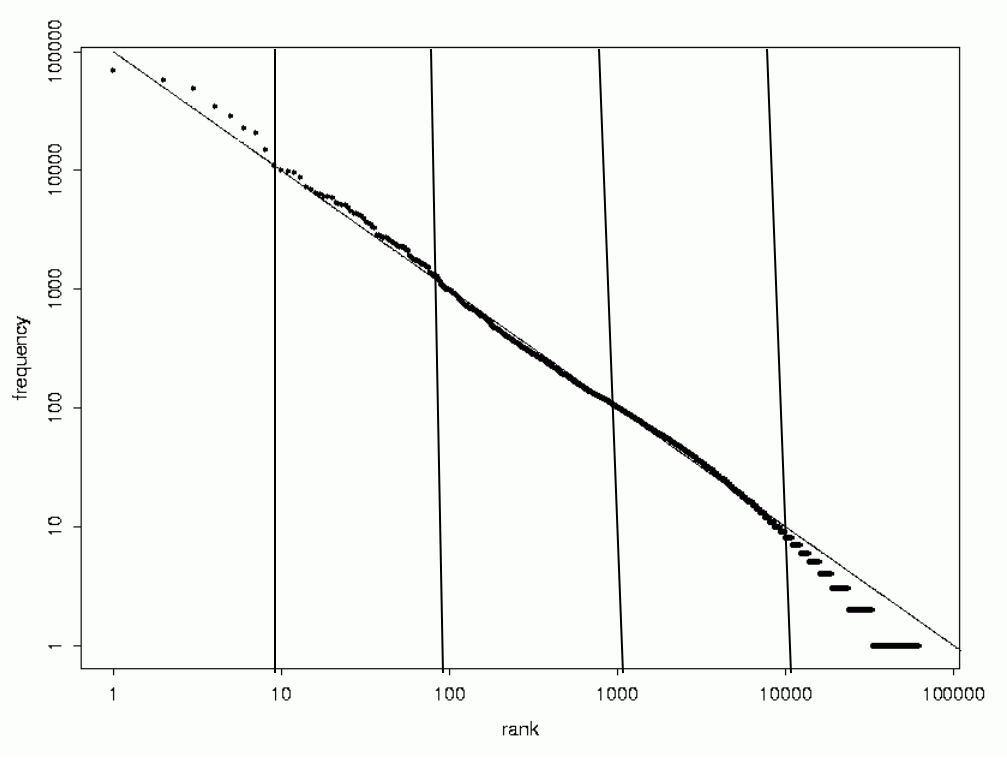
- Exact match:
  - A **document-category frequency matrix** is first derived from the **document-concept frequency matrix** by replacing each concept with its corresponding categories.
  - If a category is mapped to a document through more than one concept, the sum of the frequencies of these concepts is the category's frequency.
  - Based on the generated document-category frequency matrix, we further derive the **document-category TFIDF matrix**

Zipf's Law

- Look at the frequency of English words:
  - "the" is the most common, followed by "of", "to", etc.
  - claim: frequency of the n-th most common ~ 1/n (power law, $\alpha = 1$)
- General theme:
  - *rank* events by their *frequency of occurrence*
  - resulting distribution often is a power law!
- Other examples:
  - North America city sizes
  - personal income
  - file sizes
  - genus sizes (number of species)
- People seem to dither over exact form of these distributions (e.g. value of α), but not heavy tails

- Grows (but more slowly) with corpus size
- <u>Heap's Law</u>: The vocabulary size is
  - $m = kT^b$
- where $b \approx 0.5$, $k \approx 30–100$; T = # tokens
- m is decreased by case-folding, stemming
- Indexing all numbers could make it extremely large (so usually don't)
- Spelling errors contribute a fair bit of size
- <u>Zipf's Law:</u> The kth most frequent term has frequency proportional to $1/k$.

Zipf's law log-log plot



Rough analysis

- The *i* th most frequent term has frequency proportional to 1/*i*
- Let this frequency be *c/i.*
- Then $\sum c/i = c \log c$
- The number of terms is *c.*
- Also, the frequency of the most frequent term is *c.*
- *Remove the frequently occurring terms; how about Very Low Freq. terms?*
- We have grouped into log *c* groups; *c* words in each group.
- For dimensionality reduction, only a small number of groups are adequate.

Results on Cade2 Data

- # Documents: 1504 # Terms: 25893

Method	SVM	KNNC	DT
Unsupervised + MI (90 terms)	**79.29**	75.39	53.51
LSI (100 topics)	61.71	69.33	**75.19**
LDA (5 topics)	79.07	**86.32**	72.66
NMF (10 topics)	71.87	85.35	61.32
Random Projections	68.75	67.38	49.69

Results on NG2 Data

- # Documents: 1908 # Terms: 15566

Method	SVM	KNNC	DT
Unsupervised + MI (90 terms)	**97.77**	95.02	94.49
LSI (2 topics)	52.29	92.39	83.87
LDA (5 topics)	97.64	**97.64**	**95.28**
NMF (100 topics)	79.55	91.74	52.03
Random Projections	85.84	79.55	61.07

Results on R2 Data

- # Documents: 6215 # Terms: 14296

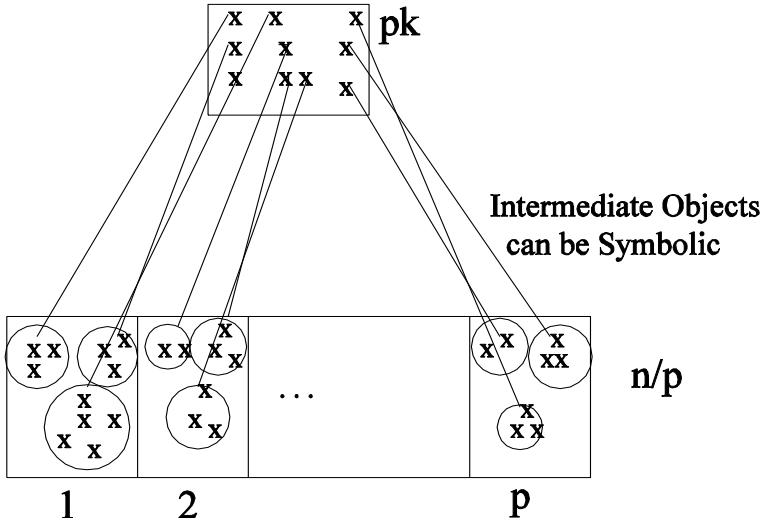
Method	SVM	KNNC	DT
Unsupervised + MI (90 terms)	**98.59**	**97.13**	93.98
LSI (100 topics)	72.51	95.39	**94.15**
LDA (100 topics)	97.86	94.32	92.07
NMF (5 topics)	94.15	71.72	59.61
Random Projections	94.88	88.92	81.90

Results on WebKB2 Data

- # Documents: 2739 # Terms: 7434

Method	SVM	KNNC	DT
Unsupervised + MI (90 terms)	**91.42**	75.82	**75.60**
LSI (100 topics)	69.56	**81.20**	69.89
LDA (100topics)	81.77	78.48	67.72
NMF (2 topics)	60.13	60.35	59.12
Random Projections	81.09	72.19	61.53

Method	Cade2	NG2	R2	WebKB2
Unsupervised + MI	2.7178	2.8964	5.5748	3.4527
LSI	17.6605	12.4350	18.6004	8.1939
LDA	252.192	447.332	593.821	517.159
NMF	1070	697	2765	523
Random Projections	4.2768	2.9449	8.1740	1.4567



1. Divide the set into p blocks, each of size n/p patterns.
2. Cluster patterns in each block into K clusters.
3. Collect the pK representatives and cluster them into K clusters.
4. Relabel the clusters obtained in step 2.

- What is the optimal number of blocks (P)?

# Patterns	# Blocks	# Clusters	# Distances (1-level)	# Distns. (2-level)
100	2	5	4950	2495
1000	10	5	499,500	50,525
10000	100	5	4,99,95,000	619,750

  - Select based on the memory available
  - Formally solve the associated optimization problem

- It is possible to show that the two-level algorithm gives a partition that is a constant factor approximation of the partition of one-level algorithm in terms of quality (fitness or accuracy).

(References: Murty and Krishna, Efficient Technique for Data Clustering, Pattern Recognition, 1980; Guha, Rastogi, Motwani, and O'Callaghan, Clustering Data Streams, FOCS 2000.)

What is MapReduce?

- Restricted parallel programming model meant for large clusters
  - User implements Map() and Reduce()
- Parallel computing framework
  - Libraries take care of EVERYTHING else
    - Parallelization
    - Fault Tolerance
    - Data Distribution
    - Load Balancing
- Useful model for many practical tasks

Map and Reduce

- Functions borrowed from functional programming languages (eg. Lisp)
- Map()
  - Process a key/value pair to generate intermediate key/value pairs
- Reduce()
  - Merge all intermediate values associated with the same key

Map

```
(define (square n)
(* n n))
 (map square '(1 2 3 4 5)) => (1 4 9 16 25)
```

Reduce

```
(define (plus a b)
   (+ a b))
(foldl plus 0 '(1 2 3 4 5)) => 15
   (5+...(2+(1+0)))
```

Example 1

- Suppose we want to count the number of occurrences of each word in a bunch of documents. For example,
- Doc 1: the weather is good
- Doc 2: today is good
- Doc 3: good weather is good.
- Map processes many pairs of the form

  <filename, contents>
- It generates many pairs of the form <word, 1> (Or <word, count> if you prefer...)

Word Index

- Map: emit <word, url> for each word in a given web document
- Reduce: combine all the <word, url> pairs associated with a given word into a single <word, url_list>. We can also sort the urls in the list, e.g., by word frequency

Example: Counting Words

- Map()
  - Input <filename, file text>
  - Parses file and emits <word, count> pairs
    - eg. <”hello”, 1>
- Reduce()
  - Sums all values for the same key and emits <word, TotalCount>
    - eg. <”hello”, (3 5 2 7)> => <”hello”, 17>

- Reduce processes all of the <word, count> pairs generated by the Map process for each word
  - Then it generates a single <word, total> for that word

```
void map (string url, string contents)
{
    foreach (word w in contents) {
        EmitIntermediate (word, 1);
    }
}
```

Example: Reduce

```
void reduce (string word, list values) {
    int total = 0;
    foreach (v in values) {
        total += (int) v;
    }
    Emit (word, total);
}
```

MapReduce

- Initially, the system selects M and R, the number of tasks to run map and reduce, respectively
- Then the input file(s) are split into M pieces, one for each map task
- One master task is in charge of submitting the M map and R reduce jobs to the server farm

How MapReduce Works

- User to do list:
  - indicate:
    - Input/output files
    - **M**: number of map tasks
    - **R**: number of reduce tasks
    - **W**: number of machines
  - Write *map* and *reduce* functions
  - Submit the job
- This requires no knowledge of parallel/distributed systems!!!

The Map Workers

- A map worker processes its portion of the input file and hands it to the user-defined map function
- As the user-defined map function generates <key,value> pairs, they are buffered in memory
- Periodically, the pairs are written out to disk into R intermediate datafiles (chosen by hash(key)%R)

The Map Workers

- When a map worker is finished, it will have produced R output files
- Then it notifies the master task that it is finished
- Note: the output files are kept in the worker machine
- If there are any more map tasks, the worker will be assigned another such task by the master

Map output

- Doc 1: the weather is good
- Doc 2: today is good
- Doc 3: good weather is good.
- Worker 1:
  - (the 1), (weather 1), (is 1), (good 1).
- Worker 2:
  - (today 1), (is 1), (good 1).
- Worker 3:
  - (good 1), (weather 1), (is 1), (good 1).

The Reduce Workers

- A reduce worker is notified when an intermediate data file is ready
- It requests the file from the map worker, and the file is transferred using a special protocol
- When all M intermediate files for a given reduce task are finished, the reduce task combines them into a <key, value_list> structure (by sorting on the keys first)

Reduce Input

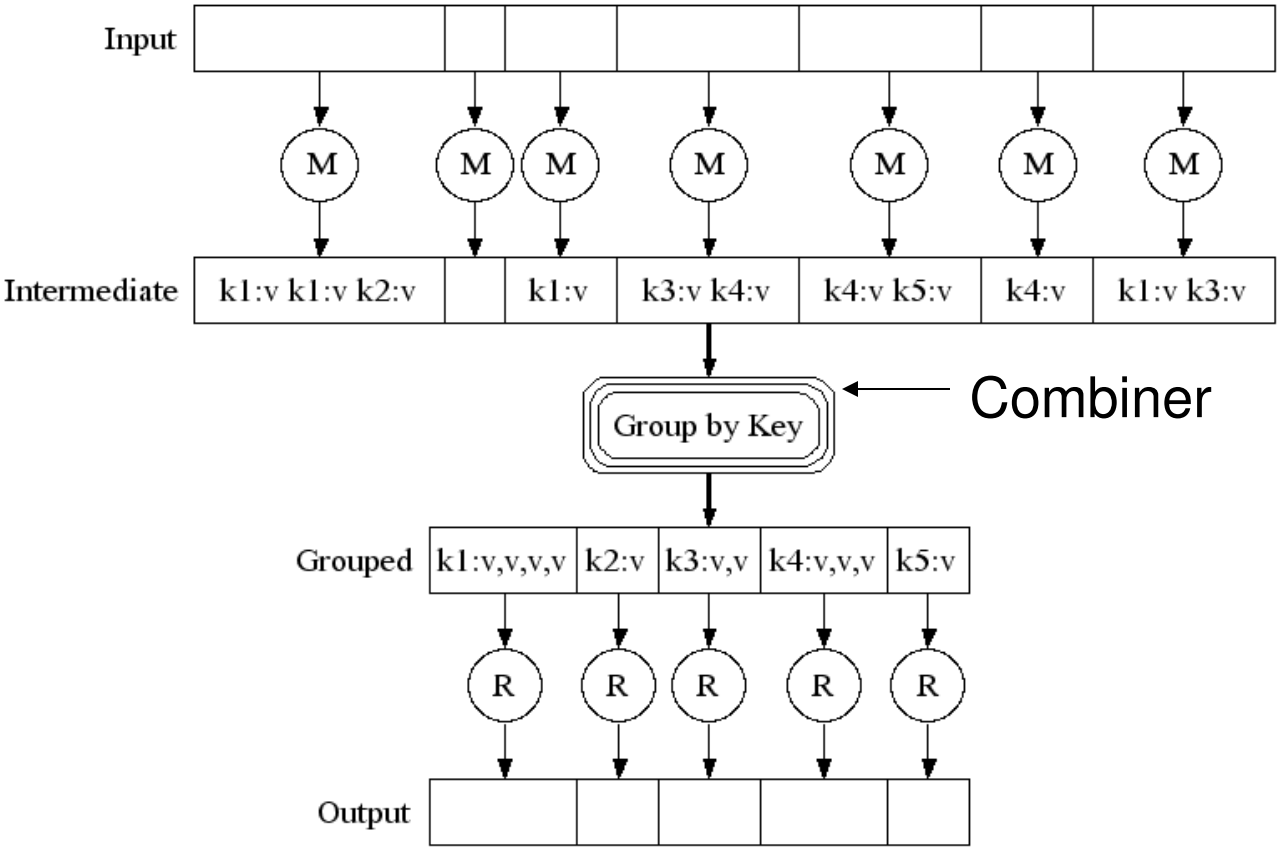
- Worker 1:
  - (the 1)
- Worker 2:
  - (is 1), (is 1), (is 1)
- Worker 3:
  - (weather 1), (weather 1)
- Worker 4:
  - (today 1)
- Worker 5:
  - (good 1), (good 1), (good 1), (good 1)

The Reduce Workers

- Once the <key, value_list> pairs are generated, the user-defined reduce function is called on each
- The workers append the output generated by the user-defined reduce function into a result file
- When the worker is finished, it notifies the master (which may give it another reduce task)

Reduce Output

- Worker 1:
  - (the 1)
- Worker 2:
  - (is 3)
- Worker 3:
  - (weather 2)
- Worker 4:
  - (today 1)
- Worker 5:
  - (good 4)


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Combiner
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

Other Applications: grep

- Map: emit a <file, line> record whenever the line matches the pattern
- Reduce: do nothing! Pass through the intermediate <file, line> records

Web Link Graph

- Map: emit record <source, target> for each link found in the source (going to target)
- Reduce: collect all the target links from a given source into a single <source, target_list> record

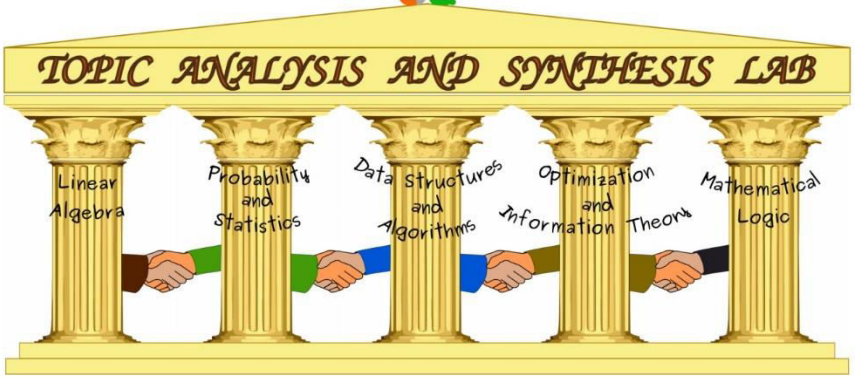
Reverse Link Graph

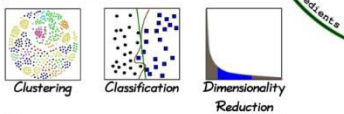
- Map: emit record <target, source> for each link found in the source (going to target)
- Reduce: collect all the sources to a given target into a single <target, source_list> record
  - Note: This is the same reduce as before...

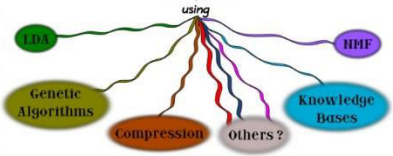
Research Directions

- Computing the eigenvalues and eigenvectors of a covariance matrix of size 100,000 x 100,000 may not be practical.
- Dimensionality reduction is important; can we ignore stop words and rare words, both?
- Relation between NMF and frequent patterns
- Topics are clusters; can we use efficient clustering schemes?
- Naïve Bayes classifier is popular. How far can we stretch it?
- How to improve the speed of GAs?
- Are heavy-tail or power-law distributions important in classification?
- What are efficient ways of summarizing documents? LDA + HMM?
- Wikipedia + LDA worked well on academic researchers data set. How do we make LDA more semantic?

TOPIC ANALYSIS AND SYNTHESIS LAB







References

- Manning, Raghavan, Schutze, *Information Retrieval*, 2009.

(some of the slides are taken from the lecture notes of Dr. P. Raghavan)

- Han and Kamber, *Data Mining*, Elsevier, 2008.
- Jing, Ng, and Huang, Knowledge-based vector space model for text clustering, *Knowl Inf Syst*, 25:35-55, 2010.
- Hu *et al*, Exploiting Wikipedia as External Knowledge for Document Clustering, KDD 2009.(some slides are from the presentation file)