# Informatics 225
# Computer Science 221

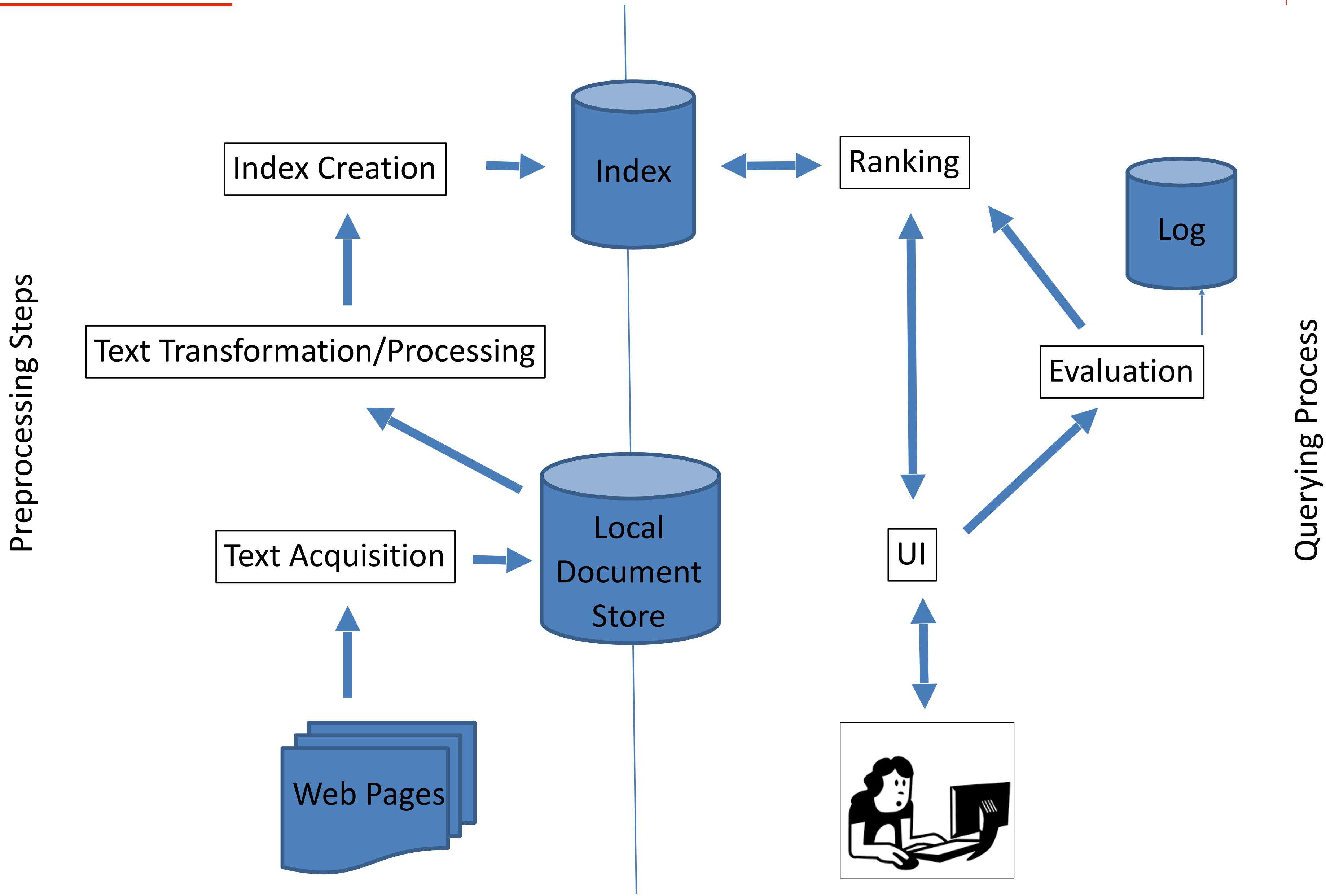
# Information Retrieval

Lecture 12



*These course materials borrow, with permission, from those of Prof. Cristina Videira Lopes, Prof. Alberto Krone-Martins, Addison Wesley 2008, Chris Manning, Pandu Nayak, Hinrich Schütze, Heike Adel, Sascha Rothe, Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. Powerpoint theme by Prof. André van der Hoek.*

# Architecture



# Detecting duplicates and removing noise

Information Retrieval

# Simhash

- Similarity comparisons using word-based representations of a document are more effective at finding near-duplicates than fingerprint techniques
  - Problem is efficiency

# Simhash

- Similarity comparisons using word-based representations of a document are more effective at finding near-duplicates than fingerprint techniques
  - Problem is efficiency

- Simhash was introduced by Charikar, 2002 to combine the advantages of the word-based similarity measures with the efficiency of fingerprints based on hashing

# Simhash

- Similarity comparisons using word-based representations of a document are more effective at finding near-duplicates than fingerprint techniques
  - Problem is efficiency

- Simhash was introduced by Charikar, 2002 to combine the advantages of the word-based similarity measures with the efficiency of fingerprints based on hashing

  - *Similarity of pages measured by the cosine correlation is proportional to the number of bits that are the same in simhash fingerprints*

# Simhash

1. Process the document into a set of features with associated weights. We will assume the simple case where the features are words weighted by their frequency.

# Simhash

1. Process the document into a set of features with associated weights. We will assume the simple case where the features are words weighted by their frequency.

2. Generate a hash value with $b$ bits (the desired size of the fingerprint) for each word. The hash value should be unique for each word.

# Simhash

1. Process the document into a set of features with associated weights. We will assume the simple case where the features are words weighted by their frequency.

2. Generate a hash value with $b$ bits (the desired size of the fingerprint) for each word. The hash value should be unique for each word.

3. In b-dimensional vector $V$, update the components of the vector by adding the weight for a word to every component for which the corresponding bit in the word's hash value is 1, and subtracting the weight if the value is 0.

# Simhash

1. Process the document into a set of features with associated weights. We will assume the simple case where the features are words weighted by their frequency.

2. Generate a hash value with $b$ bits (the desired size of the fingerprint) for each word. The hash value should be unique for each word.

3. In b-dimensional vector $V$, update the components of the vector by adding the weight for a word to every component for which the corresponding bit in the word's hash value is 1, and subtracting the weight if the value is 0.

4. After all words have been processed, generate a $b$-bit fingerprint by setting the $i$th bit to 1 if the $i$th component of $V$ is positive, or 0 otherwise.

# Simhash Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

# Simhash Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2  fish 2  include 1  found 1  environments 1  around 1  world 1
including 1  both 1  freshwater 1  salt 1  water 1  species 1

(b) Words with weights

# Simhash Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2  fish 2  include 1  found 1  environments 1  around 1  world 1
including 1  both 1  freshwater 1  salt 1  water 1  species 1

(b) Words with weights

| | | | | | |
|---|---|---|---|---|---|
| tropical | 01100001 | fish | 10101011 | include | 11100110 |
| found | 00011110 | environments | 00101101 | around | 10001011 |
| world | 00101010 | including | 11000000 | both | 10101110 |
| freshwater | 00111111 | salt | 10110101 | water | 00100101 |
| species | 11101110 | | | | |

(c) 8 bit hash values

# Simhash Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2  fish 2  include 1  found 1  environments 1  around 1  world 1
including 1  both 1  freshwater 1  salt 1  water 1  species 1

(b) Words with weights

| | | | | | |
|---|---|---|---|---|---|
| tropical | 0110000$\boxed{1}$ | fish | 1010101$\boxed{1}$ | include | 1110011$\boxed{0}$ |
| found | 0001111$\boxed{0}$ | environments | 0010110$\boxed{1}$ | around | 1000101$\boxed{1}$ |
| world | 0010101$\boxed{0}$ | including | 1100000$\boxed{0}$ | both | 1010111$\boxed{0}$ |
| freshwater | 0011111$\boxed{1}$ | salt | 1011010$\boxed{1}$ | water | 0010010$\boxed{1}$ |
| species | 1101111$\boxed{0}$ | | | | |

(c) 8 bit hash values

**3 = (2 -1 - 1 + 1 -1) + (2 + 1 -1 +1) + (-1 +1 -1 +1)**

1 -5 9 -9 3 1 3 $\boxed{3}$

(d) Vector $V$ formed by summing weights

# Simhash Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2  fish 2  include 1  found 1  environments 1  around 1  world 1
including 1  both 1  freshwater 1  salt 1  water 1  species 1

(b) Words with weights

| | | | | | |
|---|---|---|---|---|---|
| tropical | 01100001 | fish | 10101011 | include | 11100110 |
| found | 00011110 | environments | 00101101 | around | 10001011 |
| world | 00101010 | including | 11000000 | both | 10101110 |
| freshwater | 00111111 | salt | 10110101 | water | 00100101 |
| species | 11101110 | | | | |

(c) 8 bit hash values

1  -5  9  -9  3  1  3  3

(d) Vector $V$ formed by summing weights

1  0  1  0  1  1  1  1

(e) 8-bit fingerprint formed from $V$

# Simhash

1. Process the document into a set of features with associated weights. We will assume the simple case where the features are words weighted by their frequency.

2. Generate a hash value with $b$ bits (the desired size of the fingerprint) for each word. The hash value should be unique for each word.

3. In b-dimensional vector $V$, update the components of the vector by adding the weight for a word to every component for which the corresponding bit in the word's hash value is 1, and subtracting the weight if the value is 0.

4. After all words have been processed, generate a $b$-bit fingerprint by setting the $i$th bit to 1 if the $i$th component of $V$ is positive, or 0 otherwise.

# Simhash

- So, how similar are two text files A and B?
  - Compute the simhashes $H_A$ and $H_B$ for each text.

  - The similarity is simply the fraction of the bits that are the same over all n bits of the representation.

$$S_{A,B} = \frac{\sum_n (\mathcal{H}_A \equiv_{\text{bitwise}} \mathcal{H}_B)}{n}$$

# Simhash

- So, how similar are two text files A and B?
  - Compute the simhashes $H_A$ and $H_B$ for each text.
  - The similarity is simply the fraction of the bits that are the same over all n bits of the representation.

$$S_{A,B} = \frac{\sum_n (\mathcal{H}_A \equiv_{\text{bitwise}} \mathcal{H}_B)}{n}$$

- Are text A and text B near duplicates?
  - Define a threshold level τ.
  - If similarity is greater or equal than the threshold, they are near duplicates **under your definition of threshold.**

$$S_{A,B} \geq \tau \implies \text{A \& B are near duplicates}$$

# Removing noise

Information Retrieval

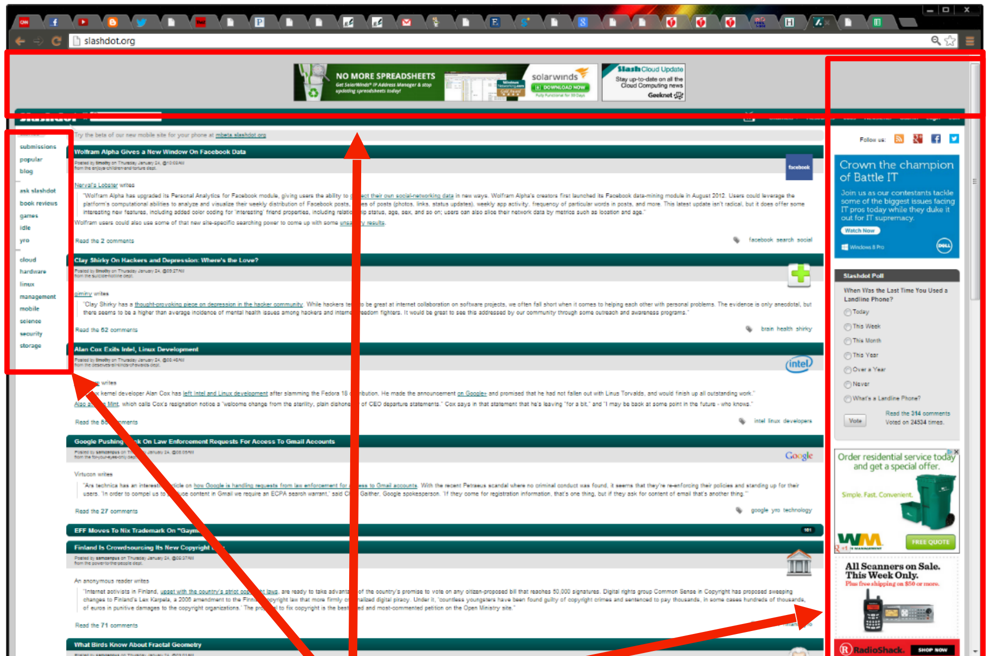
# Removing Noise

- Many web pages contain text, links, and pictures that are not directly related to the main content of the page

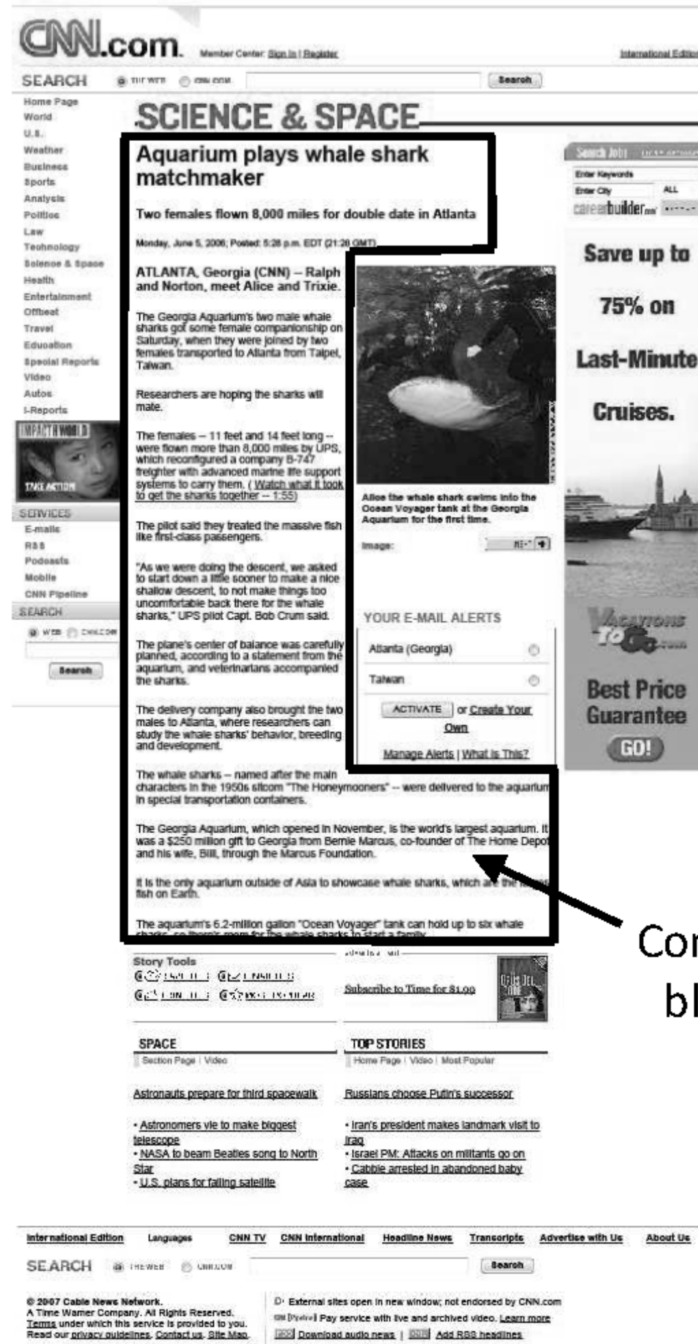
- This additional material is mostly *noise* that could negatively affect the ranking of the page

- Techniques have been developed to detect the content blocks in a web page
  - Non-content material is either ignored or reduced in importance in the indexing process

# Noise Example

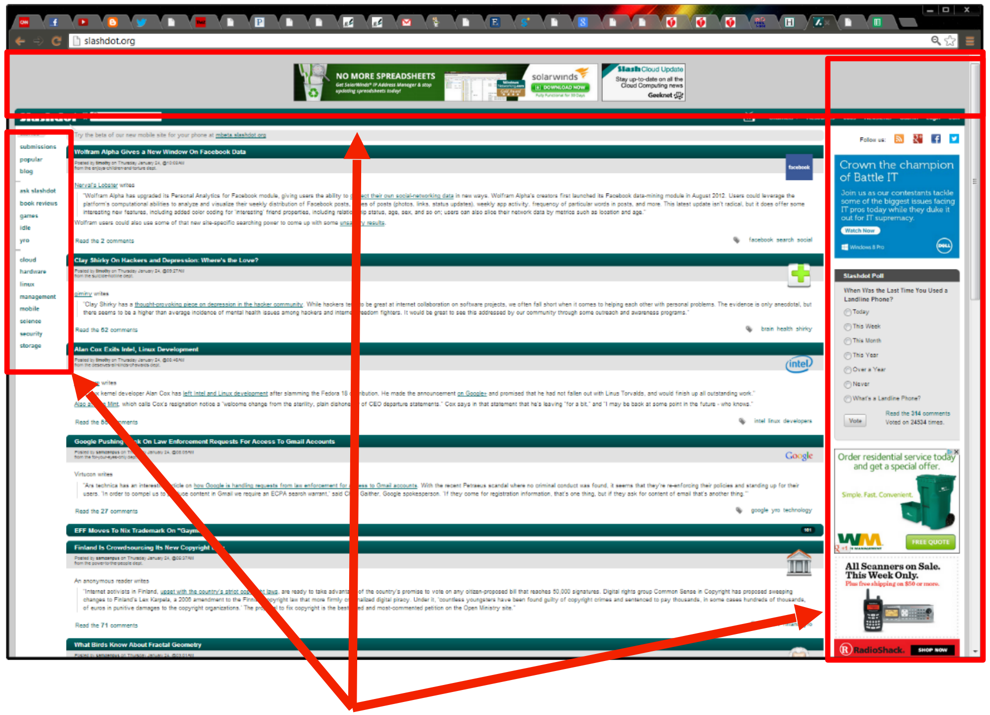



Noise blocks

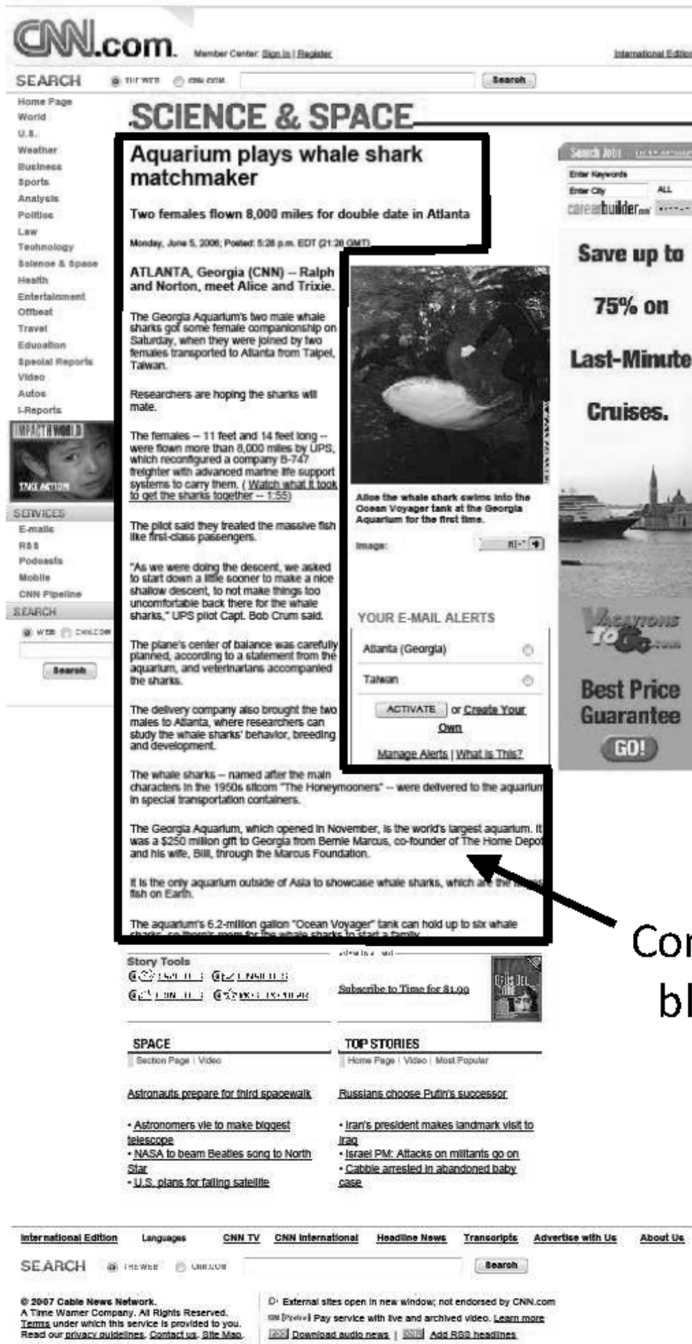



Content block

# Noise Example



Noise blocks

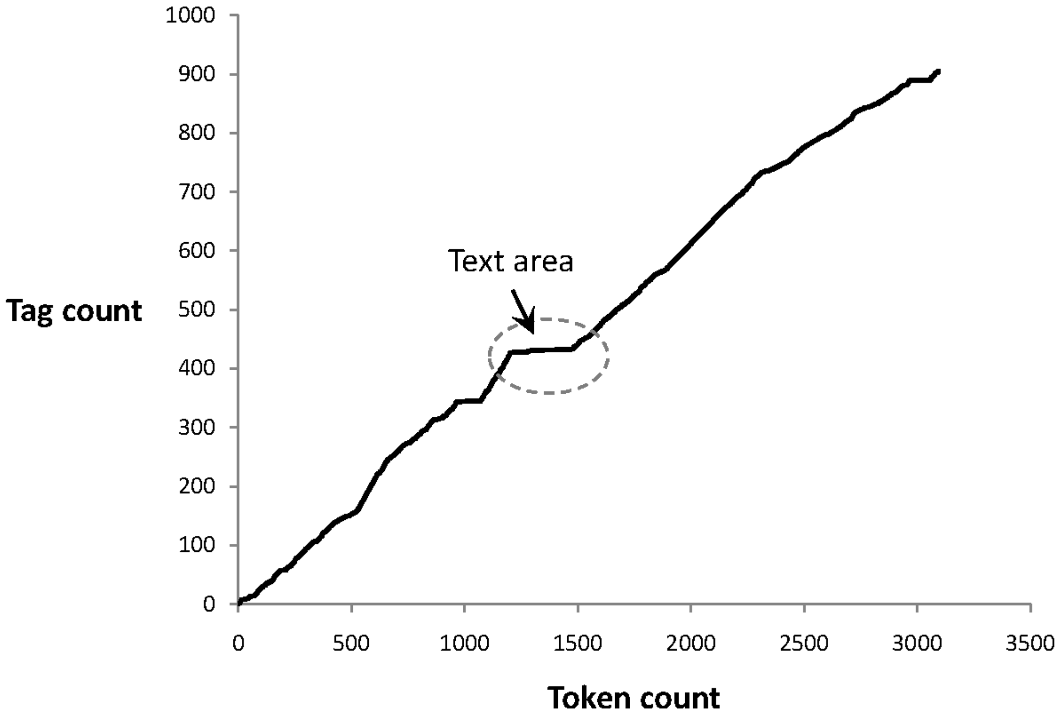


Content block

So how can you detect content?

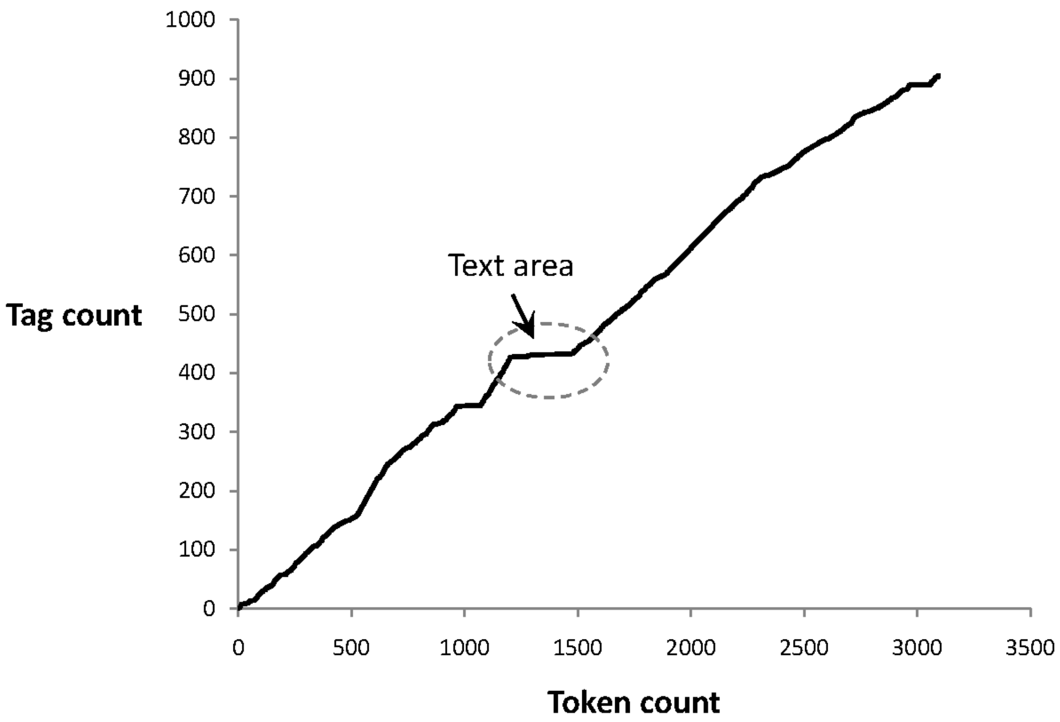
# Finding Content Blocks

- Cumulative distribution of tags
  - Document slope curve (e.g. Finn, Kushmerick & Smyth, 2001)



# Finding Content Blocks

- Cumulative distribution of tags




  – Main text content of the page corresponds to the "plateau" in the middle of the distribution

# Finding Content Blocks

- Represent a **web page** as a **sequence** of **bits**, where $b_n = 1$ indicates that the ***n*th token is a tag**

# Finding Content Blocks

- Represent a **web page** as a **sequence** of **bits**, where $b_n = 1$ indicates that the ***n*th token is a tag**
- Optimization problem where we **find values of *i* and *j* to maximize** both the number of **tags below *i* and above *j* and** the number of **non-tag tokens between *i* and *j***

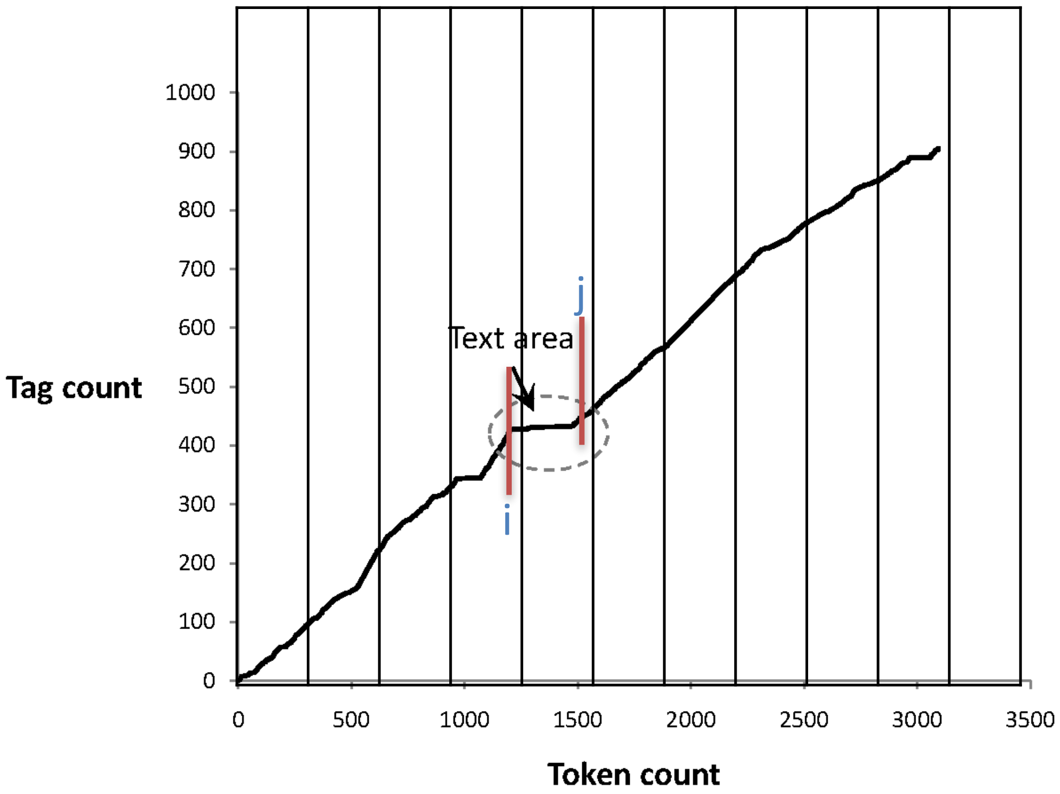
# Finding Content Blocks

- Represent a **web page** as a **sequence** of **bits**, where $b_n = 1$ indicates that the ***n*th token is a tag**
- Optimization problem where we **find values of *i* and *j* to maximize** both the number of **tags below *i* and above *j* and** the number of **non-tag tokens between *i* and *j***
- i.e., find i and j that maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1 - b_n) + \sum_{n=j+1}^{N-1} b_n$$

# Finding Content Blocks : Alternative

- Cumulative distribution of tags in the example web page

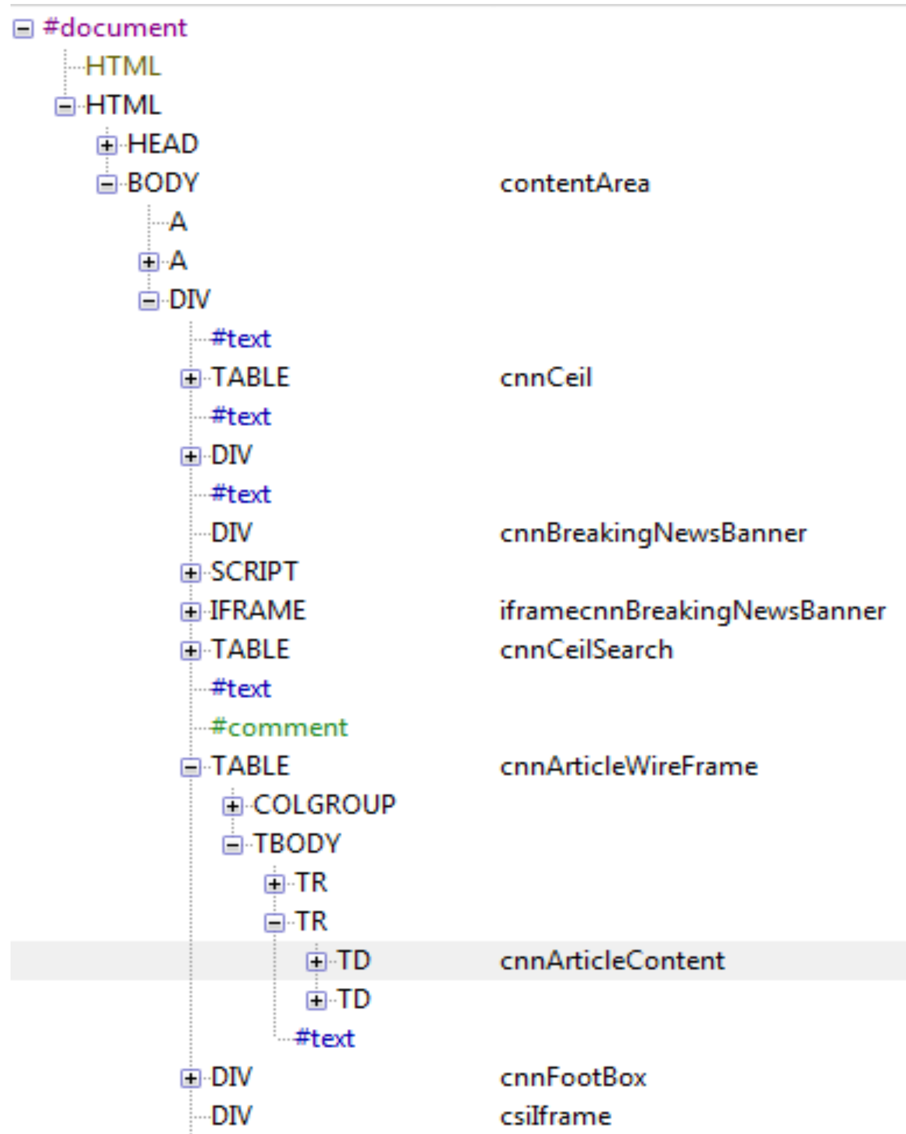



  - Determine the slope of the lines inside slices and iterate, reducing the slice size until you find zero slope windows.

# Finding Content Blocks

- Other approaches use Document Object Model (DOM) structure and visual (layout) features

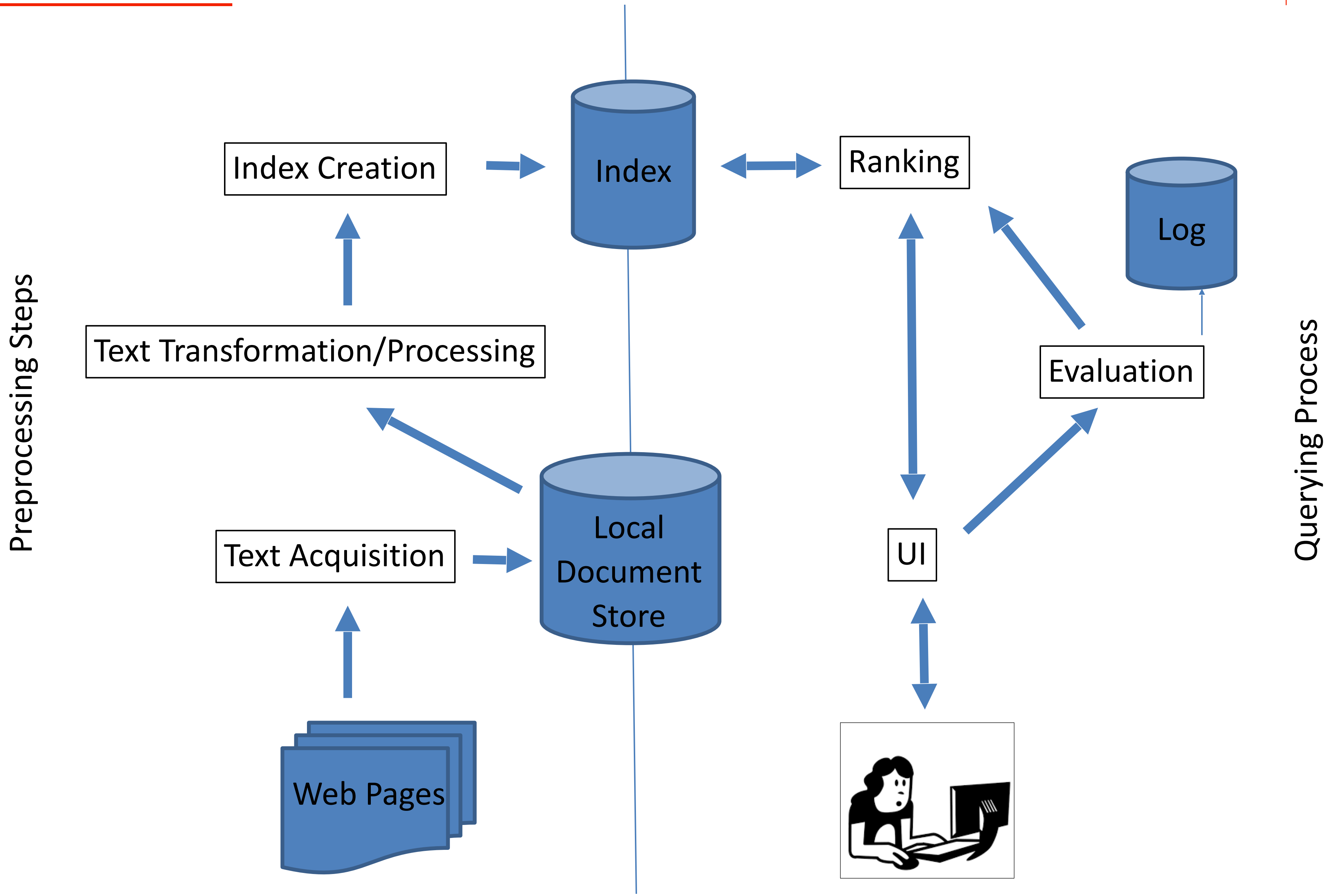
- HTML parser:
  - HTML -> Document Object Model representation
  - Tree like structure



# Text processing : statistics

Information Retrieval

# Architecture



# Processing Text

- Essential step to convert documents to *index terms*

- Why?
  - In the context of web search matching the exact string of characters typed by the user is too restrictive
    - i.e., it doesn't work very well in terms of effectiveness (e.g. case sensitive searches *may or may not* be what you want)
  - Not all words are of equal value in a search
  - Sometimes not clear where words begin and end
    - Not even clear what a word is in some languages
      - e.g., Chinese, Korean

# Text Statistics

- Huge variety of words used in text
- But many statistical characteristics of word occurrences are predictable!
  - e.g., distribution of word counts

# Text Statistics

- Huge variety of words used in text
- But many statistical characteristics of word occurrences are predictable!
  - e.g., distribution of word counts
- Retrieval models and ranking algorithms depend **heavily** on statistical properties of words
  - e.g., *important words for retrieval tasks are words that occur often in documents but are not frequent in the corpus*

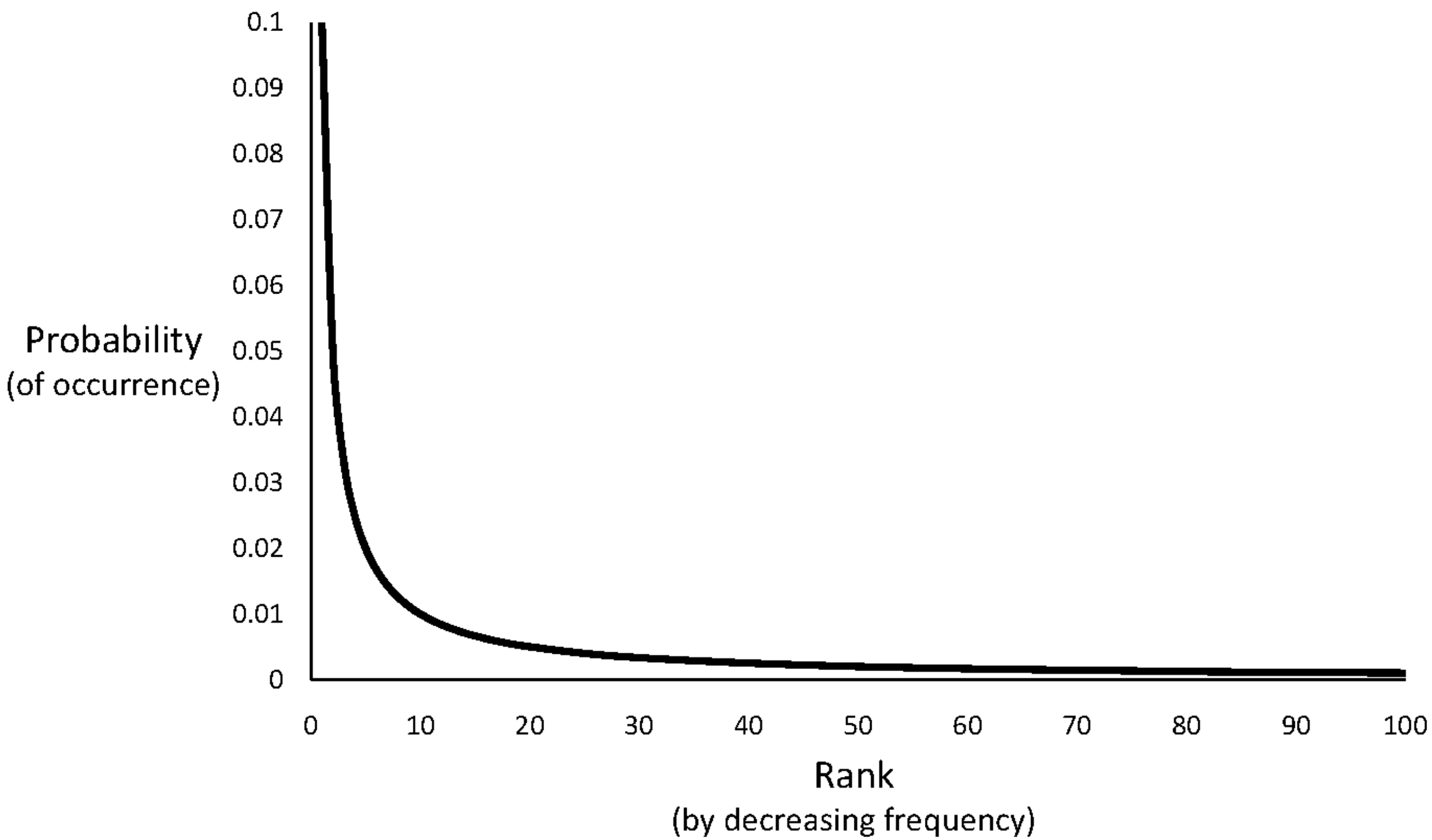
# Zipf’s Law

- Distribution of word frequencies is very *skewed*
  - a few words occur very often, many words hardly ever occur
  - e.g., two most common words (“the”, “of”) make up about 10% of all word occurrences in text documents!

# Zipf's Law

- Distribution of word frequencies is very *skewed*
  - a few words occur very often, many words hardly ever occur
  - e.g., two most common words ("the", "of") make up about 10% of all word occurrences in text documents!
- Zipf's "law":
  - "The frequency of the r-th most common word is inversely proportional to r"
  - Or: the rank (*r*) of a word times its frequency (*f*) is ~ a constant (*k*)
    - assuming words are ranked in order of decreasing frequency
  - i.e., $r.f \approx$ k or $r.P_r \approx c$, where $P_r$ is probability of word occurrence and $c \approx 0.1$ for English

# Zipf’s Law for English



# News Collection (Associated Press 89) Statistics

| | |
|---|---|
| Total documents | 84,678 |
| Total word occurrences | 39,749,179 |
| Vocabulary size (unique words) | 198,763 |
| Words occurring > 1000 times | 4,169 |
| Words occurring once | 70,064 |

Biggest variations :

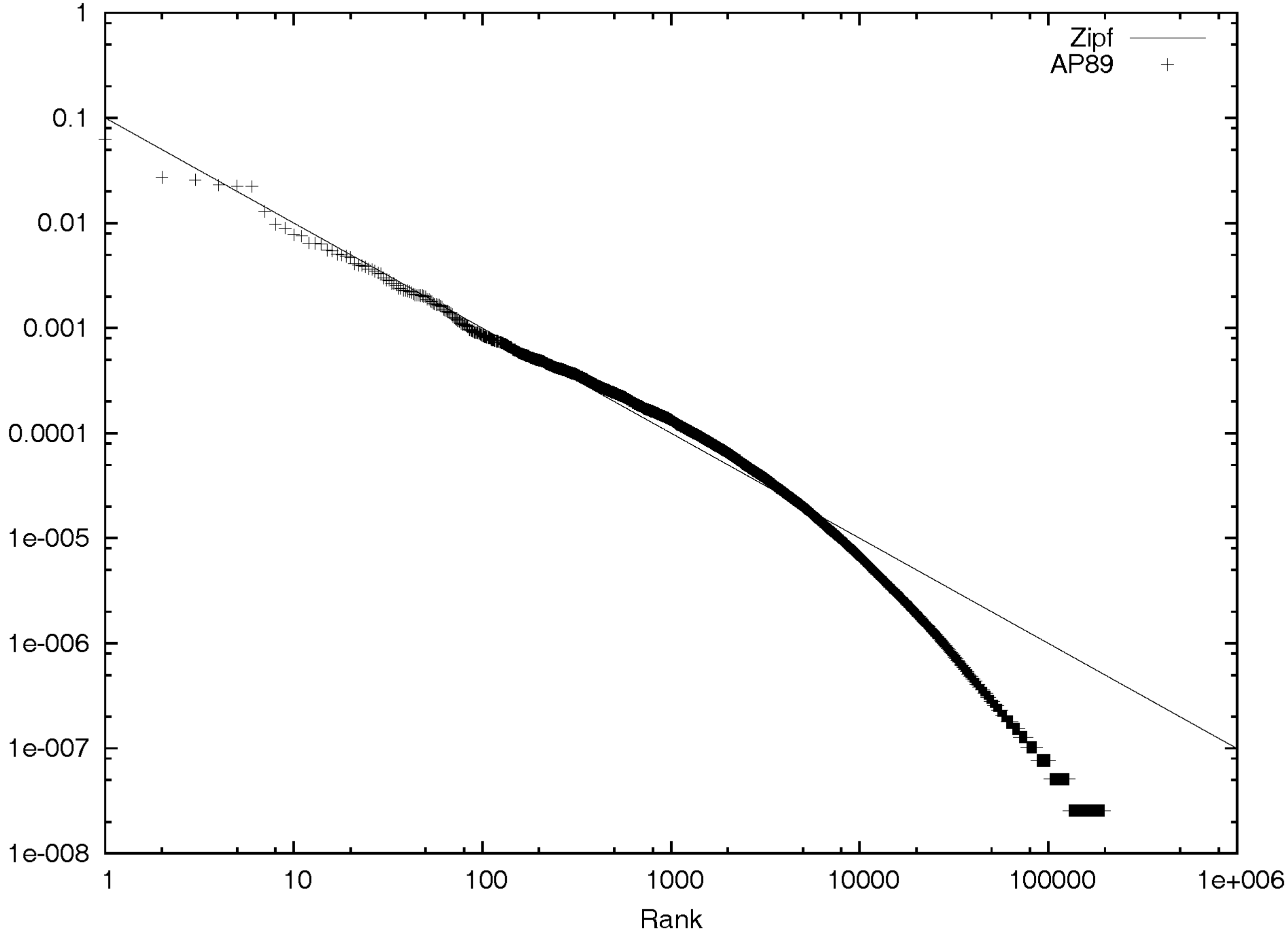
Highest frequency words

but if you keep digging…

| *Word* | *Freq.* | $r$ | $P_r(\%)$ | $r.P_r$ | *Word* | *Freq* | $r$ | $P_r(\%)$ | $r.P_r$ |
|---|---|---|---|---|---|---|---|---|---|
| the | 2,420,778 | 1 | 6.49 | 0.065 | has | 136,007 | 26 | 0.37 | 0.095 |
| of | 1,045,733 | 2 | 2.80 | 0.056 | are | 130,322 | 27 | 0.35 | 0.094 |
| to | 968,882 | 3 | 2.60 | 0.078 | not | 127,493 | 28 | 0.34 | 0.096 |
| a | 892,429 | 4 | 2.39 | 0.096 | who | 116,364 | 29 | 0.31 | 0.090 |
| and | 865,644 | 5 | 2.32 | 0.120 | they | 111,024 | 30 | 0.30 | 0.089 |
| in | 847,825 | 6 | 2.27 | 0.140 | its | 111,021 | 31 | 0.30 | 0.092 |
| said | 504,593 | 7 | 1.35 | 0.095 | had | 103,943 | 32 | 0.28 | 0.089 |
| for | 363,865 | 8 | 0.98 | 0.078 | will | 102,949 | 33 | 0.28 | 0.091 |
| that | 347,072 | 9 | 0.93 | 0.084 | would | 99,503 | 34 | 0.27 | 0.091 |
| was | 293,027 | 10 | 0.79 | 0.079 | about | 92,983 | 35 | 0.25 | 0.087 |
| on | 291,947 | 11 | 0.78 | 0.086 | i | 92,005 | 36 | 0.25 | 0.089 |
| he | 250,919 | 12 | 0.67 | 0.081 | been | 88,786 | 37 | 0.24 | 0.088 |
| is | 245,843 | 13 | 0.65 | 0.086 | this | 87,286 | 38 | 0.23 | 0.089 |
| with | 223,846 | 14 | 0.60 | 0.084 | their | 84,638 | 39 | 0.23 | 0.089 |
| at | 210,064 | 15 | 0.56 | 0.085 | new | 83,449 | 40 | 0.22 | 0.090 |
| by | 209,586 | 16 | 0.56 | 0.090 | or | 81,796 | 41 | 0.22 | 0.090 |
| it | 195,621 | 17 | 0.52 | 0.089 | which | 80,385 | 42 | 0.22 | 0.091 |
| from | 189,451 | 18 | 0.51 | 0.091 | we | 80,245 | 43 | 0.22 | 0.093 |
| as | 181,714 | 19 | 0.49 | 0.093 | more | 76,388 | 44 | 0.21 | 0.090 |
| be | 157,300 | 20 | 0.42 | 0.084 | after | 75,165 | 45 | 0.20 | 0.091 |
| were | 153,913 | 21 | 0.41 | 0.087 | us | 72,045 | 46 | 0.19 | 0.089 |
| an | 152,576 | 22 | 0.41 | 0.090 | percent | 71,956 | 47 | 0.19 | 0.091 |
| have | 149,749 | 23 | 0.40 | 0.092 | up | 71,082 | 48 | 0.19 | 0.092 |
| his | 142,285 | 24 | 0.38 | 0.092 | one | 70,266 | 49 | 0.19 | 0.092 |
| but | 140,880 | 25 | 0.38 | 0.094 | people | 68,988 | 50 | 0.19 | 0.093 |

AP89 most frequent words

# Zipf's Law for AP89




- Note problems at high and low frequencies

- Words that occur only once : *Hapax Legomena* (ἅπαξ λεγόμενον).