Information Retrieval

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur,
India

14th June, 2017
ACM Summer School on NLP and ML

Outline

1. Introduction
   - Motivation
2. Document Retrieval
   - Inverted Index
3. Tokenization
   - Stopwords
   - Token Normalization
4. Scoring
   - Zone
   - Term Frequency
   - Inverse Document Frequency
   - Tf-idf
5. Document Vector
   - Vector Model
   - Document Similarity
6. Scalability
   - Skip List
   - Inexact Retrieval
7. System
8. Conclusions

Outline

1. Introduction
   - Motivation
2. Document Retrieval
   - Inverted Index
3. Tokenization
   - Stopwords
   - Token Normalization
4. Scoring
   - Zone
   - Term Frequency
   - Inverse Document Frequency
   - Tf-idf
5. Document Vector
   - Vector Model
   - Document Similarity
6. Scalability
   - Skip List
   - Inexact Retrieval
7. System
8. Conclusions

Definition

- What is information retrieval?

Definition

- What is information retrieval?

Information Retrieval

Retrieval (finding) of information (e.g., documents) that is mostly unstructured (e.g., text) and is relevant to a particular need (query) from a large collection

Definition

- What is information retrieval?

Information Retrieval

Retrieval (finding) of information (e.g., documents) that is mostly unstructured (e.g., text) and is relevant to a particular need (query) from a large collection

- Started with documents
- Has now extended to music, images, graphs

Outline

1. Introduction
   - Motivation
2. Document Retrieval
   - Inverted Index
3. Tokenization
   - Stopwords
   - Token Normalization
4. Scoring
   - Zone
   - Term Frequency
   - Inverse Document Frequency
   - Tf-idf
5. Document Vector
   - Vector Model
   - Document Similarity
6. Scalability
   - Skip List
   - Inexact Retrieval
7. System
8. Conclusions

Document Retrieval Task

- Assume a set $\mathcal{D} = \{D_1, \dots, D_m\}$ of *documents*
- Each D_i is composed of a number of *terms* $D_i = \{t_{i_j}\}$

Document Retrieval Task

- Assume a set $\mathcal{D} = \{D_1, \dots, D_m\}$ of *documents*
- Each D_i is composed of a number of *terms* $D_i = \{t_{i_j}\}$
- The total collection of terms is called the dictionary or lexicon or vocabulary

Document Retrieval Task

- Assume a set $\mathcal{D} = \{D_1, \ldots, D_m\}$ of *documents*
- Each D_i is composed of a number of *terms* $D_i = \{t_{i_j}\}$
- The total collection of terms is called the dictionary or lexicon or vocabulary
- *Information retrieval task*
  - Given a set of terms from the vocabulary, find the documents that contain them

Document Retrieval Task

- Assume a set $\mathcal{D} = \{D_1, \dots, D_m\}$ of *documents*
- Each D_i is composed of a number of *terms* $D_i = \{t_{i_j}\}$
- The total collection of terms is called the dictionary or lexicon or vocabulary
- *Information retrieval task*
    - Given a set of terms from the vocabulary, find the documents that contain them
    - Sense is generally AND; may sometimes involve OR and NOT

Document Retrieval Task

- Assume a set $\mathcal{D} = \{D_1, \dots, D_m\}$ of *documents*
- Each D_i is composed of a number of *terms* $D_i = \{t_{i_j}\}$
- The total collection of terms is called the dictionary or lexicon or vocabulary
- *Information retrieval task*
  - Given a set of terms from the vocabulary, find the documents that contain them
  - Sense is generally AND; may sometimes involve OR and NOT
- Unix utility `grep` solves it nicely

Document Retrieval Task

- Assume a set $\mathcal{D} = \{D_1, \dots, D_m\}$ of *documents*
- Each D_i is composed of a number of *terms* $D_i = \{t_{i_j}\}$
- The total collection of terms is called the dictionary or lexicon or vocabulary
- *Information retrieval task*
    - Given a set of terms from the vocabulary, find the documents that contain them
    - Sense is generally AND; may sometimes involve OR and NOT
- Unix utility `grep` solves it nicely
- Boolean bit matrix for each document and each term
- Boolean algebra on the bit vectors of the queried terms

Document Retrieval Task

- Assume a set $\mathcal{D} = \{D_1, \dots, D_m\}$ of *documents*
- Each D_i is composed of a number of *terms* $D_i = \{t_{i_j}\}$
- The total collection of terms is called the dictionary or lexicon or vocabulary
- *Information retrieval task*
    - Given a set of terms from the vocabulary, find the documents that contain them
    - Sense is generally AND; may sometimes involve OR and NOT
- Unix utility `grep` solves it nicely
- Boolean bit matrix for each document and each term
- Boolean algebra on the bit vectors of the queried terms
- *Not* scalable (remember "large" collections)

Inverted Index

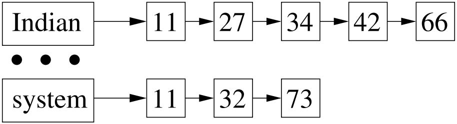
- "Invert" the sense

Inverted Index

- “Invert” the sense – let terms be composed of documents

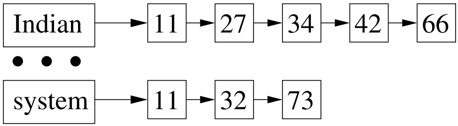
Inverted Index

- "Invert" the sense – let terms be composed of documents
- Each term contains a list of documents (called postings) that it appears in
  - May include the number of times it appears in a document, called term frequency
- This list of documents is called a postings list
  - Its size, called document frequency, is the number of documents a term appears
- The set of lists for all the terms is called postings
  - Its size is size of the vocabulary



Inverted Index

- "Invert" the sense – let terms be composed of documents
- Each term contains a list of documents (called postings) that it appears in
  - May include the number of times it appears in a document, called term frequency
- This list of documents is called a postings list
  - Its size, called document frequency, is the number of documents a term appears
- The set of lists for all the terms is called postings
  - Its size is size of the vocabulary



- Postings list is maintained as a linked list

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur
- Get corresponding postings lists
- *Intersect* them

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur
- Get corresponding postings lists
- *Intersect* them
- For two lists of size x and y, time required is $O(x.y)$

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur
- Get corresponding postings lists
- *Intersect* them
- For two lists of size x and y, time required is $O(x.y)$
- If lists are sorted by document ids
  - Merging as in mergesort
  - Two pointers to walk along the lists
  - Time complexity is $O(x + y)$

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur
- Get corresponding postings lists
- *Intersect* them
- For two lists of size x and y, time required is $O(x.y)$
- If lists are sorted by document ids
  - Merging as in mergesort
  - Two pointers to walk along the lists
  - Time complexity is $O(x + y)$
- For multiple query terms: “ancient”, “Indian” and “system”

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur
- Get corresponding postings lists
- *Intersect* them
- For two lists of size x and y, time required is $O(x.y)$
- If lists are sorted by document ids
  - Merging as in mergesort
  - Two pointers to walk along the lists
  - Time complexity is $O(x + y)$
- For multiple query terms: “ancient”, “Indian” and “system”
  - Start with terms having *smallest* postings lists
  - Progressively merge
  - Size of final answer set is never more than any intermediate step

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur
- Get corresponding postings lists
- *Intersect* them
- For two lists of size x and y, time required is $O(x.y)$
- If lists are sorted by document ids
  - Merging as in mergesort
  - Two pointers to walk along the lists
  - Time complexity is $O(x + y)$
- For multiple query terms: “ancient”, “Indian” and “system”
  - Start with terms having *smallest* postings lists
  - Progressively merge
  - Size of final answer set is never more than any intermediate step
- OR is more time consuming

Querying using Inverted Index

- Find documents where terms “Indian” and “system” occur
- Get corresponding postings lists
- *Intersect* them
- For two lists of size x and y, time required is $O(x.y)$
- If lists are sorted by document ids
  - Merging as in mergesort
  - Two pointers to walk along the lists
  - Time complexity is $O(x + y)$
- For multiple query terms: “ancient”, “Indian” and “system”
  - Start with terms having *smallest* postings lists
  - Progressively merge
  - Size of final answer set is never more than any intermediate step
- OR is more time consuming
- Query with only NOT is impractical for large collections
  - All right with AND: “system” AND NOT “Indian”

Outline

1. Introduction
   - Motivation
2. Document Retrieval
   - Inverted Index
3. Tokenization
   - Stopwords
   - Token Normalization
4. Scoring
   - Zone
   - Term Frequency
   - Inverse Document Frequency
   - Tf-idf
5. Document Vector
   - Vector Model
   - Document Similarity
6. Scalability
   - Skip List
   - Inexact Retrieval
7. System
8. Conclusions

Tokenization

- Terms are useful semantic units that need to be indexed for searching
- Tokenization is the process of breaking the text into *terms*

Tokenization

- Terms are useful semantic units that need to be indexed for searching
- Tokenization is the process of breaking the text into *terms*
  - Sounds obvious in English with whitespaces
  - What about Indian languages with *sandhi* and *samash*?
  - What about CJK languages with no space between words (or sometimes even paragraphs)?
  - Non-trivial for even some European languages
    - "computerlinguistik" (German, computational linguistics)
    - "l'ensemble" (French, the collection)
- Word segmentation

Tokenization

- Terms are useful semantic units that need to be indexed for searching
- Tokenization is the process of breaking the text into *terms*
  - Sounds obvious in English with whitespaces
  - What about Indian languages with *sandhi* and *samash*?
  - What about CJK languages with no space between words (or sometimes even paragraphs)?
  - Non-trivial for even some European languages
    - “computerlinguistik” (German, computational linguistics)
    - “l’ensemble” (French, the collection)
- Word segmentation

मूर्तमहेश्वरमुज्ज्वलभास्करमिष्टममरनरवन्दम्

Tokenization

- Terms are useful semantic units that need to be indexed for searching
- Tokenization is the process of breaking the text into *terms*
  - Sounds obvious in English with whitespaces
  - What about Indian languages with *sandhi* and *samash*?
  - What about CJK languages with no space between words (or sometimes even paragraphs)?
  - Non-trivial for even some European languages
    - “computerlinguistik” (German, computational linguistics)
    - “l’ensemble” (French, the collection)
- Word segmentation

मूर्तमहेश्वरमुज्ज्वलभास्करमिष्टममरनरवन्दम्

मूर्त + महेश्वरम् + उज्ज्वल + भास्करम् + इष्टम् + अमर + नर + वन्दम्

Problems in English

- Idioms

Problems in English

- Idioms
    - “hot potato”, “couch potato”

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations
  - “can’t”, “O’Neille”

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations
  - “can’t”, “O’Neille”
- Hyphenation

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations
  - “can’t”, “O’Neille”
- Hyphenation
  - “co-education”

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations
  - “can’t”, “O’Neille”
- Hyphenation
  - “co-education”
- White space

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations
  - “can’t”, “O’Neille”
- Hyphenation
  - “co-education”
- White space
  - “New Delhi”

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations
  - “can’t”, “O’Neille”
- Hyphenation
  - “co-education”
- White space
  - “New Delhi”
- Combinations

Problems in English

- Idioms
  - “hot potato”, “couch potato”
- Word markers or punctuations
  - “can’t”, “O’Neille”
- Hyphenation
  - “co-education”
- White space
  - “New Delhi”
- Combinations
  - “isn’t New Delhi-Uttar Pradesh a good example?”

Stopwords

- Some terms add little *semantic* content to search
  - “to”, “and”, “the”
- These are called stopwords

Stopwords

- Some terms add little *semantic* content to search
  - “to”, “and”, “the”
- These are called stopwords
- Terms that have a very high *frequency* are candidates
- Generally, manually polished by linguistics

Stopwords

- Some terms add little *semantic* content to search
  - “to”, “and”, “the”
- These are called stopwords
- Terms that have a very high *frequency* are candidates
- Generally, manually polished by linguistics
- *Variety* in the corpus of documents is very important

Stopwords

- Some terms add little *semantic* content to search
  - “to”, “and”, “the”
- These are called stopwords
- Terms that have a very high *frequency* are candidates
- Generally, manually polished by linguistics
- *Variety* in the corpus of documents is very important
- If only newspaper articles,
  - “police” appears with very high frequency

Stopwords

- Some terms add little *semantic* content to search
  - “to”, “and”, “the”
- These are called stopwords
- Terms that have a very high *frequency* are candidates
- Generally, manually polished by linguistics
- *Variety* in the corpus of documents is very important
- If only newspaper articles,
  - “police” appears with very high frequency
- Removing stopwords from queries may sometimes be erroneous
  - “to be or not to be”
- Therefore, web search engines do not bother to remove stopwords

Token Normalization

- Should documents containing “systems” be retrieved for “system”?
- Tokens are divided into equivalence classes

Token Normalization

- Should documents containing “systems” be retrieved for “system”?
- Tokens are divided into equivalence classes
- Stemming or lemmatization refers to stripping the word to its root or lemma
  - “system”, “systems”, “systematic”
  - Requires morphological analysis and is language specific

Token Normalization

- Should documents containing “systems” be retrieved for “system”?
- Tokens are divided into equivalence classes
- Stemming or lemmatization refers to stripping the word to its root or lemma
  - “system”, “systems”, “systematic”
  - Requires morphological analysis and is language specific
- Synonyms
  - “query”, “question”

Token Normalization

- Should documents containing “systems” be retrieved for “system”?
- Tokens are divided into equivalence classes
- Stemming or lemmatization refers to stripping the word to its root or lemma
  - “system”, “systems”, “systematic”
  - Requires morphological analysis and is language specific
- Synonyms
  - “query”, “question”
- Spelling versions
  - “color”, “colour”

Token Normalization

- Should documents containing “systems” be retrieved for “system”?
- Tokens are divided into equivalence classes
- Stemming or lemmatization refers to stripping the word to its root or lemma
  - “system”, “systems”, “systematic”
  - Requires morphological analysis and is language specific
- Synonyms
  - “query”, “question”
- Spelling versions
  - “color”, “colour”
- Token normalization finds more documents
  - Increases recall but decreases precision

Token Normalization

- Should documents containing “systems” be retrieved for “system”?
- Tokens are divided into equivalence classes
- Stemming or lemmatization refers to stripping the word to its root or lemma
  - “system”, “systems”, “systematic”
  - Requires morphological analysis and is language specific
- Synonyms
  - “query”, “question”
- Spelling versions
  - “color”, “colour”
- Token normalization finds more documents
  - Increases recall but decreases precision
- Character set is important
  - “pena” (sorrow) and “peña” (cliff) in Spanish

Documents

- What are documents?

Documents

- What are documents?
  - Entire book may or may not be useful
  - May be chapters or sections or sentences

Documents

- What are documents?
  - Entire book may or may not be useful
  - May be chapters or sections or sentences
- Depends on how context is defined

Outline

1. Introduction
   - Motivation
2. Document Retrieval
   - Inverted Index
3. Tokenization
   - Stopwords
   - Token Normalization
4. Scoring
   - Zone
   - Term Frequency
   - Inverse Document Frequency
   - Tf-idf
5. Document Vector
   - Vector Model
   - Document Similarity
6. Scalability
   - Skip List
   - Inexact Retrieval
7. System
8. Conclusions

Non-Boolean Match

- A query may be more general than just a set of terms
- It may itself be another document or some free text
- No document can be then expected to match it fully

Non-Boolean Match

- A query may be more general than just a set of terms
- It may itself be another document or some free text
- No document can be then expected to match it fully
- *Information retrieval task*
  - Given a free text query, find the most similar documents

Non-Boolean Match

- A query may be more general than just a set of terms
- It may itself be another document or some free text
- No document can be then expected to match it fully
- *Information retrieval task*
  - Given a free text query, find the most similar documents
- "Similarity" of a document d with the query q can be measured by a score $s(d, q)$

Non-Boolean Match

- A query may be more general than just a set of terms
- It may itself be another document or some free text
- No document can be then expected to match it fully
- *Information retrieval task*
  - Given a free text query, find the most similar documents
- "Similarity" of a document d with the query q can be measured by a score $s(d, q)$
- *Information retrieval task*
  - Given a free text query, find the documents that have the largest scores

Zones

- Each document is generally associated with metadata, e.g., title, author, date, etc.
- Sometimes, queries on indexes of these fields, called parametric indexes, are also asked
    - Find documents where “Indian” appears in the title

Zones

- Each document is generally associated with metadata, e.g., title, author, date, etc.
- Sometimes, queries on indexes of these fields, called parametric indexes, are also asked
  - Find documents where “Indian” appears in the title
- Fields are generalized to zones that may contain free text as well

Zones

- Each document is generally associated with metadata, e.g., title, author, date, etc.
- Sometimes, queries on indexes of these fields, called parametric indexes, are also asked
    - Find documents where “Indian” appears in the title
- Fields are generalized to zones that may contain free text as well
- Separate inverted indexes can be built for each zone
- Or, zone may be mentioned *explicitly* in a single inverted index

Indian.title → 11 → 34 → 56

Indian.body → 11 → 27 → 42 → 66

Indian → 11.title → 11.body → 27.body → 34.title → 42.body → 56.title → 66.body

Weighted Zone Scoring

- Each zone has a weight that adds up to 1
  - “title” has 0.3, and “body” has 0.7

Weighted Zone Scoring

- Each zone has a weight that adds up to 1
  - "title" has 0.3, and "body" has 0.7
- Given a query term, each zone scores a zone score
- It is 1 if the term is inside the zone; 0 otherwise

Weighted Zone Scoring

- Each zone has a weight that adds up to 1
  - “title” has 0.3, and “body” has 0.7
- Given a query term, each zone scores a zone score
- It is 1 if the term is inside the zone; 0 otherwise
- Weighted zone score is the linear sum of the zone scores

$$s(q, d) = \sum_{\forall z \in d} s_z(q)$$

Weighted Zone Scoring

- Each zone has a weight that adds up to 1
  - “title” has 0.3, and “body” has 0.7
- Given a query term, each zone scores a zone score
- It is 1 if the term is inside the zone; 0 otherwise
- Weighted zone score is the linear sum of the zone scores

$$s(q,d) = \sum_{\forall z \in d} s_z(q)$$

- Given a Boolean combination of query terms, each zone is scored
- These scores are then accumulated

Weighted Zone Scoring

- Each zone has a weight that adds up to 1
  - "title" has 0.3, and "body" has 0.7
- Given a query term, each zone scores a zone score
- It is 1 if the term is inside the zone; 0 otherwise
- Weighted zone score is the linear sum of the zone scores

$$s(q,d) = \sum_{\forall z \in d} s_z(q)$$

- Given a Boolean combination of query terms, each zone is scored
- These scores are then accumulated
- Weights of zones
  - Can be supplied by the application
  - Machine learned

Term Frequency

- Moving away from the binary model
- If a document contains a query term more number of times, it is more important and should score higher
- Weight of a document d is, therefore, simply the number of times the term t appears in it, called the term frequency

$$tf(t, d) = |t \in d|$$

Term Frequency

- Moving away from the binary model
- If a document contains a query term more number of times, it is more important and should score higher
- Weight of a document d is, therefore, simply the number of times the term t appears in it, called the term frequency

$$tf(t, d) = |t \in d|$$

- This assumes the bag of words model
- *Context* and *sequence* are lost
  - I love butter but I hate cheese
  - I love cheese but I hate butter

Inverse Document Frequency

- Certain terms may appear across all or most documents
  - “bat” in cricket pages
- Consequently, they discriminate little among the documents and are not useful

Inverse Document Frequency

- Certain terms may appear across all or most documents
  - “bat” in cricket pages
- Consequently, they discriminate little among the documents and are not useful
- Document frequency of a term is the number of documents it appears in
  - Lesser is more discriminative

Inverse Document Frequency

- Certain terms may appear across all or most documents
  - “bat” in cricket pages
- Consequently, they discriminate little among the documents and are not useful
- Document frequency of a term is the number of documents it appears in
  - Lesser is more discriminative
- If m is the total number of documents in the corpus

$$idf(t) = \log \frac{m}{df_t}$$

- This is called the inverse document frequency

Inverse Document Frequency

- Certain terms may appear across all or most documents
  - “bat” in cricket pages
- Consequently, they discriminate little among the documents and are not useful
- Document frequency of a term is the number of documents it appears in
  - Lesser is more discriminative
- If m is the total number of documents in the corpus

$$idf(t) = \log \frac{m}{df_t}$$

- This is called the inverse document frequency
- Logarithmic to make it less drastic

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t,d) = tf(t,d) \times idf(t)$$

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t,d) = tf(t,d) \times idf(t)$$

- The tf-idf score has following properties

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t, d) = tf(t, d) \times idf(t)$$

- The tf-idf score has following properties
  - *High* when t appears in a small number of documents

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t,d) = tf(t,d) \times idf(t)$$

- The tf-idf score has following properties
  - *High* when t appears in a small number of documents
  - *Low* when t appears in many documents

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t, d) = tf(t, d) \times idf(t)$$

- The tf-idf score has following properties
  - *High* when t appears in a small number of documents
  - *Low* when t appears in many documents
  - *High* when t appears many number of times in d

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t,d) = tf(t,d) \times idf(t)$$

- The tf-idf score has following properties
  - *High* when t appears in a small number of documents
  - *Low* when t appears in many documents
  - *High* when t appears many number of times in d
  - *Low* when t appears few number of times in d

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t, d) = tf(t, d) \times idf(t)$$

- The tf-idf score has following properties
  - *High* when t appears in a small number of documents
  - *Low* when t appears in many documents
  - *High* when t appears many number of times in d
  - *Low* when t appears few number of times in d
  - *Zero* if t does not appear at all in d

Tf-idf

- Combination of term frequency (tf) and inverse document frequency (idf)
- Weight of a term t in a document d is

$$tf\text{-}idf(t,d) = tf(t,d) \times idf(t)$$

- The tf-idf score has following properties
  - *High* when t appears in a small number of documents
  - *Low* when t appears in many documents
  - *High* when t appears many number of times in d
  - *Low* when t appears few number of times in d
  - *Zero* if t does not appear at all in d
- Tf-idf has many different forms

Outline

1 Introduction
- Motivation

2 Document Retrieval
- Inverted Index

3 Tokenization
- Stopwords
- Token Normalization

4 Scoring
- Zone
- Term Frequency
- Inverse Document Frequency
- Tf-idf

5 Document Vector
- Vector Model
- Document Similarity

6 Scalability
- Skip List
- Inexact Retrieval

7 System

8 Conclusions

Document Vector

- Each document d has a score with each term t in the vocabulary
  - If t is absent in d, then this score is 0

Document Vector

- Each document d has a score with each term t in the vocabulary
  - If t is absent in d, then this score is 0
- Imagine a n-dimensional vector space where n is the total number of terms in the vocabulary
- Each document can be, thus, thought of as a vector (point) in this n-dimensional space
- Its coordinates are the scores correponding to the scores

$$d[t_i] = tf\text{-}idf(t_i, d)$$

- This is called the document vector model

Exercise

- d_1: Water, water everywhere, not a drop to drink
- d_2: I have filtered water
- d_3: Drinking and driving is not good
- d_4: Water quality is not good here
- d_5: Milk is not good for health
- d_6: Drinking water just after dinner is not healthy

Exercise

- d_1: Water, water everywhere, not a drop to drink
- d_2: I have filtered water
- d_3: Drinking and driving is not good
- d_4: Water quality is not good here
- d_5: Milk is not good for health
- d_6: Drinking water just after dinner is not healthy
- Query q: drinkable water

Exercise

- d_1: Water, water everywhere, not a drop to drink
- d_2: I have filtered water
- d_3: Drinking and driving is not good
- d_4: Water quality is not good here
- d_5: Milk is not good for health
- d_6: Drinking water just after dinner is not healthy
- Query q: drinkable water
- Find tf, idf (with $\log_2$) and tf-idf ($\log_2$) scores

Similarity between Documents

- What is the "similarity" between two documents (i.e., their vectors)?

Similarity between Documents

- What is the "similarity" between two documents (i.e., their vectors)?
- *Euclidean distance* may not be suitable
    - Longer documents have larger distances

Cosine Similarity

- Consider two documents d_1 and d_2 with their corresponding document vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$
- Cosine similarity measures the *normalised* dot product

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1).\vec{V}(d_2)}{|\vec{V}(d_1)|.|\vec{V}(d_2)|}$$

  - Measures the cosine of the angle between the vectors

Cosine Similarity

- Consider two documents d_1 and d_2 with their corresponding document vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$
- Cosine similarity measures the *normalised* dot product

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1).\vec{V}(d_2)}{|\vec{V}(d_1)|.|\vec{V}(d_2)|}$$

  - Measures the cosine of the angle between the vectors
- Consider the length-normalised document vectors

$$\vec{v}(d_i) = \frac{\vec{V}(d_1)}{|\vec{V}(d_1)|}$$

- Then, cosine similarity is their dot product

$$\text{sim}(d_1, d_2) = \vec{v}(d_1).\vec{v}(d_2)$$

Example

Term	d_1	d_2	d_3
Indian	115	58	20
ancient	10	7	11
system	2	0	6

Example

Term	d_1	d_2	d_3
Indian	115	58	20
ancient	10	7	11
system	2	0	6
length	115.45	58.42	23.60

Example

Term	d_1	d_2	d_3
Indian	115	58	20
ancient	10	7	11
system	2	0	6
length	115.45	58.42	23.60

- Similarities between documents

$$\text{sim}(d_1, d_2) = \frac{115}{115.45} \cdot \frac{58}{58.42} + \frac{10}{115.45} \cdot \frac{7}{58.42} + \frac{2}{115.45} \cdot \frac{0}{58.42} = 0.99$$

- d_1 and d_2 is the closest pair

Query Vector

- Similar to a document, the query can also be viewed as a vector
- It is again just a *bag of words*
- Find similarities with documents and retrieve the top-k documents

Query Vector

- Similar to a document, the query can also be viewed as a vector
- It is again just a *bag of words*
- Find similarities with documents and retrieve the top-k documents
- Consider query q with the keywords "ancient" and "system"
- $\vec{V}(q) = (0, 1, 1)$ with $\vec{v}(q) = (0.00, 0.71, 0.71)$

Query Vector

- Similar to a document, the query can also be viewed as a vector
- It is again just a *bag of words*
- Find similarities with documents and retrieve the top-k documents
- Consider query q with the keywords “ancient” and “system”
- $\vec{V}(q) = (0, 1, 1)$ with $\vec{v}(q) = (0.00, 0.71, 0.71)$
- Then, most similar document is the one with the *highest* cosine similarity

Query Vector

- Similar to a document, the query can also be viewed as a vector
- It is again just a *bag of words*
- Find similarities with documents and retrieve the top-k documents
- Consider query q with the keywords “ancient” and “system”
- $\vec{V}(q) = (0, 1, 1)$ with $\vec{v}(q) = (0.00, 0.71, 0.71)$
- Then, most similar document is the one with the *highest* cosine similarity
- In previous example, sim(q, d_3) is highest
- Thus, d_3 is the most similar document

Query Vector

- Similar to a document, the query can also be viewed as a vector
- It is again just a *bag of words*
- Find similarities with documents and retrieve the top-k documents
- Consider query q with the keywords “ancient” and “system”
- $\vec{V}(q) = (0, 1, 1)$ with $\vec{v}(q) = (0.00, 0.71, 0.71)$
- Then, most similar document is the one with the *highest* cosine similarity
- In previous example, sim(q, d_3) is highest
- Thus, d_3 is the most similar document
- If $\vec{v}(q).\vec{v}(d_i)$ is the highest, then $\vec{V}(q).\vec{v}(d_i)$ is the highest as well
  - Normalization of query is not required

Query Vector

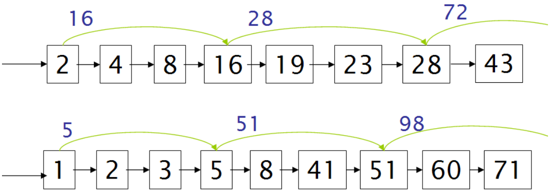
- Similar to a document, the query can also be viewed as a vector
- It is again just a *bag of words*
- Find similarities with documents and retrieve the top-k documents
- Consider query q with the keywords “ancient” and “system”
- $\vec{V}(q) = (0, 1, 1)$ with $\vec{v}(q) = (0.00, 0.71, 0.71)$
- Then, most similar document is the one with the *highest* cosine similarity
- In previous example, sim(q, d_3) is highest
- Thus, d_3 is the most similar document
- If $\vec{v}(q).\vec{v}(d_i)$ is the highest, then $\vec{V}(q).\vec{v}(d_i)$ is the highest as well
  - Normalization of query is not required
- Brute-force method of computing scores with all the documents and ranking them is *not* scalable

Outline

1. Introduction
   - Motivation
2. Document Retrieval
   - Inverted Index
3. Tokenization
   - Stopwords
   - Token Normalization
4. Scoring
   - Zone
   - Term Frequency
   - Inverse Document Frequency
   - Tf-idf
5. Document Vector
   - Vector Model
   - Document Similarity
6. Scalability
   - Skip List
   - Inexact Retrieval
7. System
8. Conclusions

Skip Lists

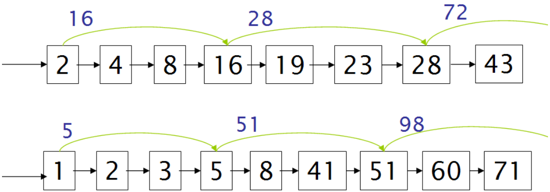
- Skip lists are used to traverse linked lists faster



- "Skips" are provided as jumps

Skip Lists

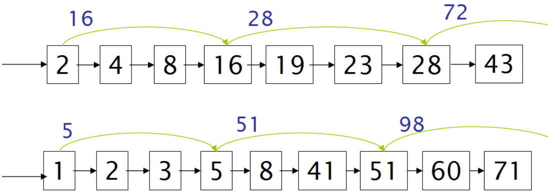
- Skip lists are used to traverse linked lists faster



- "Skips" are provided as jumps
- Useful when taking intersection of postings lists
- Once 41 is reached in the second list, 16 can jump to 28 in the first list

Skip Lists

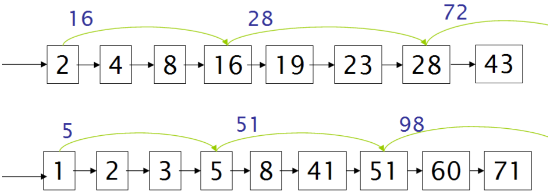
- Skip lists are used to traverse linked lists faster



- "Skips" are provided as jumps
- Useful when taking intersection of postings lists
- Once 41 is reached in the second list, 16 can jump to 28 in the first list
- How to determine skip positions?

Skip Lists

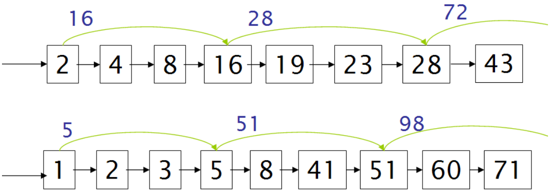
- Skip lists are used to traverse linked lists faster



- “Skips” are provided as jumps
- Useful when taking intersection of postings lists
- Once 41 is reached in the second list, 16 can jump to 28 in the first list
- How to determine skip positions?
- Look at closely occuring values; insert a skip to the end

Skip Lists

- Skip lists are used to traverse linked lists faster



- "Skips" are provided as jumps
- Useful when taking intersection of postings lists
- Once 41 is reached in the second list, 16 can jump to 28 in the first list
- How to determine skip positions?
- Look at closely occuring values; insert a skip to the end
- $\sqrt{l}$ equally spaced skips for a l-length list

Approximation

- Finding the *exact* top-k documents is too difficult
- Can resort to *approximate* top-k most relevant documents

Approximation

- Finding the *exact* top-k documents is too difficult
- Can resort to *approximate* top-k most relevant documents
- Notion of similarity is not perfect anyway
  - Cosine similairty is simply one measure

Approximation

- Finding the *exact* top-k documents is too difficult
- Can resort to *approximate* top-k most relevant documents
- Notion of similarity is not perfect anyway
  - Cosine similairty is simply one measure
- Information needs are generally satisfied with approximate answers
  - Google search

Approximation

- Finding the *exact* top-k documents is too difficult
- Can resort to *approximate* top-k most relevant documents
- Notion of similarity is not perfect anyway
  - Cosine similairty is simply one measure
- Information needs are generally satisfied with approximate answers
  - Google search
- Retrieval process becomes much faster

Approximation

- Finding the *exact* top-k documents is too difficult
- Can resort to *approximate* top-k most relevant documents
- Notion of similarity is not perfect anyway
  - Cosine similairty is simply one measure
- Information needs are generally satisfied with approximate answers
  - Google search
- Retrieval process becomes much faster
- Generally, a two-step process
  1. Retrieve *approximate* top-K documents where $k \leq K \ll m$
  2. Retrieve *exact* top-k from K

Improving Efficiency

Improving Efficiency

- Only terms that appear in query need to be examined
  - Rest of the scores are 0

Improving Efficiency

- Only terms that appear in query need to be examined
  - Rest of the scores are 0
- Filter query terms whose *idf* is too low
  - Similar to stop-word idea
  - “to be or not to be”

Improving Efficiency

- Only terms that appear in query need to be examined
  - Rest of the scores are 0
- Filter query terms whose *idf* is too low
  - Similar to stop-word idea
  - “to be or not to be”
- *Pre-compute* champion lists for each term
  - Documents ranked for only that term
  - Offline process
  - Take union of top-r of every query term to get top-K

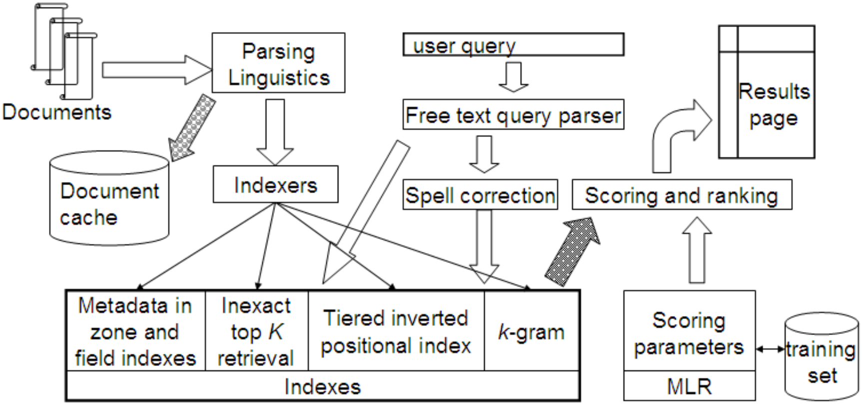
Improving Efficiency

- Only terms that appear in query need to be examined
  - Rest of the scores are 0
- Filter query terms whose *idf* is too low
  - Similar to stop-word idea
  - “to be or not to be”
- *Pre-compute* champion lists for each term
  - Documents ranked for only that term
  - Offline process
  - Take union of top-r of every query term to get top-K
- Build tiered index
  - Each level (tier) lists only those documents whose *tf* for the term is greater than a threshold
  - Continue with tiers till top-K results are obtained

Outline

1. Introduction
   - Motivation
2. Document Retrieval
   - Inverted Index
3. Tokenization
   - Stopwords
   - Token Normalization
4. Scoring
   - Zone
   - Term Frequency
   - Inverse Document Frequency
   - Tf-idf
5. Document Vector
   - Vector Model
   - Document Similarity
6. Scalability
   - Skip List
   - Inexact Retrieval
7. System
8. Conclusions

The Complete Information Retrieval System



Outline

1 Introduction
- Motivation

2 Document Retrieval
- Inverted Index

3 Tokenization
- Stopwords
- Token Normalization

4 Scoring
- Zone
- Term Frequency
- Inverse Document Frequency
- Tf-idf

5 Document Vector
- Vector Model
- Document Similarity

6 Scalability
- Skip List
- Inexact Retrieval

7 System

8 Conclusions

Conclusions

- Inverted Index
- Tokenization
- Term Frequency, Document Frequency (Tf-idf)
- Document Vector
- Document Scoring

Conclusions

- Inverted Index
- Tokenization
- Term Frequency, Document Frequency (Tf-idf)
- Document Vector
- Document Scoring

THANK YOU!

Conclusions

- Inverted Index
- Tokenization
- Term Frequency, Document Frequency (Tf-idf)
- Document Vector
- Document Scoring

THANK YOU!

Questions?
Answers!