Web Content can be dynamically generated

- Most web is static but some could be dynamically generated.
    - Ex. Generating when scrolling down, based on user's preference (Weibo, Instagram, facebook)

GenerativeAI Limitation:

- Human beings can answer "Don't know" to a specific question, but AI cannot.
    - Based on "Massive word embedding", AIs generate answers step by step and only move forward.

Metrics based on Confusion Matrix Data

		Actual	
		Dog	**Not Dog**
Predicted	**Dog**	True Positive (TP =5)	False Positive (FP=1)
	Not Dog	False Negative (FN =1)	True Negative (TN=3)

1. ***Accuracy***

Accuracy is used to measure the performance of the model. It is the ratio of Total correct instances to the total instances.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

For the above case:

Accuracy = (5+3)/(5+3+1+1) = 8/10 = 0.8

2. ***Precision***

Precision is a measure of how accurate a model's positive predictions are. It is defined as the ratio of true positive predictions to the total number of positive predictions made by the model.

$$\text{Precision} = \frac{TP}{TP+FP}$$

For the above case:

Precision = 5/(5+1) =5/6 = 0.8333

3. ***Recall***

Recall measures the effectiveness of a classification model in identifying all relevant instances from a dataset. It is the ratio of the number of true positive (TP) instances to the sum of true positive and false negative (FN) instances.

$$\text{Recall} = \frac{TP}{TP+FN}$$

For the above case:

Recall = 5/(5+1) =5/6 = 0.8333

4. ***F1-Score***

F1-score is used to evaluate the overall performance of a classification model. It is the harmonic mean of precision and recall,

$$\text{F1-Score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

For the above case:

F1-Score: = (2* 0.8333* 0.8333)/(0.8333+ 0.8333) = 0.8333

We balance precision and recall with the F1-score when a trade-off between minimizing false positives and false negatives is necessary, such as in information retrieval systems.

5.**In IR:**

Precision in IR:

- Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances. Written as a formula:

$$\text{Precision} = \frac{\text{Relevant retrieved instances}}{\text{All retrieved instances}}$$

Recall in IR:

- Recall (also known as sensitivity) is the fraction of relevant instances that were retrieved. Written as a formula:

$$\text{Recall} = \frac{\text{Relevant retrieved instances}}{\text{All relevant instances}}$$

- ***Recall*** only gets **HIGHER** in IR!

Formalizing Precision/Recall

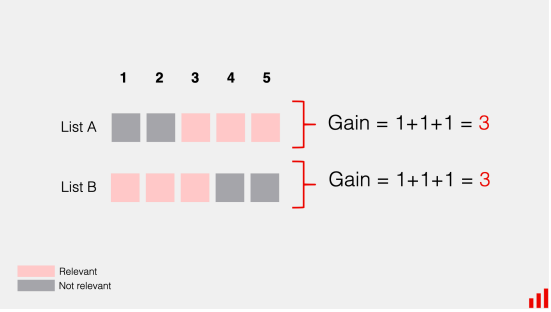
For web applications, ***Precision*** is more important than ***Recall***

Precision/Recall Two Observations

• You can get high ***recall*** (but low precision) by retrieving all docs for all queries!
  - a rather foolish strategy
• In a good system, ***precision*** decreases as the number of docs retrieved (or ***recall***) increases
  - This is not a theorem, but a result with strong empirical confirmation
  - E.g. viewing multiple pages of Google results often does not improve ***precision*** at all

Evaluation Measures

1. Calculating Recall/Precision at Fixed Positions
2. Average Precision of the Relevant Documents
3. Averaging Across Queries
4. Mean Average Precision (MAP)
5. Discounted Cumulative Gain
    - The idea is to ‘punish’ results that are further down the line from the top, even if they are relevant - because we want the relevant results to be at/near the top.
    - Gain without ‘Discount’



-> need to ‘punish’ the result with lower rank

- Using a logarithm for the position penalty makes the decay effect more gradual compared to using the position itself.

How Evaluation is Done at Web Search Engines

- Search engines have test collections of queries and hand-ranked results
- Recall is difficult to measure on the web
- Search engines often use precision at top k positions, e.g., k = 10
  or measures that reward you more for getting rank / right than for getting rank 10 right.
- Search engines also use non-relevance-based measures
    - Click-through on first result
        - Not very reliable for single click-through, but pretty reliable in the aggregate.
    - Studies of user behavior in the lab
    - A/B testing
        - Another B ver. of google in some area/domain/…,
        - update the whole world from A ver. to B if B ver. works better.

A/B Testing at Web Search Engines

A/B testing: comparing two versions of a web page to see which one performs better. You compare two web pages by showing the two variants (let's call them A and B) to similar visitors at the same time. The one that gives a better conversion rate, wins!

1. **Purpose**: Test a single innovation
2. **Prerequisite**: You have a large search engine up and running.
3. **Have most users use old system**
4. Divert a **small proportion** of traffic (e.g., 1%) to an experiment to evaluate an innovation
5. **Evaluate with an automatic measure** like click through on first result

Web Crawling Issues

How to crawl?

- **Quality**: how to find the “Best” pages first
- **Efficiency**: how to avoid ***duplication*** (or near duplication)
- **Etiquette**: behave politely by not disturbing a website’s performance

Simple Picture - Complications

Crawling the entire web isn’t feasible with one machine

- Steps can be distributed

Challenges

- Handling/Avoiding malicious pages
  - Some pages contain spam
  - Some pages contain spider traps - especially dynamically generated pages
- Even non-malicious pages pose challenges
  - ***Latency / bandwidth*** to remote servers can vary widely
  - ***Robots.txt*** stipulations can prevent web pages from being visited
  - How can one avoid mirrored sites and duplicate pages
- Maintain politeness - don’t hit a server too often

robots.txt :

- *robots.txt* in the root of the web site hierarchy (e.g. https://www.example.com/robots.txt).
- A *robots.txt* file contains instructions for bots indicating which web pages they can and cannot access. Robots.txt files are particularly important for web crawlers from search engines such as Google.
- sites with spider traps usually have a *robots.txt* telling bots **not** to go to the trap, so a legitimate "polite" bot would not fall into the trap, whereas an "impolite" bot which disregards the robots.txt settings would be affected by the trap

Link Extraction

URLs occur in tags other than <a>,

- E.g. <frame src=“site-index.html”>, <area, href=“…”>, <meta>, <link>, <script>

Two Anomalies

1. Some anchors don’t have links, e.g. <a name=“banner”>
2. Some anchors produce dynamic pages which can lead to looping
   E.G., <a href=http://www.mysite.com/search?x=arg1&y=arg2>

Representing URLs

Speed up Methods

1. To determine if a new URL has already been seen
    - First hash on host/domain name, then
    - Use a trie data structure to determine if the path/resource is the same as one in the URL database
2. URLs are sorted lexicographically and then stored as a delta-encoded text file
    - Each entry is stored as the difference (delta) between the current and previous URL; this substantially reduces storage
    - However, restoring the actual URL is slower, requiring all deltas to be applied to the initial URL
    - To improve speed, checkpointing (storing the full URL) is done periodically

Handling Spam Web Pages

The first generation:

- consisted of pages with a high number of repeated terms, so as to score high on search engines that ranked by word frequency
- Words were rendered in the same color as the background, so as to not be visible, but still count

The second generation: cloaking

- Give a different page to a crawler than the page it returns to users.

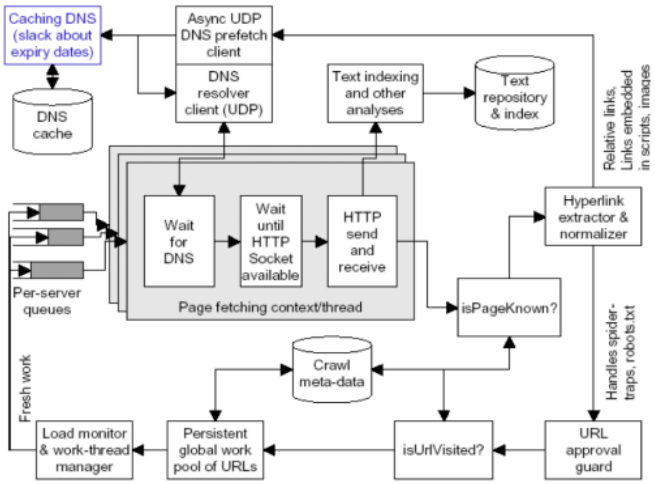
The third generation: doorway pages

- contains more data chosen to rank highly, users get a more “commercially oriented” (more ads) page

Cloaking and Doorway pages:

- “Cloaking”: crawlers get a worse result
- “Doorway”: users get a worse result

The Mercator Web Crawler



Measuring and Tuning a Crawler

Measuring and tuning a crawler for peak performance

- Improving URL parsing speed
- Improving network bandwidth speed
- Improving fault tolerance

More Issues (some of which are discussed ahead)

- Refresh Strategies: how often is the process re-started
- Detecting duplicate pages
- Detecting mirror sites
- Speeding up DNS lookup (see previous slide)
- URL normalization (discussed earlier)
- Handling malformed HTML

Issues and Benefits of Distributed Crawling

- Benefits:
- scalability: for large-scale web-crawls
- costs: use of cheaper machines
- network-load dispersion and reduction: by dividing the web into regions and crawling only the nearest pages
- Issues:
- overlap: minimization of multiple downloaded pages
- quality: depends on the crawl strategy
- communication bandwidth: minimization

Coordination of Distributed Crawling

Three strategies

1. Independent:
   ► no coordination, every process follows its extracted links
2. Dynamic assignment:
   ► a central coordinator dynamically divides the web into small partitions and assigns each partition to a process
3. Static assignment:
   ► Web is partitioned and assigned without a central coordinator before the crawl starts

General Conclusions of Cho and Garcia-Molina

- Firewall crawlers attain good, general coverage with low cost
- Cross-over ensures 100% quality, but suffer from overlap
- Replicating URLs and batch communication can reduce Overhead

Cho and Garcia-Molina, 2000

Two simple re-visiting policies

- ***Uniform*** policy: This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.
- ***Proportional*** policy: This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.

• In terms of average freshness, the ***uniform policy*** outperforms the proportional policy in both a simulated Web and a real Web crawl.
- the explanation for this result comes from the fact that, when a page changes too often, the crawler will waste time by trying to recrawl it too fast and still will not be able to keep its copy of the page fresh.

• To improve freshness, we should penalize the elements that change too often

Implications for a Web Crawler

"in-place" or ***"shadowing"*** when a crawler replaces an old version by a new page
- **Shadowing**: implies a new set of pages are collected and stored separately and all are updated at the same time
- The above implies that queries need to check two databases, the current database and the database of new pages
- Shadowing either slows down query processing or decreases freshness

Conclusions:
- running multiple types of crawlers is best
- Updating in-place keeps the index current

Help the Search Engine Crawler Creating a SiteMap

Sitemap: a list of pages of a web site accessible to crawlers
- This helps search engine crawlers find pages on the site
- XML is used as the standard for representing sitemaps
- A sitemap can be posted anywhere on your site, but a sitemap affects only descendants of the parent directory

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset
        xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
        <url>
                <loc>http://www.example.com/?id=who</loc>
                <lastmod>2009-09-22</lastmod>
                <changefreq>monthly</changefreq>
                <priority>0.8</priority>
        </url>
        <url>
                <loc>http://www.example.com/?id=how</loc>
                <lastmod>2009-09-22</lastmod>
                <changefreq>monthly</changefreq>
                <priority>0.5</priority>
        </url>
</urlset>
```

Deduplication:

- the process of identifying and avoiding essentially identical web pages
- In web crawling, de-duplication essentially refers to the identification of identical and nearly identical web pages and indexing only a single version to return as a search result

Mirroring

- Mirroring is the systematic replication of web pages across hosts.
  - Mirroring is the **single largest cause of duplication** on the web
  - ***Host1/A*** and ***Host2/B*** are mirrors iff
    - For all (or most) paths ***p*** such that when
      http://Host1/A / p exists
      http://Host2/B/ p exists as well
      with identical (or near identical) content, and vice versa.

P10

Reasons for Deduplication:

- Smarter crawling

- Avoid returning many duplicate results to a query
- Allow fetching from the fastest or freshest server

- Better connectivity analysis

- By combining in-links from the multiple mirror sites to get an accurate PageRank (measure of importance)
- Avoid double counting out-links

- Add redundancy in result listings

- "If that fails you can try: <mirror>/samepath'

- Reduce Crawl Time: Crawlers need not crawl pages that are identical or near identical

- Ideally: given the web's scale and complexity, priority must be given to content that has **not** already been seen before or has recently changed

- Saves resources (on the crawler end, as well as the remote host)
- Increases crawler politeness
- Reduces the analysis that a crawler will have to do later

P12

1. *Duplicate Problem*: Exact match;

- Solution: compute fingerprints using **cryptographic hashing**
- Useful for URL matching and also works for detecting identical web pages
- Hashes can be stored in sorted order for ***log N*** access (better than *N*log N* !)
  - Importance of designing a **good** hashing function
  - e.g., MD5, SHA-1,... SHA-3, Whirlpool

2. *Near-Duplicate Problem*: Approximate match

- Solution: compute the syntactic similarity with an edit-distance measure, and
- Use a similarity threshold to detect near-duplicates
  - e.g., Similarity > 80% => Documents are "near duplicates"

cryptographic hash function

- A cryptographic hash function is a hash function which takes an input (or 'message') and returns a fixed-size *alphanumeric string*, which is called the hash value (sometimes called a message digest, digital fingerprint, digest or a checksum).
- e.g., MD5, SHA-1,... SHA-3, Whirlpool
- Four main properties:
    1. It is extremely easy (i.e. fast) to calculate a hash for any given data.
    2. It is extremely computationally difficult to calculate an alphanumeric text that has a given hash.
    3. A small change to the text yields a totally different hash value.
    4. It is extremely unlikely that two slightly different messages will have the same hash

Deduplication Approaches:

Not Good:

1. ***Compare character by character*** two documents to see if they are identical
    - very time consuming !!
2. ***Hash just the first few characters*** and compare only those documents that hash to the same bucket
    - But what about web pages where every page begins with <HTML>??
3. ***Use a hash function that examines the entire document***
    - But this requires lots of buckets
4. ***Better approach*** - pick some fixed random positions for all documents and make the hash function depend only on these;
    - This avoids the problem of a common prefix for all or most documents, yet we need not examine entire documents unless they fall into a bucket with another document
    - But we still need a lot of buckets
5. ***Even better approach***: Compute the **cryptographic hash** (SHA-2 or MD5) of each web page and maintain in sorted order, O(*log n*) to search

Better ways:

1. ***Produce fingerprints and test for similarity*** - Treat web documents as defined by a set of features, constituting an *n*-dimensional vector, and transform this vector into an *f*-bit fingerprint of a small size
    - Use Simhash or Hamming Distance to compute the fingerprint
      • SimHash is an algorithm for testing how similar two sets are
    - Compare fingerprints and look for a difference in at most *k* bits
2. ***Instead of documents defined by n-vector of features, compute subsets of words (called* shingles*) and test for similarity of the sets***
    - Broder et al., WWW 1997, Finding Near Duplicate Documents

    Procedure:
    1. Define a function that captures the contents of each document in a number
        - E.g. hash function, signature, or a fingerprint
    2. Create the pair <f(doc), ID of doc; for all doc;

3. Sort the pairs
4. Documents that have the same f-value or an f-value within a small threshold are believed to be duplicates or near duplicates

Distance measures (to compute similarity)

- To compute similarity, we need a distance measure.
- Distance measures in locating duplicate and near- duplicate documents:

1. ***Euclidean distance***
   - D([X]...X], [Y1,...,yn]) = sqrt(Sum(x; y)^2) i=1...n
2. ***Jaccard distance***
   - D(x,y) = 1 SIM(x,y) or 1 minus the ratio of the sizes of the intersection and union of sets x and y
3. ***Cosine distance***
   - the cosine distance between two points (two n element vectors) is the angle that the vectors to those points make; in the range 0 to 180 degrees
4. ***Edit distance***
   - the distance between two strings is the smallest number of insertions and deletions of single characters that will convert one string into the other
5. ***Hamming distance***
   - between two vectors is the number of components in which they differ (usually used on Boolean vectors)

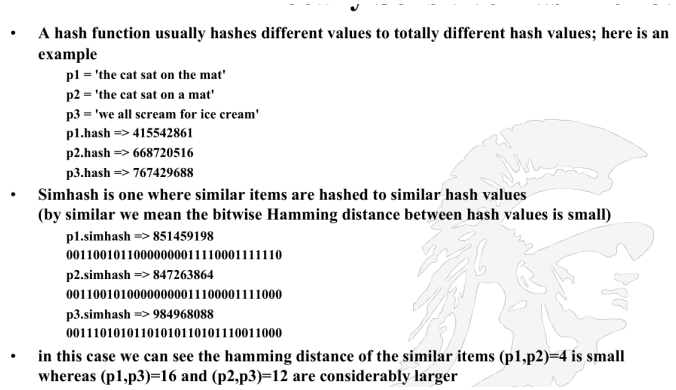
Shingle:

- a contiguous subsequence of words in a document is called a shingle;
- *4-shingling*: 4-shingling of the phrase below produces a bag of 5 items:
  - "a rose is a rose is a rose" => a set S(D,w) is defined as {(a rose is_a), (rose is_a_rose), (is_a_rose_is), (a_rose_is_a), (rose_is_a_rose)}
- *S(D,w)* is the set of shingles of a document *D* of width *w*
- Similarity Measures
  - *Jaccard(A,B)* (also known as Resemblance) is defined as
    - size of (S(A,w) intersect S(B,w)) / size of (S(A,w) union S(B,w))
  - *Containment(A, B)* is defined as
    - size of (S(A,w) intersect S(B,w))/ size of (S(A,w))
  - 0 <= Resemblance <= 1
  - 0 <= Containment <= 1

SimHash

- documents that are nearly identical have nearly similar fingerprints that **differ only in a small # of bits**. In other words, similar inputs lead to similar outputs (hash values), hence 'Sim'Hash; other hashing techniques, eg. MD5, do **not** have this property (in other words, even a tiny change in the input leads to a **huge** change in the output). This

similarity property is what makes SimHash, an excellent tool for similarity detection of documents.



Normal way: for 'n' documents, comparing them all pairwise would take a long time [O*(n^2)*] comparing SimHash values (ie computing 'Charikar Similarity' values): **Faster**

Search Logics:

1. Keyword matching
    - Simplest notion of relevance is that the query string appears verbatim in the document.
        - any order (like viewing the document as a ‘bag of words’).
    - But that may not retrieve relevant documents that include synonymous terms.
        - "restaurant" vs. "café"
    - And it may retrieve irrelevant documents that include ambiguous terms.
        - "Apple" (company vs. fruit)
2. More than keyword matching
    - Takes into account the *meaning* of the words used
    - Takes into account the *order* of words in the query
    - Adapts to the user based on direct or indirect feedback
    - Takes into account the *authority* of the source

P19

- Parsing forms index words (tokens) and includes:
    1. *Stopword* removal
    2. *Stemming*: reducing a word to its root
- **Indexing** constructs an **inverted index** of word to document pointers.
- **Searching** retrieves documents that contain a given query token from the inverted index.
- **Ranking** scores all retrieved documents according to a relevance metric.

inverted index:

e.g.,

id	content
101	‘Multi cloud’
102	‘Elastic scale’
103	‘Multi region’
104	‘Cloud native’

→

token	id
multi	101, 103
cloud	101, 104
elastic	102
scale	102
region	103
native	104

A retrieval model specifies the details of:

- Document representation
- Query representation
- Retrieval function

Three major Information Retrieval Models are:

1. *Boolean models* (set theoretic) (Chapter 1 in Manning et al)
2. *Vector space models* (statistical/algebraic) (Chapter 2 in Manning et al)
3. *Probabilistic models* (Chapter 11 in Manning et al)

A document is represented as a **set** of keywords.
Queries are Boolean expressions of keywords, connected by AND,

- OR, and NOT, including the use of brackets to indicate scope
    - [[Rio & Brazil] | [Hilo & Hawaii]] & hotel & !Hilton]

Vector space models

- One way to compute the weight is to use the term's frequency in the document
- Assumption: the more frequent terms in a document are more important, i.e. more indicative of the topic.

    f_{ij} **= frequency** of **term** ***i*** in **document** ***j***
- May want to normalize term frequency (*tf*) across the entire corpus:

    $tf_{ij} = f_{ij} / \max\{f_{ij}\}$

- Terms that appear in many *different* documents are less indicative of overall topic

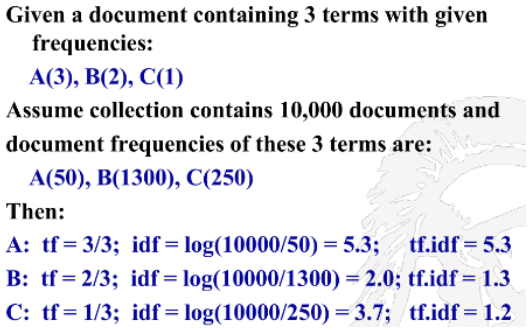
    df_i = document frequency of **term** ***i***
    = number of documents containing **term** ***i***
    of course df_i is always <= *N* (total number of documents)
    idf_i = inverse document frequency of **term** ***i***,
    = $\log_2 (N / df_i)$ (*N*: total number of documents)

- A typical combined term importance indicator is ***tf-idf*** weighting (note: it is often written with a hyphen, but the hyphen is NOT a minus sign; some people replace the hyphen with a dot):

    $W_{ij} = tf_{ij}.idf_i = (1 + \log tf_{ij}) * \log_2 (N / df_i)$

- Given a query q, then we score the query against a document d using the formula

    $Score(q, d) = \sum(tf.idf_{t,d})$ **where t is in q** ∩ **d**
- e.g.,



Classification Methods

1. *Manual classification*
    - Used by the original Yahoo! Directory, Looksmart, about.com, ODP, PubMed
    - Accurate when job is done by experts
    - Consistent when the problem size and team is small
    - Difficult and expensive to scale
        - Means we need automatic classification methods for big problems
2. *Hand-coded rule-based classifiers*
    - One technique used by news agencies, intelligence agencies, etc.
    - Widely deployed in government and enterprises
    - Vendors provide “IDE” for writing such rules
    - Accuracy is can be high if a rule has been carefully refined over time by a subject expert
    - Building and maintaining these rules is expensive
3. *Supervised learning*
    - Naive Bayes (simple, common), k-Nearest Neighbors (simple, powerful), Support-vector machines (newer, generally more powerful)...
    - No free lunch: requires hand-classified training data
    - But data can be built up (and refined) by amateurs
    - Many commercial systems use a mixture of methods (= Ensemble learning)
        - Ensemble learning: a machine learning technique that enhances accuracy and resilience in forecasting by merging predictions from multiple models.

Feature Selection: Why?

1.Text collections have a large number of features
    - 10,000 - 1,000,000 unique words and more
2. Selection may make a particular classifier feasible
    - Some classifiers can't deal with 1,000,000 features
3. Reduces training time
    - Training time for some methods is quadratic or worse in the number of features
4. Makes runtime models smaller and faster
5. Can improve generalization (performance)
    - Eliminates noise features
    - Avoids overfitting

Feature Selection: Frequency

The simplest feature selection method:

- Just use the most common terms
- No particular foundation
- But it make sense why this works
    - They are the words that can be well-estimated and are most often available as evidence
    - In practice, this is often 90% as good as better methods

Naive Bayes is Not So Naive

• Very fast learning and testing (basically just count words)
• Low storage requirements
• Very good in domains with many equally important features
• More robust to irrelevant features than many learning methods
- Irrelevant features cancel each other without affecting results

Evaluating Categorization

• Measures: precision, recall, F1, classification *accuracy*
• **Classification *accuracy***: ***r/n*** where *n* is the total number of test docs and *r* is the number of test docs correctly classified

Classification Using Vector Spaces

In vector space classification, training set corresponds to a labeled set of points (equivalently, vectors)
Premise 1: Documents in the **same class form a contiguous region** of space
Premise 2: Documents from different classes **don't overlap (much)**
Learning a classifier: build **surfaces** to delineate classes in the space

Definition of centroid

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$$

• Where D_c is the set of all documents that belong to class *c* and *v(d)* is the vector space representation of *d*.
• Note that centroid will in general not be a unit vector even when the inputs are unit vectors

k Nearest Neighbor Classification

• kNN = *k* Nearest Neighbor
• To classify a document d:
• Define *k*-neighborhood as the *k* nearest neighbors of *d*
• Pick the majority class label in the *k*-neighborhood
• For larger k can roughly estimate *P(c|d)* as *#(c)/k*

• Using only the closest example (1NN) subject to errors due to:
- A single atypical example.
- Noise (i.e., an error) in the category label of a single training example.
• More robust: find the k examples and return the majority category of these k
• k is typically odd to avoid ties; 3 and 5 are most common

KNN: Discussion

- No feature selection necessary
- No training necessary
- Scales well with large number of classes
  - Don't need to train n classifiers for n classes
- Classes can influence each other
  - Small changes to one class can have ripple effect
- Done naively, very expensive at test time
- In most cases it's more accurate than NB or Rocchio

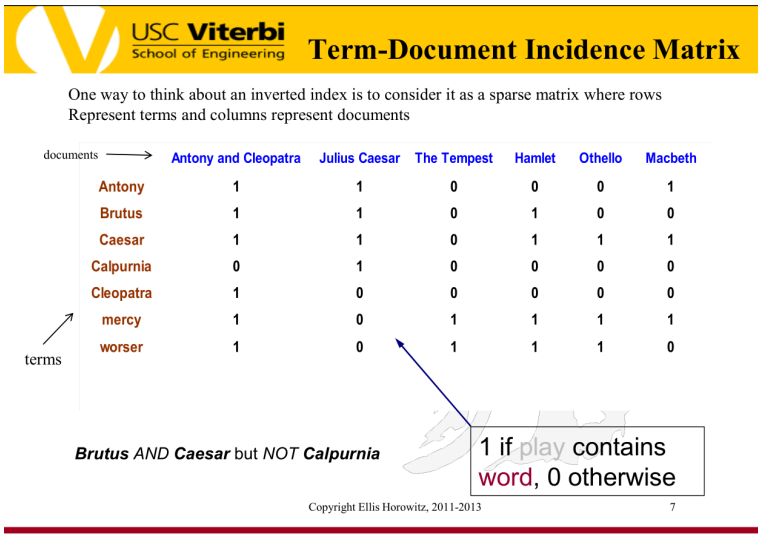
Creating an Inverted Index

An inverted index is typically composed of a vector containing all distinct words of the text collection in lexicographical order (which is called the **vocabulary**) and for each word in the vocabulary, a list of all documents (and text positions) in which that word occurs

- This is nothing more than an index that one finds at the back of a book

Terms in the inverted file index are refined:

- ***Case folding***: converting all uppercase letters to lower case
- ***Stemming***: reducing words to their morphological roots
- ***Stop words***: removing words that are so common they provide no information



Inverted Indexes are Naturally Sparse

Given 1 million documents and 500,000 terms (英语常用词数量)
The term x Document matrix in this case will have size 500K x 1M or half- a-trillion 0's and 1's.
But it has no more than one billion 1's.

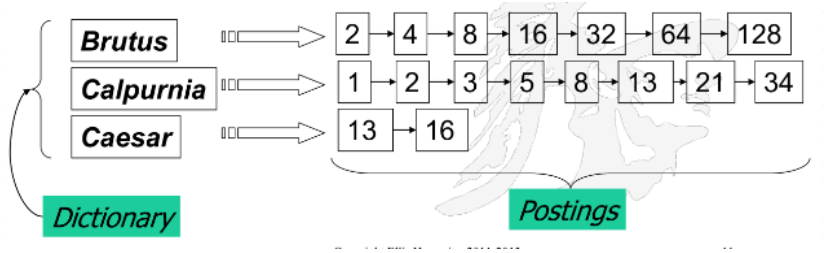
- So the matrix is extremely sparse, only 0.1% of the elements are 1

So instead we use a data structure for an inverted index that exploits sparsity and then devise algorithms for query processing

Inverted Index Stored In Two Parts

For each term T, we must store a list of all documents that contain T.
Linked lists are generally preferred to arrays
Dynamic space allocation
Insertion of terms into documents easy

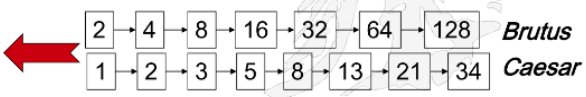
However, there is space overhead of pointers, though this is not too serious



Query Processing Across the Postings List

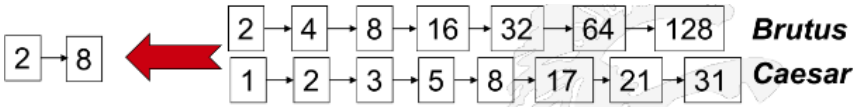
Consider processing the query:
Brutus AND Caesar

- Locate Brutus in the Dictionary;
    - Retrieve its postings.
- Locate Caesar in the Dictionary;
    - Retrieve its postings.
- "Merge" the two postings (postings are document ids):



Basic Merge

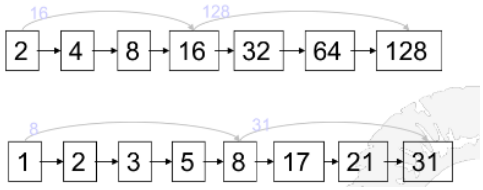
Walk through the two postings simultaneously, in time linear in the total number of postings entries



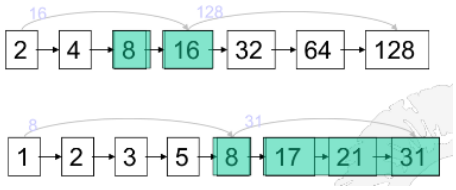
<u>If the list lengths are m and n, the merge takes O(m+n) operations</u>.
→ How to speed up ?

The Technique of Skip Pointers

Augment postings with skip pointers (at indexing time)



→ <u>To skip postings that will not figure in the search results</u>



- Suppose we've stepped through the lists until we process 8 on each list.
- When we get to 16 on the top list, we see that its successor is 32.
- But the skip successor of 8 on the lower list is 31, so we can skip ahead past the intervening postings.

Phrase queries

- phrase: "stanford university"
- sentence: "I went to university at Stanford"

→ No longer suffices to store only

Using Biword Indexes for Phrase Searching

- A biword (or a 2-gram) is a consecutive pair of terms in some text
- e.g., "Friends, Romans, Countrymen":
    - (friends, romans), (romans, countrymen)
- Consequences
    - Biwords will cause an explosion in the vocabulary database
    - Queries longer than 2 words will have to be broken into biword segments

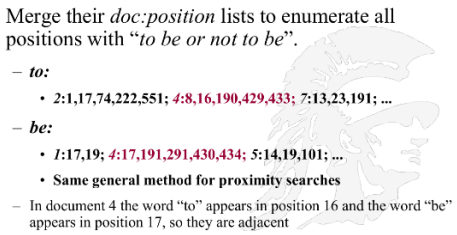
→ but may also produce false positives (i.e. occurrences of the biwords, but not the full 4 word query)

Alternate Solution Using Positional Indexes

Store, for each term, entries of the form:

```
<number of docs containing term;
doc1: position1, position2 ... ;
doc2: position1, position2... ;
etc.>
```

- e.g., for each term in the vocabulary, we store postings of the form `docID: position1, position2, ...,` where each position is a token index in the document. Each posting will also usually record the term frequency


Merge their *doc:position* lists to enumerate all positions with "*to be or not to be*".

– ***to:***
  • *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
– ***be:***
  • *1*:17,19; *4*:17,191,291,430,434; *5*:14,19,101; ...
  • **Same general method for proximity searches**
– In document 4 the word "to" appears in position 16 and the word "be" appears in position 17, so they are adjacent

Adopting a positional index expands required postings storage significantly, even if we compress position values/offsets

Building n-gram Indexes

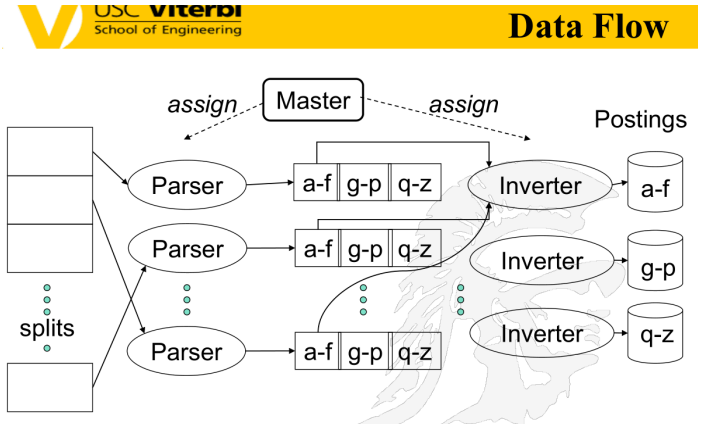
- Generalizing from bi-words, an ***n-gram*** is any sequence of *n* consecutive words
- N-grams can be identified at the time of parsing
- N-grams of all lengths form a Zipf distribution with a few common phrases occurring very frequently and a large number occurring with frequency 1
- Common n-grams are usually made up of stop words (e.g. "and the" "there is")
- For each n-gram, the inverted index will need pointers to all dictionary terms containing it - the "postings"
- Therefore, the larger n, the larger space required to hold all ***n-grams***

Distributed Indexing

- One approach is to use two sets of parallel tasks
  - Parsers
  - Inverters
- Break the input document corpus into *splits*
  - Each *split* is a subset of documents
- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into j partitions
- Each for a range of terms' first letters (e.g., a-f, g-p, q-z) - here j=3.

inverters:

- Collect all (term, doc) pairs for a partition
- Sorts and writes to postings list
- Each partition contains a set of postings



Video Search Engines – Quick Summary

video search engine:

- a web-based search engine which crawls the web primarily for video content.
- YouTube is not strictly a video search engine as it does not crawl the web looking for video content

The **indexing** of video content:

- normally done by acquiring ***meta-data*** associated with the video,
- e.g. Author, title, creation date, duration, coding quality, tags, description

The **ranking** of videos under a query is generally done using:

- Relevance: using metadata and user preferences
- Ordered by date of upload
- Ordered by number of views
- Ordered by duration
- Ordered by user rating

→ So indexing and ranking are a lot **simpler** than for document search engines

YouTube Recommendation System Uses Graph Properties

Association Rule Mining

- For each pair of videos v_i, v_j; compute co-visitation counts, i.e. they count how often they were co-watched; if $c_{i,j}$, is the co-visitation count, then relatedness is defined as

$$r(v_i, v_j) = \frac{c_{ij}}{f(v_i, v_j)}$$

  - where c_i; and c_j; are the total occurrence counts across all sessions for videos v_i and v_j; . f(v_i, v_j) is a normalization function that takes the global popularity of both the seed video and the candidate video into account; e.g. f(Vi, Vj) = Ci * Cj
  - The set of related videos, R; for a given seed video vi is determined by taking the top N candidate videos ranked by their scores (vi, Vj)
  - Related videos induce a directed graph over the set of videos, namely: For each pair of videos (vi, vj), there is an edge $e_{i,j}$, from v_i to v_j; iff vi is in to vj; iff Vj is in Ri

Content Delivery Networks

content distribution network (CDN)

- CDN consists of a large set of content servers and a means for dynamically selecting servers based on the location of the user and possibly the content being requested

YouTube Video Delivery System

Two Critical Technology Challenges for YouTube:

- how to *identify* billions of videos
- How to *efficiently deliver* the video to the desktop/mobile device

The Solutions:
Identification: YouTube assigns a fixed-length, 11 character string, base 64, unique identifier to each video, see: link
Efficient Delivery: YouTube makes use of Google's data centers using them as a content distribution network, see: link

Sound Generate AI: Synthesizer V

Stemming and lemmatization

Stemming:

- usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

Lemmatization:

- Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma .

e.g.,

For token ***'saw'***, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun.

- Also, **stemming** most commonly collapses derivationally related words, whereas **lemmatization** commonly only collapses the different inflectional forms of a lemma.

Query Box Default is AND

If you search for more than one keyword at a time, Google/Bing will automatically search for pages that contain **ALL** of your keywords

- This is called "implicit AND"
- e.g.,
  - disney disneyland pirates = disney AND disneyland AND pirates
  - disney disneyland "pirates of the caribbean"
    ⇒ disney AND disneyland AND "pirates of the Caribbean"

Google Stemming and Stop Words

1. The query [child bicycle helmet] finds pages that contain words that are **similar** to some or all of your search terms,
   - e.g., "child," "children," or "children's," "bicycle," "bicycles," "bicycle's," "bicycling," or "bicyclists," and "helmet" or "helmets."
   - Google calls this feature ***word variations*** or ***automatic stemming***
2. Google will often ignore ***Stop Words***
   - However, a **query with only Stop Words**, e.g. [the who] gets treated as significant, returning pages for the Rock Group, the Who
3. Google limits queries to **32** words

4. Google favors results that have your search **terms near each other**
   - The query [snake grass] finds pages about plants;
   - The query [snake in the grass] finds pages about sneaky people
5. Google gives higher priority to pages that have the terms in the **same order** as in query
6. Google is **NOT** case sensitive; it shows both upper- and lowercase results
   - [Red Cross], [red cross], and [RED CROSS] return the same results.

7. Google **ignores** some **punctuation and special characters**, including !?, .; [ ] @ / #<>
   - Exceptions: C++, or math symbols in Google calculator

Boolean OR

The Boolean ***OR*** operator is acceptable in Google queries, placed between keywords, and **OR** is **always** in all caps

- e.g., disney disneyland OR "pirates of the caribbean"
  ⇒ disney AND the word (disneyland OR the phrase 'pirates of the caribbean')

OR has **higher** precedence than AND

How Google/Bing Treat Queries with Boolean Operators

All query terms are implicitly ***AND***ed
OR has **higher** precedence than ***AND***

- Three examples (**a, b, c** stand for query terms):
  1. **a b** *OR* **c d** is treated as **a** *AND* (**b** *OR* **c**) *AND* **d**
  2. **a** *OR* **b c** *OR* **d** is treated as (**a** *OR* **b**) *AND* (**c** *OR* **d**)
  3. **a** *OR* "**b c**" **d** is treated as (**a** *OR* ("**b c**")) *AND* **d**

Google Advanced Operators

Query modifiers
- **daterange:**
- **filetype:**
- **inanchor:**
- **intext:**
- **intitle:**
- **inurl:**
- **site:**

Alternative query types
- **cache:**
- **link:**
- **related:**
- **info:**

Other information needs
- **stocks:**

Search Service	Search Operators
Web Search	allinanchor:, allintext:, allintitle:, allinurl:, cache:, define:, filetype:, id:, inanchor:, info:, intext:, intitle:, inurl:, link:, related:, site:
Image Search	allintitle:, allinurl:, filetype:, inurl:, intitle:, site:
Groups	allintext:, allintitle:, author:, group:, insubject:, intext:, intitle:
Directory	allintext:, allintitle:, allinurl:, ext:, filetype:, intext:, intitle:, inurl:
News	allintext:, allintitle:, allinurl:, intext:, intitle:, inurl:, location:, source:
Product Search	allintext:, allintitle:

see also: link1, link2

- ***filetype*:**
  - restricts your results to files ending in a specific file suffix, e.g ".doc" (or .xls, .ppt. etc.), and shows you only files created with the corresponding program
  - There can be **no space** between `filetype:` and the file extension
  - The "dot" in the file extension -.doc - is optional
    - `filetype:doc` will **NOT** return `docx`
- ***inanchor:***
  - will restrict the results to pages containing the query terms you specify in the anchor text or links to the page.
  - e.g., [restaurants inanchor:gourmet]
    will return pages in which the anchor text on links to the pages contain the word "*gourmet*" and the page contains the word "*restaurants*."

- ***allinanchor***:
  - restricts results to pages containing **all query terms** you specify in the anchor text on links to the page.
  - e.g., [allinanchor: best museums sydney]
    will return pages in which the anchor text on links to the pages contain the words "*best*," "*museums*," and "*sydney*."
- ***intext:***
  - ignores link text, URLs, and titles, and only searches body text.
  - `intext:` helps you avoid query words that are too common in URLs and links.
    - pirates `intext:"Disney.com"` requires *Disney.com* to be within the body of the web page
- ***intitle:***
  - restricts the results to documents containing a particular word in its title.
  - You can also search for phrases. Just put your phrase in quotes
    - `intitle:pirates, intitle:"pirates of the caribbean"`
- ***inurl:***
  - restricts the results to documents containing a particular word in its URL.
    - `Inurl:disney`:
      • Results include: Disney.go.com, www.disney.de
- ***site:***
  - restricts the results to those **websites in a domain**.
  - There can be **no space** between site: and the domain.
    - `masters site:cs.usc.edu`

  • You can use `site:` in conjunction with another search *term* or *phrase*.
  pirates site:disney.com

  • You can also use `site:` and negation to **exclude** sites.
  pirates `-site:disney.com`

  • You can use `site:` to exclude or include entire top level ***domains*** (like with filetype, the dot is optional).
  `pirates -site:com`; `pirates site:edu`
- ***cache:***
  - url shows the version of a web page that Google has in its cache.

- ***link:***
  - restricts the results to those web pages that have links to the specified URL.
    e.g., `link: Disney.com`
  - Note: apparently the link operator only returns a sample of web pages pointing to the link
- ***related***:
  - lists web pages that are "similar" to a specified web page.
  - There can be **no** space between `related:` and the URL.
- ***info***:
  - presents some information that Google has about a particular web page.

Auto-completion

Auto-completion:

- is the process of predicting a word or phrase that the user wants to type in without the user actually typing it in completely

Auto-completion is a form of relevance feedback

- This feature is effective when it is easy to predict the word being typed, e.g. when a browser fills in your name, address and/or email in a form;
- Search engines may use past history, phonetic Soundex algorithms, and spelling corrections algorithms to assist in making guesses;
- The challenge is to search a large index or a long list of popular queries in a very short amount of time so the results pop up while the user is typing

Judging the Quality of Answers: Mean Reciprocal Rank (MRR) Scoring

Mean Reciprocal Rank (**MRR**)

- MRR is a statistical measure for evaluating any process that produces a list of possible responses to a sample of queries, ordered by probability of correctness.
- The **MRR** is the average of the reciprocal ranks of results for a sample of queries

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i}$$

- e.g., suppose we have the following three sample queries. In each case, the system makes three guesses, with the first one being the one it thinks is most likely correct:

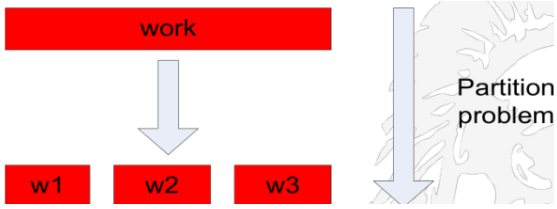
Query	Results	Correct response	Rank	Reciprocal rank
cat	catten, cati, **cats**	cats	3	1/3
tori	torii, **tori**, toruses	tori	2	1/2
virus	**viruses**, virii, viri	viruses	1	1

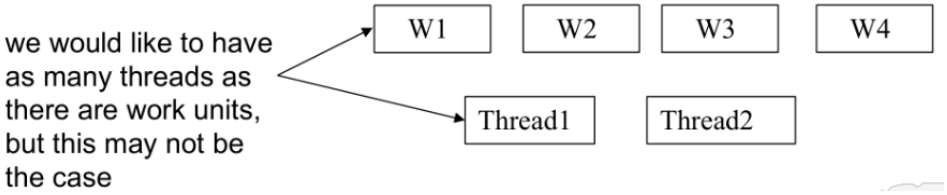
the mean reciprocal rank as **(1/3+1/2+1)/3 = 11/18** or about **0.61**.

Introduction to MapReduce

- **MapReduce** is a methodology for exploiting **parallelism in computing clouds** (racks of interconnected processors)
- It has become a common way to analyze very large amounts of data
- 背景 (为什么我们需要MapReduce & MapReduce做了什么):
  - 需要处理数量非常庞大的数据, 如果只用一台机器耗费的时间太久了
    - (e.g. Google每天可能要处理几十亿的搜索, 如果只有一台机器有的人等一辈子也等不 到结果)
  - MapReduce相当于把一个很大的任务分成很多小份, 每一份交给一台机器完成, 这一部分叫做 ***Map***. 在所有机器得到结果之后它们的结果会汇总到另外一些机器进行合并, 这一部分叫做 ***Reduce***

Why Parallelization is Hard

- **Parallelization** is "***easy***" if processing can be cleanly split into n units.
  - Assigned to n workers, And there is an easy way to combine the outputs
  - 

- Intention:
  - 

- Issues:
  - What if we have more work units than threads?
  - How do we assign work units to worker threads?
  - How do we aggregate/combine the results at the end?
  - How do we know all the workers have finished?
  - What if the work cannot be divided into completely separate tasks?

  → MapReduce solves all of these problems.

Multithreaded = Unpredictability

- When we run a multithreaded program, we don't know what order threads run in, nor do we know when they will interrupt one another.
- If the initial state is x = 6, y = 0, what are the final values of x and y after the threads finish running?
  Possible solutions include: (8,8) and (8,7)

```
Thread 1:          Thread 2:
void foo() {       void bar() {
  x++;               y++;
  y = x;             x++;
}                  }
```

Synchronization problems

- Processing Across a Machine Cluster Introduces Unpredictability on Many Levels
- Synchronization problems apply to more than just low level operations within a critical section of code
    - Other issues include:
        - Pulling work units from a queue
        - Assigning work units to an available thread
        - Work units reporting back to the master unit
        - Telling another thread that it can begin the "next phase" of processing
        - …

MapReduce in solving Parallelization Problems

- MapReduce provides
    - Automatic parallelization of code across multiple threads and across multiple processors
    - Fault tolerance in the event of failure of one or more nodes
    - I/O scheduling
    - Monitoring & Status updates

Map/Reduce

- MapReduce Model:
    1. A ***map function*** extracts some intelligence from raw data
    2. A ***shuffle step*** organizes the resulting output
    3. A ***reduce function*** aggregates the data output from the shuffle step
    - Users specify the computation in terms of a ***map*** and a ***reduce*** function,
    - Underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, and
    - Underlying system also handles machine failures, efficient communications, unicati and performance issues.

The Map/Reduce Paradigm

1. A large number of records are broken into segments
2. ***Map***: extracts something of interest from each segment
3. ***Group*** (= ***Shuffle***): sorts the intermediate results from each segment
4. ***Reduce***: aggregates intermediate results
5. Generate final output

- Key idea: to re-phrase problems in such a way that the input can be divided into parts and operated on in parallel and the results combined to produce a solution to the original problem

ex. 统计单词的例子

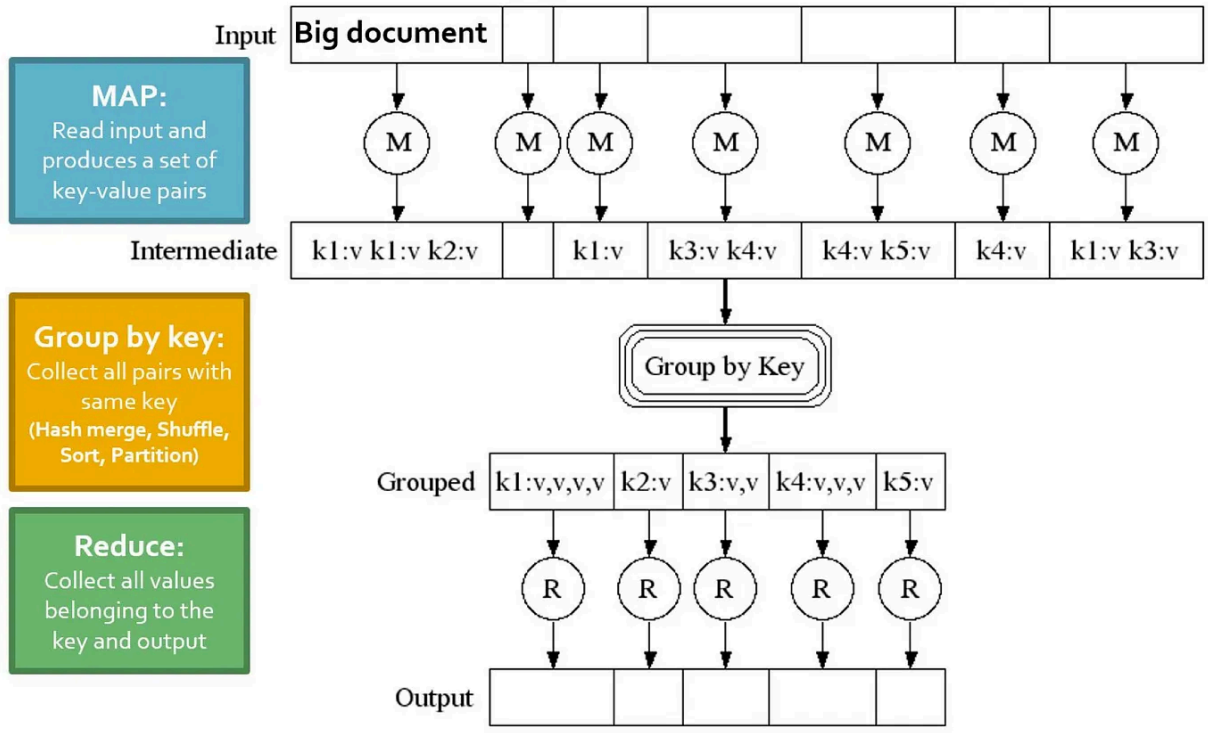
- 任务: 现在有三段话,我需要统计这段话里每个不同的单词及它出现的次数

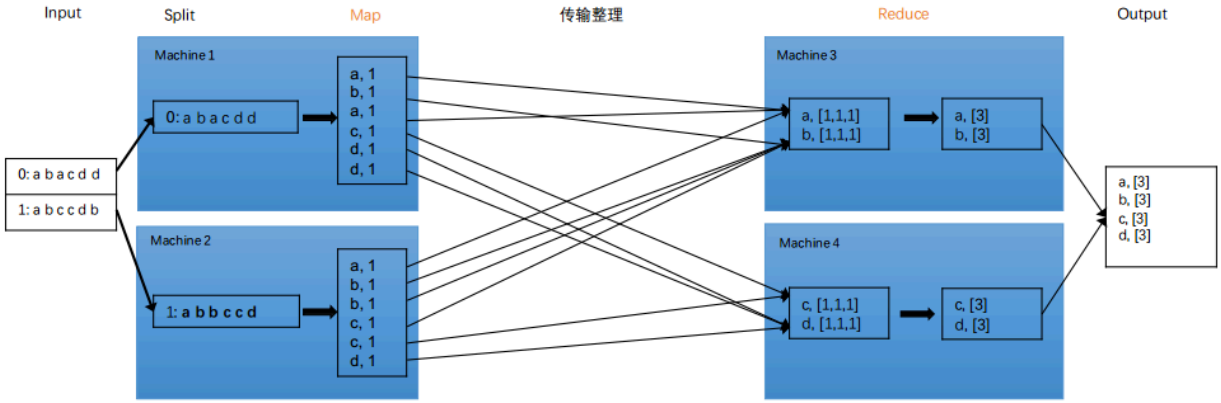
- 传统方式: 一个人从头到尾记录每个单词及它的出现次数

- ***Map***: 把三段话分成三份, 交由三个人来负责, 每个人负责处理其中的一段话. 每个人的任务是将每个单词及它在该段落出现的次数封装成(A, #of A)的形式.
- ***Shuffle(Group)***: 得到三个人分别统计之后的结果, 对它们的Key也就是(A, #of A)中的A进行hash, 根据hash的值决定将这个tuple分给哪个Reducer.
- ***Reduce***: 在Map之后每个人都会得到很多单词记录, 但是三段话里可能有些单词是重复出现的, 我需要最终得到合并以后的结果. 因此, 我又找了两个人帮我统计. Shuffle会决定将哪些tuple分给第一个人, 哪些tuple分给第二个人. 这两个人就根据tuple中的Key把它们对应的Value也就是#of A进行相加就得到了最终的结果.

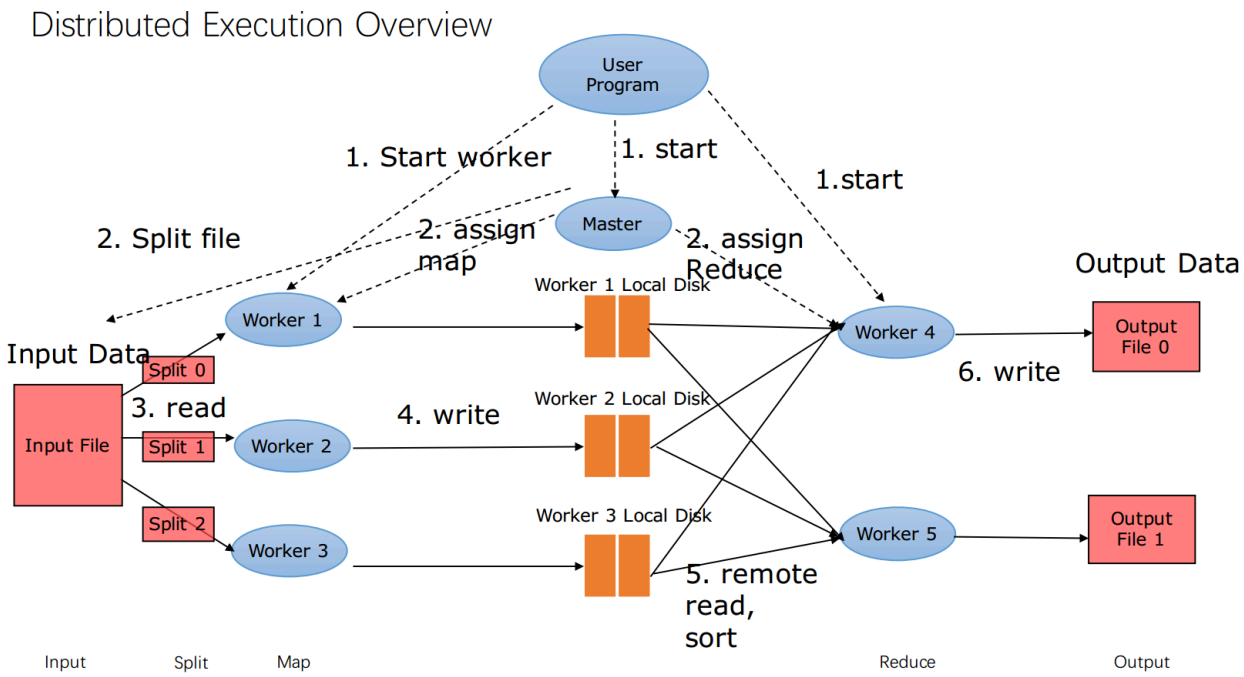
★ Note: **MapReduce** 的加速倍数符合Amdahl's law (即并行的机器数和提升速度成近似线性相关

e.g.用原来三倍的机器跑就可以比原来快接近三倍)





Distributed Execution Overview



步骤说明

1. input 文件 —— 存在哪里呢？disk? memory? gfs? —— 其实在gfs里
2. user写好了代码，配置多少个master, 多少个worker，谁做map，谁做reduce。然后把user的代码放到各个机器上。然后就启动了。
3. master找到输入文件，然后拆分，给各个worders，跑Map的部分
4. Map输出后，放入worker的硬盘里（为什么要放到disk? 能不能放到gfs呢？没有必要啊，没那么大）
5. 传输整理
6. Reduce拿到输入，开始reduce运算
7. 输出文件—— 一定要放在gfs里，很重要，需要replica，因此要在GFS里

- MapReduce之上的宏观概念，如何分配机器来进行Map和Reduce
- Master Node:
    - 一个Master可以控制若干个worker
    - Master可以分配给worker一个map任务或者一个reduce任务
    - Master要随时追踪每个worker的状态***(idle, executing, completed, crashed)***
    - Master知道所有map任务产生的文件的位置和大小，并且会把这些信息告诉对应的reducer

Q&A

1. Mapper和Reducer是同时工作还是先Mapper后Reducer？先Mapper后Reducer
2. 万一Mapper或者Reducer挂了咋办？重启？重选机器？ 重新分配一台机器做
3. Reducer一个机器key特别大咋办？（例如统计网站流量，网站url作为key,有的key的内容特别多，一个机器做不了）加一个random后缀，类似shard key，将同一个url分到不同机器上
4. input和outpu放哪里？ GFS
5. local disk上的mapper output data有没有必要放在GFS上，要丢了咋办？ 重做就行，不用GFS

6. mapper和reducer可以放在同一台机器吗 这样不是很好。因为mapper和reducer之前需要很多预处理工作，而如果放在两台机器，就可以做并行预处理

Map

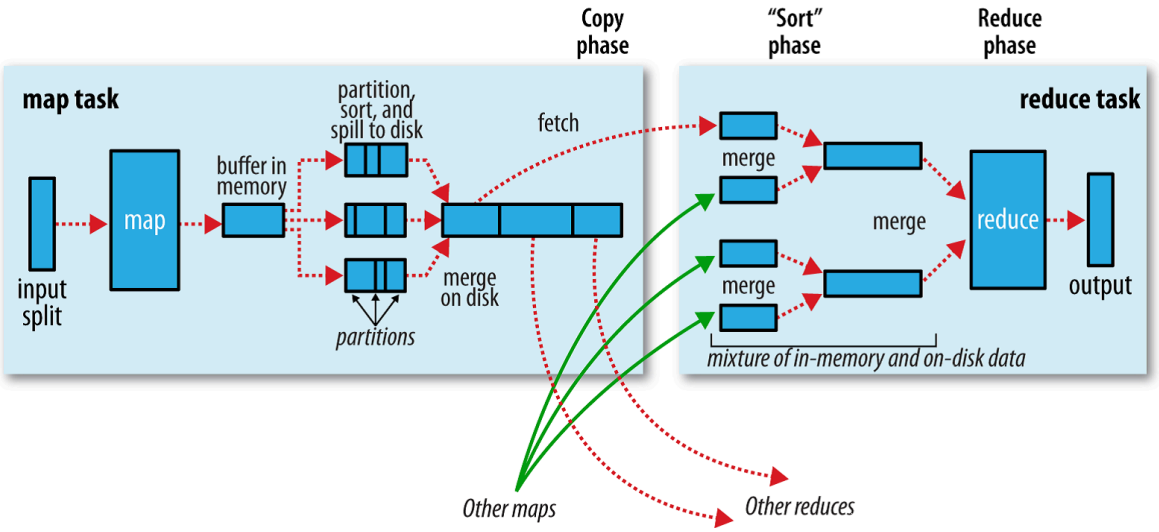
- Mapper负责"分"，即把复杂的任务分解为若干个"简单的任务"来处理. "简单的任务"包含三层含义:
    1. 数据或计算的规模相对原任务要大大缩小
    2. 就近计算原则，即任务会分配到存放着所需数据的节点上进行计算
    3. 这些小任务可以并行计算，彼此间几乎没有依赖关系 (不需要其他map的结果来完成自己的map工作)

Reduce

- Reducer负责"合"，即把所有Mapper得到的中间结果根据任务的不同需要合并成最终用户想看到的格式
    - (比如在(ex. 统计单词的例子)的🌰里reducer的主要工作就是对同一个单词在三个段落里出现的次数进行累加)

Shuffle

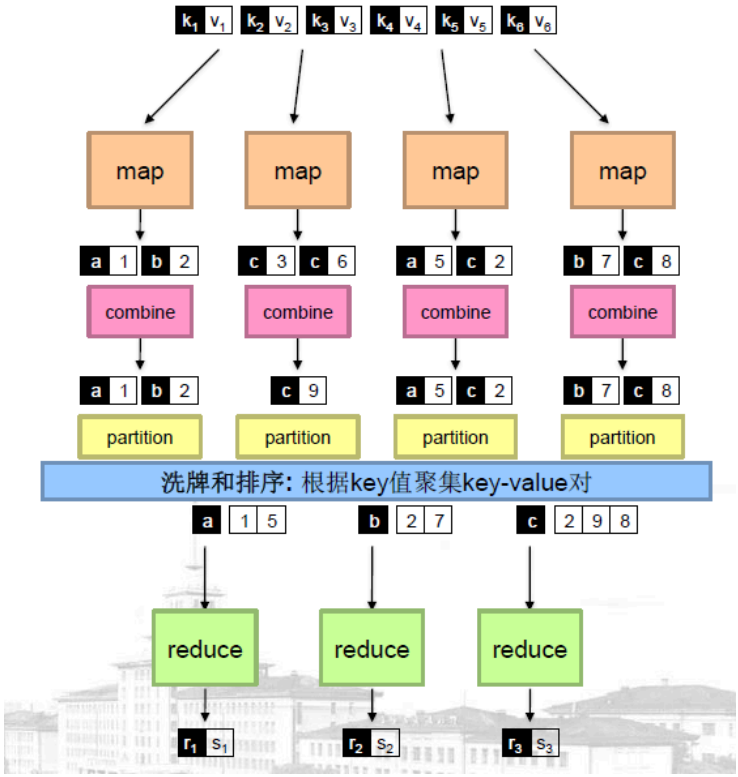
- Shuffle负责对中间结果进行排序，并且分发给对应的Reducer
- ★ 为什么要排序?
    - → 因为要group，排序(key一样的放在一起 and order)
      以后从头开始看，什么时候key跟上一个key不一样了
      什么时候就应该是新的group了
- Shuffle过程(下图): 这里的sort&merge用的是外部排序.
    - 首先Maps会各自得到一些(key,value)的对，然后根据hash分别将不同的key分开成不同的partitions各自排序. 比如在Map Task2中这一步的结果应该是{(k4,v), (k4,v), (k5,v)}, {(k3,v)}, 因为k4和k5是要给Reduce Task1的，而k3是要给Reduce Task2的
    - 之后Reduce Task会收到来自不同Map Task的包裹，包裹内各自是排好序的,因此Reduce Task只需要执行外部排序的merge部分就可以最终排好序的group了

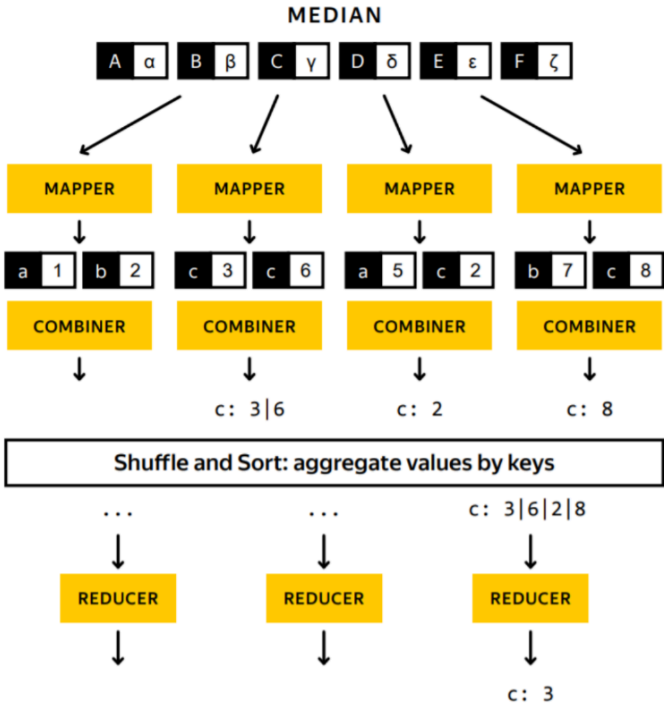


Combiner (Optional)

- 可以理解为Map Tasks处理完各自的任务之后先进行一个中间操作, 再给ReducerTasks; 这么做可以减小数据传输从而加快整体速度(但不是什么情况都可以用 combiner的,比如求平均就不可以)
- 以统计单词个数为例:假设有两个Map Tasks和一个Reduce Task。其中:
  - ex.
    - Mapper 1
      input: : [(0, “here”), (5, “here and there”)]
      output: [(“here”, 1), (“here”, 1), (“and”, 1), (“there”, 1)]
    - Mapper 2
      input: : [(0, “here or there”), (14, “there”)]
      output: [(“here”, 1), (“or”, 1), (“there”, 1), (“there”, 1)]
  - 经过combiner的处理以后:
    - *Combiner 1*: **[(“here”, 2), (“and”, 1), (“there”, 1)]**
    - *Combiner 2:* **[(“here”, 1), (“or”, 1), (“there”, 2)]**
  - 可以看出来这里***combiners***先统计了各自的答案,再给Reducer
  - 最后Reducer整理出的结果:**[(“here”, 3), (“and”, 1), (“or”, 1), (“there”, 3)]**

MapReduce过程图解 (One more example)





- ★ Note: 在整个Web Search Engine中用户**Search**的过程和Engine内部**Create Inverted index**的过程都需要**MapReduce** (每个**block**包含一部分**documents**)

Fault Tolerance in MapReduce

1. If a task crashes:

- Retry on another node
  - OK for a map because it had no dependencies
  - OK for reduce because map outputs are on disk
- If the same task repeatedly fails, fail the job or ignore that input block

2. If a node crashes:

- Relaunch its current tasks on other nodes
- Relaunch any maps the node previously ran
  - Necessary because their output files were lost along with the crashed node

3. If a task is going slowly (straggler):

- Launch second copy of task on another node
- Take the output of whichever copy finishes first, and kill the other one

4. The compute node of a Map worker fails

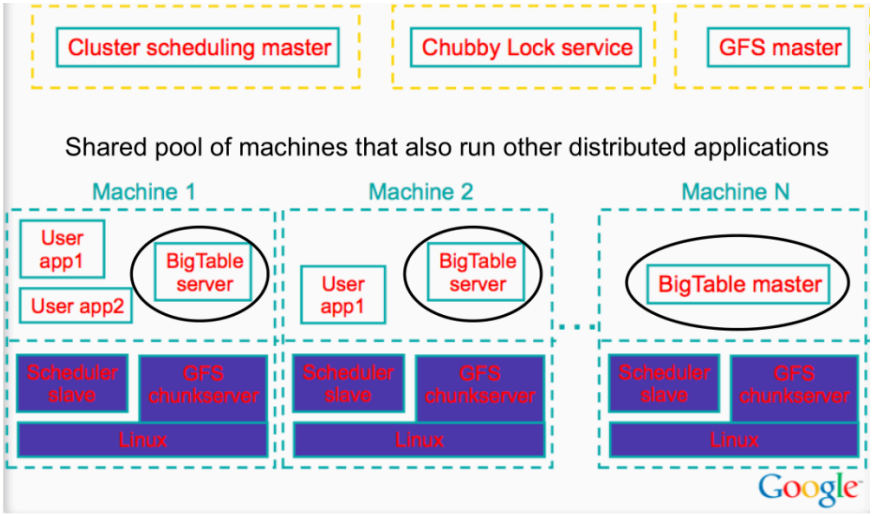
- This is detected by the Master and all Map tasks that were assigned are re-done
- The Master sets the status of each Map task to idle and re-schedules them when a worker becomes available
- The Master informs each Reduce task of the location of its new input

5. The compute node of a Reduce worker fails

- The Master sets the status of its currently executing Reduce tasks to idle and they will be re-scheduled on another reduce worker later

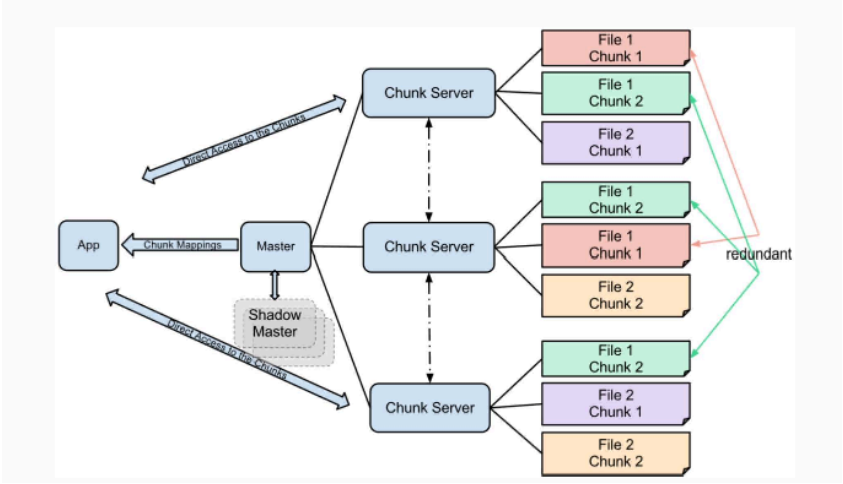
1. 如果是一个Task crashed了:
   - 在另一个node上重新尝试这个task
     - 对Map任务是可行的因为Map是MapReduce的第一步, 没有需要读入的东西
     - 对Reduce任务是可行的因为Reduce需要的是Map的输出, 而Map的输出是存在disk里的
   - 如果一个task在换过多次node之后还是fail了, 忽略这个chunk
2. 如果一个Node crashed了:
   - 把当前分配给这个node的所有tasks分配给其他nodes
   - 把之前在这个node上跑完的所有Map任务在其他nodes上重新跑一遍
     - (因为Map跑完的结果都会存在node上, 如果node挂了那这些输出也都丢失了)
3. 如果一个task跑的非常慢:
   - 把这个task的一个copy放在另一个node同时跑
   - 哪一个node先跑完就取那一个node的结果, 并且kill掉还没跑完的那个
4. 如果一个Map worker node crashed了:
   - 会被Master发现并且所有被分配给这个worker的map任务都要重新分配给其他map worker重新做一遍
   - Master会把分配给这个worker的map任务的状态都设置成idle, 当某个worker的状态成为available了就分配map任务给它
   - Master会将新的map结果的地址位置发给Reduce worker
5. 如果一个Reduce worker node crashed了:
   - Master会把分配给这个worker的reduce任务的状态都设置成idle, 当某个worker的状态成为available了就分配reduce任务给它

Typical Cluster Machine Configuration



GFS (Google File System)

- 问题: Google要存的东西实在是太多了, 即使用了MapReduce, Master node也存不下map worker生成的结果
- **Solution**: 再加一层**chunk node (chunk server => team leader (a little master))**

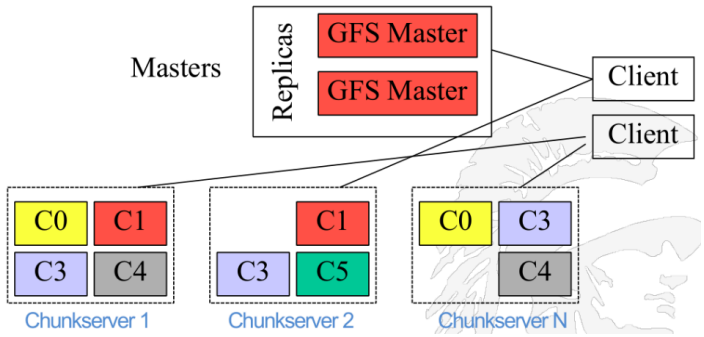


-

Master Server and Chunk Servers

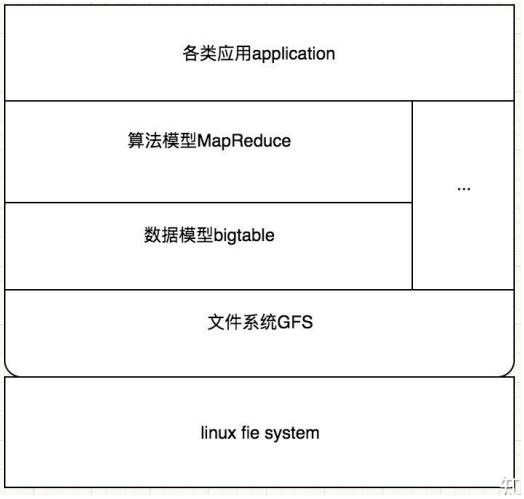
- **Master Server**
  - Master Server holds all metadata:
    - Namespace (directory hierarchy)
    - Access Control information (per-file)
    - Mapping from files to chunks
    - Current locations of chunks (chunk servers)

    - Delegates **consistency** management
    - **Garbage collects** orphaned chunks
    - **Migrates chunks** between chunk servers
- **Chunk Server**
    - Stores 64 MB file chunks on **local disk**
      using standard Linux filesystem, each with version number and checksum
    - Read/write requests specify **chunk handle and byte range**
    - Chunks **replicated** on configurable number of chunkservers (default: 3)
    - No caching of file data (beyond standard Linux buffer cache)



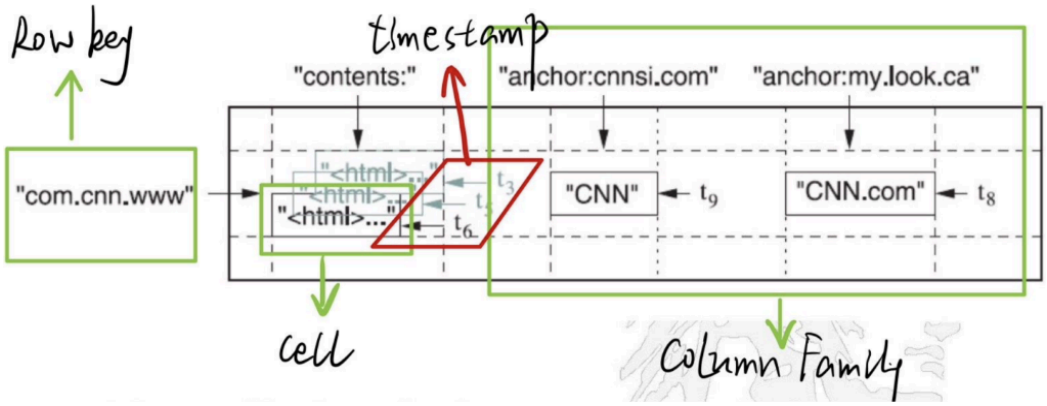
- Master manages metadata
- **Data transfers happen directly between clients/chunk servers**
- Files broken into chunks (typically 64 MB)
- Chunks triplicated across three machines for safety

BigTable Data Model



- BigTable 是一种压缩的, 高性能的, 高可扩展性的, 基于GFS的数据存储系统, 用于存储大规模结构化数据, 适用于云端计算. BigTable中的row存储方式是(key,value)
- BigTable中的每一个table具有的特点:
    - Sparse
    - Distributed
    - Persistent
    - **Multidimensional (various time and versions)**
    - Sorted map

- BigTable中的要素:
    - ***Row key***: 即行关键字, 存的是比如学生id之类的.
        - 是排好序的, 排序方式可以自定义
    - ***Column Family***: 把一些相关的列整合成一个family. BigTable也可以根据 column检索 (比如一列存的是学生的GPA, 就可以直接把这一列取出来求平均而不需要一行一行访问)
    - ***Cell***: 表格里的每一个数据被称为一个Cell
    - ***Timestamp***: 这里体现了12.11.2.4中的BigTable的Multidimensional的特点. 通常我们存的表格都是二维的, 但是在BigTable中还可以存同一个cell在不同时间的数据 (比如一个学生在不同学期的GPA)


Row key
timestamp
"contents:"
"anchor:cnnsi.com"
"anchor:my.look.ca"
"com.cnn.www"
"<html>..."
"<html>..."
"<html>..."
t3
t5
t6
"CNN"
t9
"CNN.com"
t8
cell
Column Family

Taxonomy

- Definition: A **classification** or **categorization** of a complex system
- 特点:
  - 通常以 **tree** 的结构呈现 (e.g. CS下面有不同的major, 每个major又有自己的不同topic)
  - 遵循 **Inheritance**, 即可以把Taxonomy理解成父类与子类之间的继承关系 (特点是这里的父类的所有**property**都是public的而没有private的, 即子类会继承父类的所有 **property**)
  - 主要应用于knowledge management, information retrieve等领域

Ontology

- Definition: A set of concepts and categories in a subject area or domain that shows their **properties** and the **relations** between them.
- 特点:
  - 突出的是entity之间的relationship (e.g. 同义关系、相反关系、包含关系), 目的在于给AI提供一种人类知识的表示方式
  - 通常以directed, labeled, cyclic graph的结构呈现
  - 主要应用于AI领域

Knowledgebase

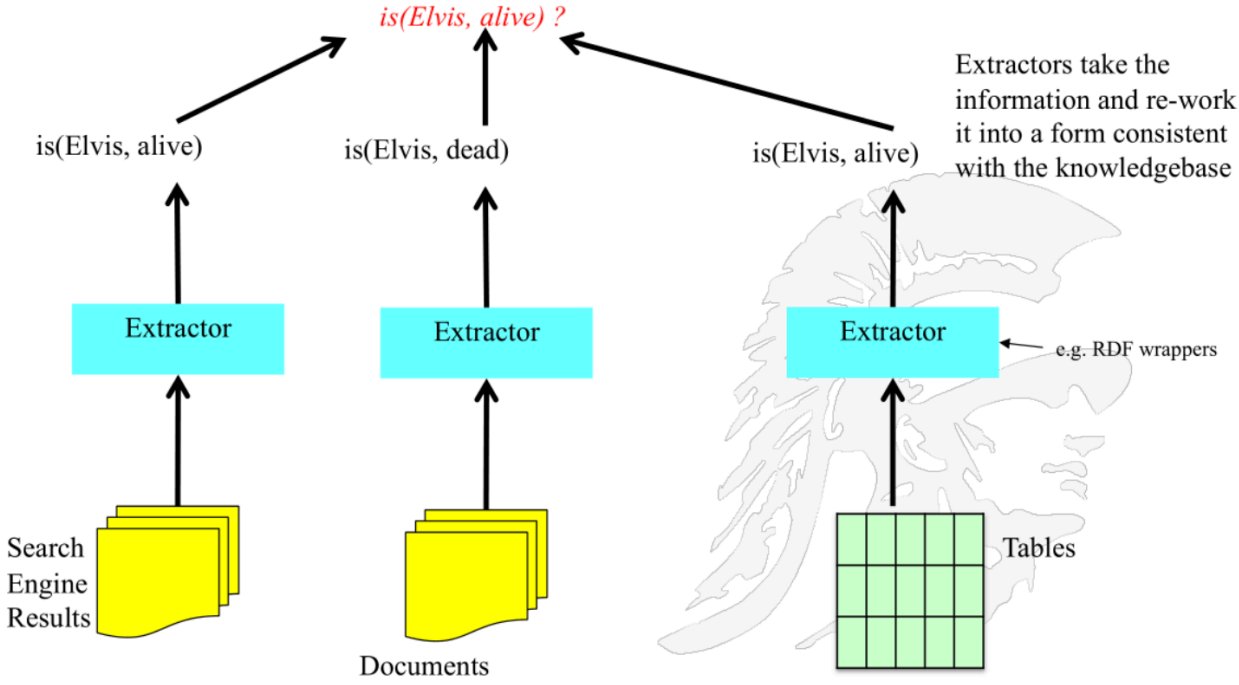
- Definition: A **knowledgebase (KB)** is a technology used to store and retrieve complex structured and unstructured information as stored in an ontology
  - 一种用于store和retrieve在Ontology的数据和关系的计算机系统
- Two components:
  1. a way of **representing** information and
  2. a method for **reasoning** about the information
- A **knowledgebase** is a collection of information stored in an ontology that includes software used to author and present that information
- 特点:
  - **Representing**: 能够表达在ontology中每个的entity的property及information
  - **Reasoning**: 能分析ontology中不同的entity之间的relationship
- Knowledge-based system: 一般用于搜索引擎的AI, 由两个elements组成:
  - **Knowledgebase**: 能存储和表达某个entity的信息
  - **Inference engine:** 能通过某些逻辑推导出entity之间可能存在的关系 (e.g. 西雅图经常下雨 → 防滑轮胎在西雅图的销量比普通轮胎更高; 用户搜索毕加索, 推荐给用户毕加索的画作)

RDF Data Model

- RDF allows us to make statements about resources.
  - The format of these statements is:
    `<subject> <predicate> <object>`
  - ex.

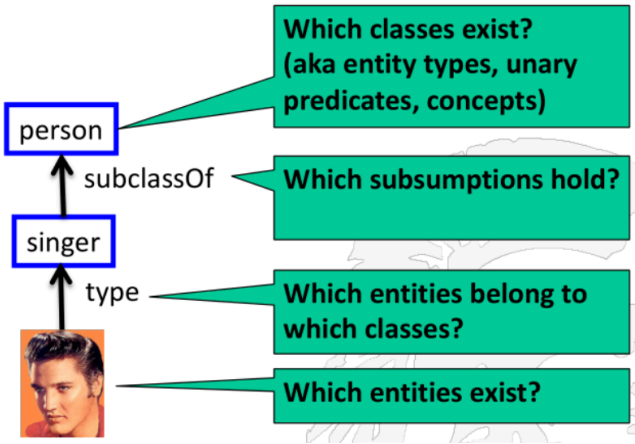
- <Bob> <is a> <person>
- <Bob> <is a friend of> <Alice>
- <Bob> <is born on> <the 4th of July 1990>
- <Bob> <is interested in> <the Mona Lisa>

★ Note: 在回答一个问题的时候knowledgebase会综合多种信息来源总结出答案



Find Classes and Instances.

- To Build a Knowledgebase One Must Find Classes and Instances.

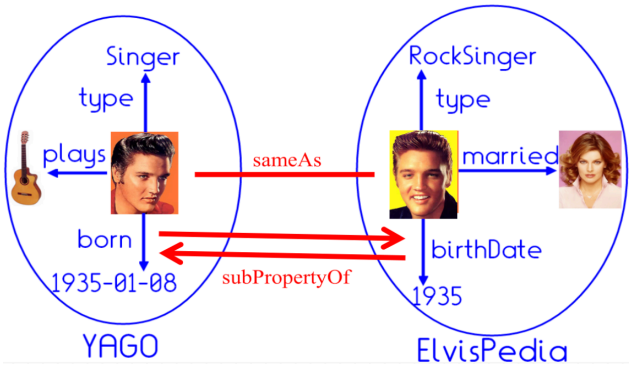


Multiple Ontologies

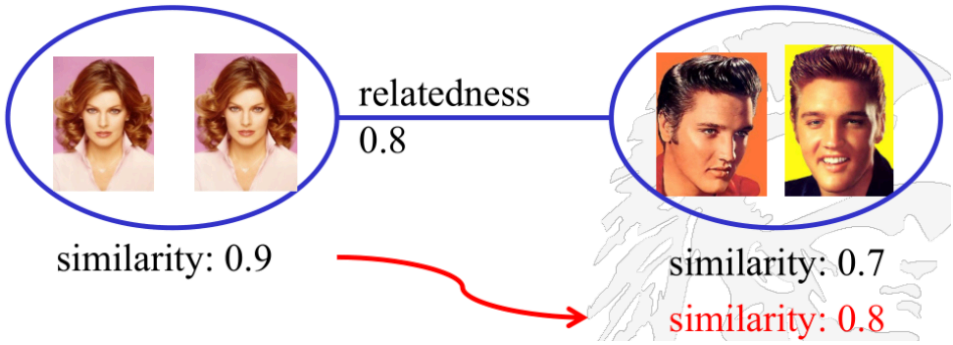
- A Knowledgebase Must Work Across Multiple Ontologies
- 如果有多个*Ontology*, 一个*Knowledgebase*该如何将它们整合在一起呢?
  - → 寻找不同的**Ontology**之间等价的**Entity**

- 🌰: 同一种病在全球不同国家不同地区可能有很多不同的叫法, 并且有各自的治疗方法, 它们构成了很多不同的Ontology. 这些病的不同名字就构成了等价的**Entity**, 连接这些等价Entity就能把所有治疗这个病的方法整合进一个knowledgebase.

★ 如何判断哪些*Entity*是等价的呢? → 通过它们共有的 property 计算相似性

- We Need to Match Entities, Classes and Relations
  - 

- Combining Elements From Different Knowledgebases Means Matching Entities
  - Build a graph:
    - nodes: pairs of entities, weighted with similarity
    - edges: weighted with degree of relatedness



- Iterate until convergence:
  - similarity := weighted sum of neighbor similarities

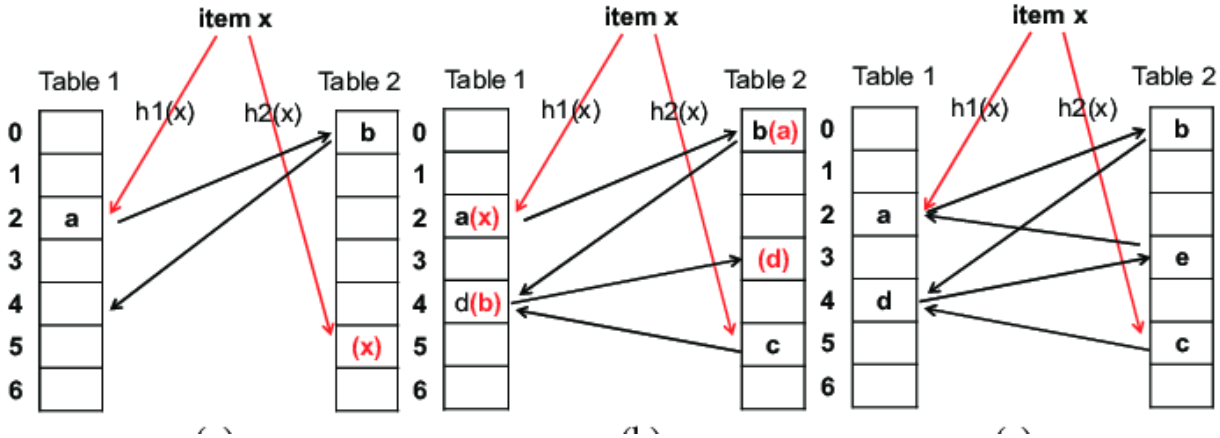
Inference engine

- **Inference engine**: a component of a system that applies logical rules to a knowledgebase to deduce new information.
- Two modes: Forward chaining, Backward chaining
- **Forward chaining:**
  - starts with the known facts and asserts **new facts**.
  - 通过现有的facts推测出新的**facts** (即当前knowledgebase不存在的facts)
  - Forward chaining is the repeated application of ***modus ponens***
    - 表示形式:

        - "P implies Q" and "P" are both asserted to be true, so therefore Q must be true." ⇒ Also: ((P -> Q) and P) -> Q
    - 🌰: “If today is Tuesday, I have a class”, “Today is Tuesday” → “I have a class”
  - 搜索引擎绝大多数时候使用 forward chaining
- **Backward chaining:**
  - starts with goals, and works backward to determine what facts must be asserted so that the goals can be achieved
  - 给定一个目标, 推测出要让这个目标发生需要有的facts
    (要比**forward chaining**困难)
    - eg. 考试, 给定一个需要达成的目标, 求需要哪些技术和 facts实现这个目标

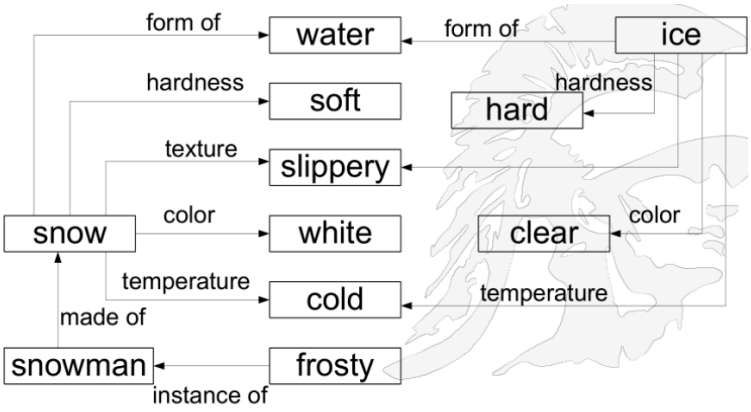
★ Note: **Cuckoo Hashing**:

- (Tiktok用于高速缓存和查找的算法).
- 它通过两个不同的哈希函数来避免哈希冲突, 并通过迭代方式来解决任何冲突.
- 基本思想是将一个键值对映射到两个哈希表位置中的一个, 如果其中一个位置已经被占用, 则将该键值对放置在另一个位置, 如果另一个位置也被占用, 则将原位置上的键值对移到它的另一个位置, 依此类推, 直到所有键值对都能够被插入到哈希表中.
- 如果无法插入, 则需要重新哈希或扩展哈希表大小, 以容纳更多的键值对.
- Cuckoo hashing的优点是平均插入和查找的时间复杂度为 ***O(1)***,而且不需要解决链式哈希表中可能出现的长链问题.
- 但是, Cuckoo hashing的缺点是需要占用两倍的内存空间, 而且对于某些哈希函数和数据集, 可能会出现插入失败的情况, 需要进行重哈希或扩展哈希表大小.



Semantic Network

- A semantic network is a knowledge representation scheme that represents knowledge as a graph.
  - The nodes of the graph correspond to concepts or facts and
  - the arc correspond to relations or associations between concepts.
  - In a semantic network both the nodes and arcs are labeled
- A semantic network that defines the properties of snow and ice
- The concept snowman inherits all the properties of snow

- The concept of ice and snow share a number of properties
  - 

PageRank

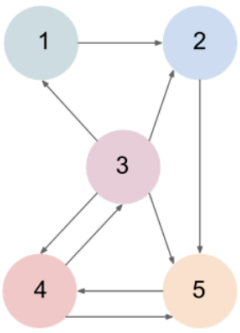
- **Random Surfer Model:** 从网络中随机一个 **page** 开始,随机选择一个link点进去。
  一个页面被点击到的次数越多它就越重要

- 特点:
  - 本质上是一个 **vote** 模型, 通过其他 pages 的投票表示自己的重要性
  - 只关心 link, 不关心 **page** 内容. 只有 in-link 能给 page 带来重要性
  - 可以看作一个 **probability distribution**,
    用于形容一个人随机点 **link** 能到某个 **page** 的概率

Simplified PageRank algorithm

- If page j with importance r_j has **N** out-links, each link gets votes r_j/N
  (把自己的重要性分成N份给自己的目标)
- Page j's own importance is the **sum of the votes on its in-links**
- 可以描述为 $r_j = \sum_{i \to j} \frac{r_i}{d_i}$, 其中 d_i 表示点 i 的 **out-degree**
- 流程:
  1. 初始化 page 的 rank 为 **1/n**, 其中 **n** 为page总数
  2. 对每个点按照 $r_j = \sum_{i \to j} \frac{r_i}{d_i}$ 的公式计算新的rank值.
  3. 重复 2. 直到网络拟合 (前后两个iteration每个page的rank都没什么变化)

★ Example:



	Iteration 0	Iteration 1
P_1	1/5	1/20
P_2	1/5	5/20
P_3	1/5	1/10
P_4	1/5	5/20
P_5	1/5	7/20

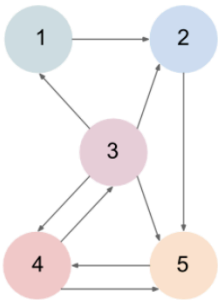
$PR(P_5) = 1/5 + 1/5 * 1/4 + 1/5 * 1/2 = 7/20$

1. Iteration 0: Initialize all pages to have rank ⅕.
2. Iteration 1:
3. **P_1:** has 1 link from P_3, and P_3 has 4 outbound links, so we take the rank of P_3 from iteration 0 and divide it by 4, which results in rank (⅕)/4 = 1/20 for P1

$PR(P_1) = (1/5)/4 = 1/20$

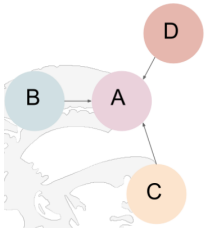
4. **P_2:** has 2 links from P_1 and P_3, P_1 has 1 outbound link and P_3 has 4 outbound links, so we take (the rank of P_1 from iteration 0 and divide it by 1) and add that to (the rank of P_3 from iteration 0 and divided that by 4) to get ⅕ + 1/20 = 5/20 for P_2

$PR(P_2) = 1/5 + (1/5)/4 = 5/20$



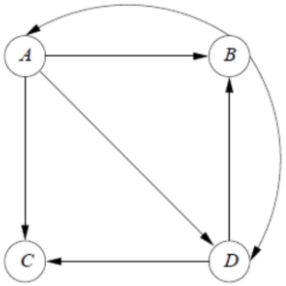
	Iteration 0	Iteration 1	Iteration 2	Final Ranking
P_1	1/5	1/20	1/40	5
P_2	1/5	5/20	3/40	4
P_3	1/5	1/10	5/40	3
P_4	1/5	5/20	15/40	2
P_5	1/5	7/20	16/40	1

- 第三次 iterrarion 时收敛（可以rank）

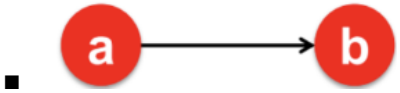


Extreme Case

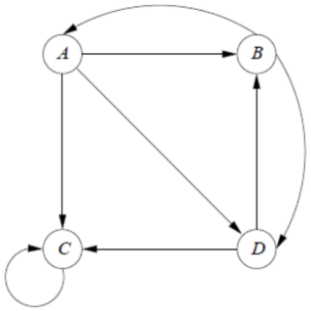
- Initial: PR(A) = PR(B) = PR(C) = PR(D) = **1/4**
- Iterrarion:
  - **PR(A)** = PR(B) + PR(C) + PR(D) = **0.75**
  - BUT ❗: **PR(B) = PR(C) = PR(D) = 0**



Complete PageRank algorithm (用于解决Extreme Cases)

- **Dead End (a page with no edges out)**
  - **Absorb PageRanks**
  - **PageRank => 0** for **any page that can reach the dead end (including the dead end itself)**
  - 

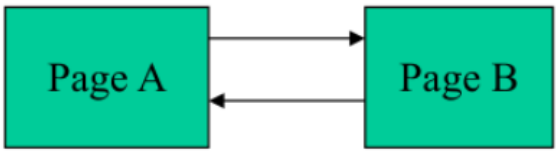
- Spider Trap (**Group** of pages with no edges going out of group)
  - Absorb all PageRanks (rank of C =>1, others =>0)
    - Surfer can never leave, once trapped
    - Can have > 1 such trap nodes (group的大小可能>1)



- 解决方法: **Taxation** (给每个 page 添加一个概率 β, 表示在跳转的时候有(1 − β)的概率会随机跳转到任意某个其他 page)
- 改进后的公式: $r_j = (1 - \beta) + \beta \sum_{i \to j} \frac{r_i}{d_i}$ 通常 β 是介于0.8至0.9的数

([计算例](#))Computing PageRank by Iteration Example

- Consider 2 pages: Page A and Page B pointing to each other.



- C(A) = 1, number of outgoing links of A
- C(B) = 1, number of outgoing links of B
- What should be the initial value of P(A) or P(B) ?

ex.1

- **Suppose the initial values are :**
  - **P(A) = 1 and P(B) = 1 and d= 0.85; then**

    PR(A)= (1 – d) + d*(PR(B)/1)

    PR(B)= (1 – d) + d*(PR(A)/1)

i.e.

- **PR(A)= 0.15 + 0.85 * 1**
  = 1
- **PR(B)= 0.15 + 0.85 * 1**
  = 1
- **In one iteration we are done**
- **Let's try another set of initial values.**

Ex.2

- **Initial Values : P(A) = 0, P(B) = 0 and d= 0.85**

PR(A)= (1 – d) + d(PR(B)/1)
PR(B)= (1 – d) + d(PR(A)/1)

- **PR(A)= 0.15 + 0.85 * 0 = 0.15**
  PR(B)= 0.15 + 0.85 * 0.15 = 0.2775

Iterating again we get:

- **PR(A)= 0.15 + 0.85 * 0.2775 = 0.385875**
  PR(B)= 0.15 + 0.85 * 0.385875 = 0.47799375

And iterating again

- **PR(A)= 0.15 + 0.85 * 0.47799375 = 0.5562946875**
  PR(B)= 0.15 + 0.85 * 0.5562946875 = 0.62285048437

- **After 20 iterations…**
- **PR(A) = 0.99**
- **PR(B) = 0.99**
- **Both approaching to 1.**

Suggestions for improving PageRank

- **Increasing the internal link in your site** (site内的in-link越多整个网站就越稳定, 你指向别的external page的时候分享出去的rank就越少)
- **Use a hierarchical structure to highlight the root page** (可以理解为用tree的结构去维护一个site, 这样main page会获得最高的Rank)
- 理论上不指向任何 **external page** 会给你带来最高的 **Rank**, 但是不推荐这么做 (如果所有人都这么做, 网络将被分割成无数个小的block而不会互相连接)

Snippets

- 定义: 指搜索结果页面中的摘要或简介
  - Automatic summarization by computer is a traditional subject of information retrieval
- TWO general approaches to automatic summarization:
  - ***extraction***:
    - *Extractive* methods work by selecting a subset of existing words, phrases, or sentences in the original text to form the summary
  - ***abstraction***:
    - *Abstractive* methods build an internal semantic representation and then use natural language generation techniques to create a summary that is closer to what a human might express
  - Research to date has focused primarily on extractive methods, which are appropriate for documents, images, and videos

Featured Snippets

- 定义: Feature Snippet通常位于搜索结果的顶部, 以突出显示与用户搜索意图相关的答案. 它们通常会直接回答用户的问题, 而不需要用户进入网页查找答案
- 类型: **Paragraph, List, Table**
- 内容来源: page的metadata; page的内容

Snippets Generation算法:

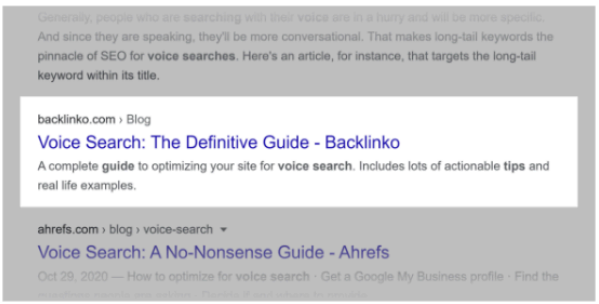
- 流程:
  1. 找到包含query terms的paragraphs
  2. 对这些paragraphs进行打分, 找出分数最高的那一个paragraph
  3. 返回这个paragraph中包含query terms的一小段话
  4. 如果这个document有abstract或者conclusion, 可以直接选取这一部分作为 3. 中需要的paragraph

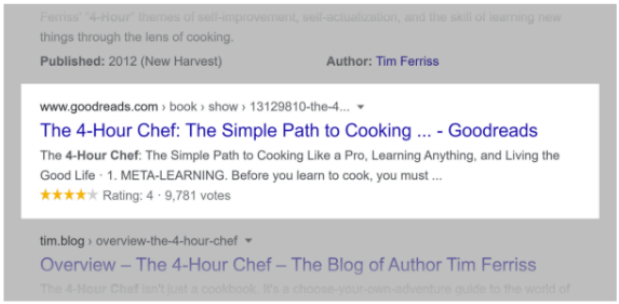
- 流程:
  1. 首先长度小于一定阈值的paragraph就是0分
  2. 剩下的paragraph的公式: $Score_{k-th} = \beta_k + max(APL, MPL)$, 其中$Score_{k-th}$表示第k段paragraph的分数, β_k 表示位置参数, *APL*(Actual Paragraph Length)表示当前paragraph的长度, *MPL*(Max Paragraph Length)表示所有paragraphs中的最长长度

Rich Snippets

- Rich Snippets are normal Google search results with additional data displayed. This extra data is usually pulled from Structured Data found in a page's HTML.
  - (简单来说就是在写网页的时候HTML里有一些tag可以添加为rich snippets)
  - Common Rich Snippet types include ***reviews***, ***recipes*** and ***events***.

★ **Example:** Without and with Rich Snippets:





- 优点:
  - 对于webmasters:
    - Provides webmasters the ability to add useful information to their web search result snippets to help Google make sense of their bits.
  - 对于users:
    - Provides more information to a user about the content that exists on page so they can decide which result is more relevant for their query.
- Two Good Reasons for using rich snippets:
  - 对于Google:
    - Provides **additional traffic to a webpage**
    - **higher click through rate**
  - **Easy to add.** Just simple lines of existing HTML.

Summary

Snippets can be divided into **five** categories

1. ***Regular snippets*****,** displayed in organic search results
2. ***Rich snippets*** come from structured data dictionary schema.org including RDFa, Microdata or JSON
3. ***Google News*****,** created automatically from news feeds to Google
4. ***Entity types,*** come from the KnowledgeGraph, are constructed object and concepts including people, movies, places, events, books, etc
5. ***Features snippets,*** determine that a page contains a likely answer to the user's question; the snippet is displayed. In four different forms: paragraph, table, ordered list, unordered list

Auto correction两个主要任务

- Spelling **Error Detection**:
  - need a big dictionary; using context may be necessary
- Spelling **Error Correction:**
  - Web search engines ***always*** try to suggest a correction
  - Two major techniques:
    - edit distance algorithm
    - *n-gram* matching

Three Types of Spelling Errors

1. Non-word errors:
   - e.g. *graffe* → *giraffe*
2. Typographical errors:
   - e.g. *three* → *there (***especially bad as both are legal words***)*
3. Cognitive errors:
   - e.g. *your* → *you're*

Basic Spelling Correction Algorithm的步骤

1. Initial Step: Create a dictionary and encode it for fast retrieval
2. When a query is submitted, the spell checker examines each word and for words not in the dictionary looks for possible character edits, namely
   - 当接收到一个query时, 检查这个query中每个 **word** 以及 **n-gram** 是否在dictionary里.如果是则判断不需要correction; 如果不是则找possible character edits
     (这里用**Levenshtein algorithm**来算距离)

★ Note: n-gram在这里的存储结构是字典树 **(tire)**, 查找复杂度为**O(k)**, k为query长度

★ Levenshtein algorithm: cf. leetcode72

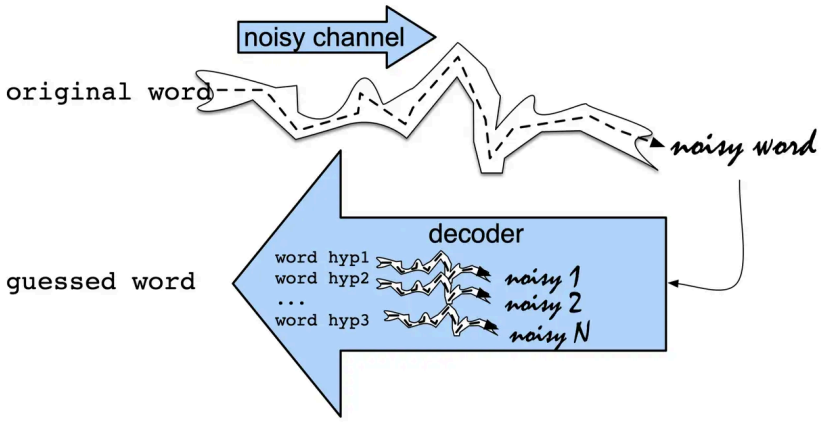
- 计算两个word之间的edit distance, 其中有三个操作
  - ***insertion, deletion, change***
  - 每个操作的距离为1 (e.g. sitten到setting的距离为3)
- Levenshtein algorithm实际上是一个动态规划,
  - 假设一个函数edit(i, j), 它表示第一个字符串的长度为i的子串到第二个字符串的长度为j的子串的编辑距离
  - 迁移方程:
  -

$$edit(i,j) = \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0 \\ j, & \text{if } i = 0 \text{ and } j > 0 \\ i, & \text{if } i > 0 \text{ and } j = 0 \\ \min\{ & edit(i-1,j)+1, \\ & edit(i,j-1)+1, \\ & edit(i-1,j-1)+f(i,j)\}, \\ & \text{if } i \geq 1 \text{ and } j \geq 1 \end{cases}$$

        - 其中,当第一个字符串的第i个字符不等于第二个字符串的第j个字符时, f(i, j) = 1; 否则,f(i, j) = 0
    - 为edit distance增加权重: 为什么?
        → 现实世界中某些字母之间更容易被拼错(e.g. A和E).
        建立一个混淆矩阵(confusion matrix)来记录不同的字母之间被互相混淆的次数, 这个次数就可以作为考量correction时的一个权重.
3. 通过Levenshtein algorithm算出距离错误 **term**为 ***k(k=1 or 2)*** 的字典里的 **term** 作为替换掉错误的 term 作为 candidates. 这样 candidates 会是若干个新的 ***n-gram***
4. 用 **Noisy Channel Model** 计算出每个 candidate 是用户想搜的内容的 probability, 选取最高的那个进行搜索代替用户原来的 query

★ Noisy Channel Model:

- 设计理念为, 拼写错误是由正确的文本经过 “Noisy Channel”被混淆的结果. 通过建模这个通道, 可以逐词排查, 寻找最接近的单词. 这种有噪通道模型属于贝叶斯推断(Bayesian inference), 因此概率分布遵循贝叶斯原则.



概率公式:

$$P(\text{candidate word} \mid \text{received word}) = \frac{P(\text{received word} \mid \text{candidate word}) \times P(\text{candidate word})}{P(\text{received word})}$$

- 其中,P(candidate word | received word) 表示在收到word后, 正确的word为candidate word的概率

Auto-completion

17.4.1. 在 Correction Algorithm的步骤 2 中我们讲过存储n-gram的结构是tire, 这也是auto-complete的思路. 匹配到某个candidate的时间为O(k), k为candidate的n-gram的长度. 然后再按照 probability 从高到低显示给用户

Query processing

- how does the search engine respond?

Query processing

- 目的: Speeding up Indexed Retrieval
- Search Engine's Task:
    1. **Minimally return documents that contain the query terms**
    2. **Top-ranking documents should be both relevant and authoritative**
    3. **Determine what the user is actually trying to accomplish, even though the query may be vaguely stated**

Speeding up step 1

Strategy1: Consider only query terms with high-idf scores

- ex.
    - For a query such as *'catcher in the rye'*:
    - 只累加 '*catcher*' 和 '*rye*' 的 cosine 分数, 因为 *'in'* 和 *'the'* 这两个词的idf很低, 即在绝大多数文档都有出现, 他们对整体排名的贡献度很低

Strategy2: Consider only docs containing several query terms

- **Only accumulate (cosine) scores for docs containing several of the query terms**
- ex,
    - 假设query是 "Spatial Data Analysis":
    - 理论上我们会找包含 "Spatial"或 "Data"或 "Analysis"或 "Spatial Data"...的docs并且计算cosine分数然后排序返回给用户. 但是我们现在只找包含长度为 2 和 3 的term的docs (only compute cosine scores for docs containing several of the query terms)

- 好处:
    - 不用考虑很短的term, 这些term往往对应很多docs, 大大加快了速度;
    - 同时easy to implement in postings traversal

Strategy3: Introduce Champion Lists Heuristic

- 方法: Pre-compute for each dictionary term ***t***, the ***r*** docs of highest tf-idf in ***t***'s postings
    - (相当于对语料库里的每个 term 预处理出一个 posting list, 这个 list 里只存 tf-idf 最高的 *r* 个文档)
    - 这个list也叫 **champion list**

  ★ Note: ***r*** 有可能比 ***K*** (返回给用户的文档数)小
    - (因为 ***r*** 是在建立 **index** 的时候确定的, 而 ***K*** 是在 **query** 的时候确定的); ***r*** 对于不同的**term**来说可以不一样
- 好处:
    - 在query的时候, 只需要计算在champion list里的docs的cosine score, 然后最后merge之后排序输出前K个给用户就行了, 需要考虑的文档数大大减小了

Strategy6: High and Low lists Heuristic

- **For each term, maintain <u>two</u> postings lists called *high* and *low***
- 方法: 相较于传统方法只维护一个postings list, 现在维护两个postings lists
  - (可以想像一个为high一个为low, high即为champion list).
  - 当拿到一个query时首先看high lists里包含的文档数有没有达到K:
    ① 如果已经<u>比**K**多</u>了, 那就可以只用*high lists*来返回用户的query;
    ② 如果<u>比**K**少</u>, 为了给用户一共K个结果, 需要从*low lists*里再取若干个直到达到K
- 好处:
  - 既像 ***Strategy3*** 一样节省了时间, 又保证用户一定能得到一定数目的结果

★ Note: This assumes we have a means for segmenting index into two tiers
这么做的前提是我们有某种方法把 **inverted index** 分成两个 **tiers**

Static Quality Scores Heuristic

- We want top-ranking documents to be both ***<u>relevant</u>*** and ***<u>authoritative</u>***
  - ***Relevance***: is being modeled by **<u>cosine scores</u>**
  - ***Authority***: is typically a query-independent
    - 这个文档的权威性(可信度), 比如我写的page和NY Times写的page肯定就是他写的Authority更高

Strategy4: Introduce an Authority Measure

- 方法:
  - Assign to each document ***d*** a *query-independent* <u>quality score</u> in [0,1]
    - Denote this by ***g(d), g*** stands for ***<u>goodness</u>***
  - 定义一个 ***g(d)*** 表示某个 document ***d*** 的 **<u>Authority</u>**, g(d) ∈ [0, 1]
  - 修改score的公式为: $score_{net}(q, d) = g(d) + cosine(q, d)$
    - A score of combination of authority and relevance
  - 原来在 posting lists 排序的顺序是按照 DocID 的, 现在改为按照 g(d) 从大到小排, 这样越权威的 doc 会出现在越靠前的位置 (<u>前提是两个 docs 的 cosine score 相同</u>)

Strategy5: Reorganize the Inverted List

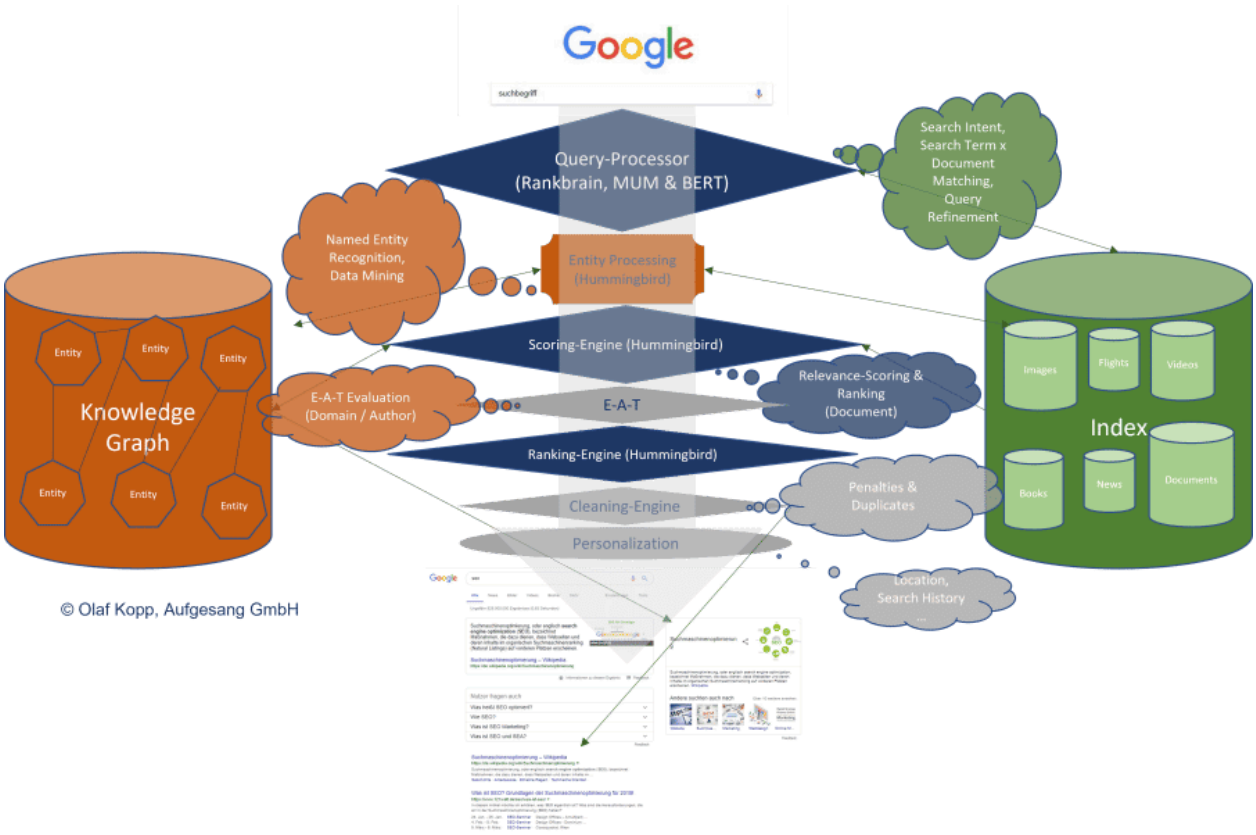
- Instead order all postings by ***g(d)*** the ***authority measure***
- Combine <u>champion lists</u> with *g(d)*-ordering
  - Maintain for each term a champion list of the ***r*** docs with highest ***g(d)*** **+tf-idf$_{td}$**
  - **This is equivalent to**

$$\text{net-score}(q,d) = g(d) + \frac{\vec{V}(q)\cdot\vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}.$$

Reverse Engineering of Google's query processing algorithm

- 两家公司Searchmetrics和Moz.com通过检测相同query得到的结果随着时间和多 metrics (例如点击率) 的变化在 google search 上的排序变化, 从而反向推断 google 的算法目前更看重哪些指标

Google' Query Processing Architecture



Entity Recognition in the Knowledge Graph

RankBrain An Entity-Based Processor

- 定义:
  - **RankBrain:** is a deep learning-based algorithm that is used after the selection of an initial subset of search results
    - (简单来说就是RankBrain会猜测用户真实想要搜索的内容, 比如搜索"美国的首都是哪里?", 你的真实搜索意图是寻找美国的首都的名字, 而"哪里"这个关键词是否存在于目标网页上, 已经不再重要了)
- 方法:
  - RankBrain ***maps keywords into entities*** which are then looked for in the Knowledge Graph (将keywords和entities一一对应)

★ Note: **Entity-based**和**term-based**的区别:

- A purely term-based search engine would not recognize the difference in meaning of the search queries “red stoplight” and “stoplight red” because the terms in the query are the same, just arranged differently. Depending on the arrangement, the meaning is different. **An entity-based search engine recognizes the different context based on the different arrangement**. “stoplight red” is its own entity while “red stoplight” is a combination of an attribute and an entity “stoplight”
- 纯粹基于术语的搜索引擎不会识别搜索查询“red stoplight”和“stoplight red”的含义差异，因为查询中的术语相同，只是排列不同。根据排列的不同，意义也不同。基于实体的搜索引擎根据不同的排列来识别不同的上下文。“stoplight red”是它自己的实体，而“red stoplight”是属性和实体“stoplight”的组合。

- 流程:
  - 可以将每个用户的一次搜索看成对 **RankBrain** 的一次训练
    (记住这是一个 **deep learning** 的模型), 对于每一次搜索:
    - RankBrain 根据现有的权重对这个 query 输出一个搜索结果页
    - RankBrain 会观察用户对当前结果的 ’态度’ (用户是否点击了这些link,用户对排序的顺序是否满意)
    - RankBrain 会根据得到的用户反馈更新自己的权重 (假设搜索同一个东西的很多用户对推荐的列表的后几个很感兴趣, uprank这些link)

- Google claims that ***RankBrain*** is **the 3rd most important factor** in their ranking algorithm **(links/words being numbers 1 and 2)**

Entity Recognition:

- Google的 Knowledge Graph 很大程度上依靠 wikipedia 的 structured content.
  - 🌰: 搜索 “Apple founder”,
    - 首先Identify Entity这一步会将这个query识别为 “Apple公司的founder”,
    - 然后google去自己的的Knowledge Graph中Apple这个词条下founder的信息, 找到是Steve Jobs.
    - 所以最后返回给用户的结果是和Steve Jobs相关的.
    - 而Steve Jobs甚至完全没在用户的query中出现

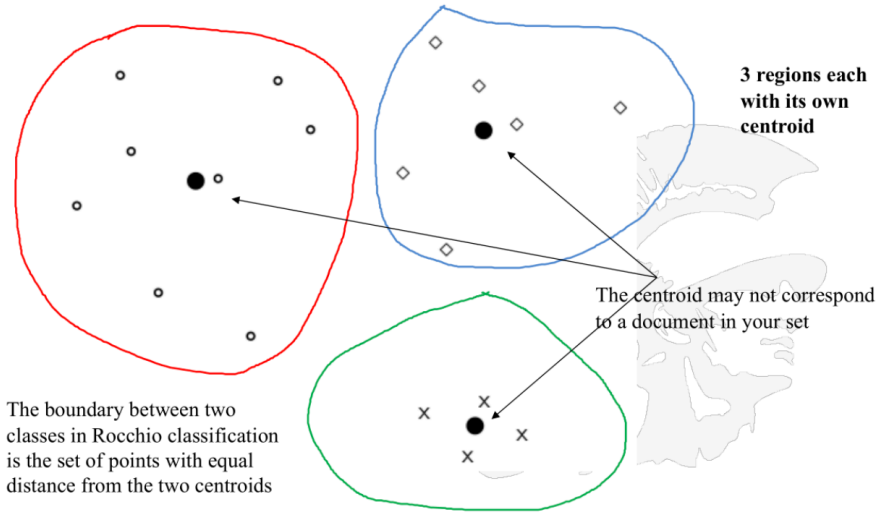
Classification **(How to classify a document?)**

Rocchio (Basics)

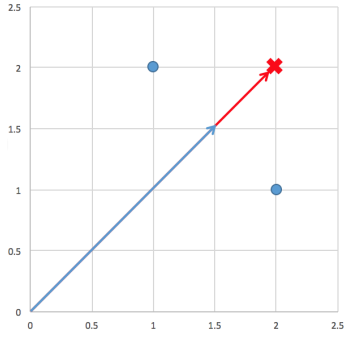
- The Rocchio algorithm is a method of **relevance feedback**
- It assumes documents are represented using the ***vector space model***
- The algorithm uses the notions of relevant/non-relevant documents and centroids
- **Centroid:** the **center of mass** (or vector average) of a set of points.
- Definition:
  - **Centroid:**

$$\vec{\mu}(C) = \frac{1}{|C|}\sum_{d \in C}\vec{d}$$

  - where **C** is a set of documents, **|C|** is the size of the set, and ***d*** is the normalized vector representing document ***d***



- ***Rocchio*** 问题:
  - *Rocchio* 算法是通过计算 *query* 到每个 *class centroid* 的距离, 选择最近的那个 *centroid* 作为新的 *query* 以一个范围去找最相关的 *document* 返回给用户.
  - 假设现在有相关文档和不相关文档两个类别,
    分布如图所示, 其中圆点代表相关文档, 叉代表不相关文档.
    - 我们就会选择(1.5, 1.5)这个 *centroid* 作为我们的新 *query*.
    - 但是这里如果以(1.5, 1.5)为中心, 不相关文档和相关文档到这个点的距离是一样的, 我们仍然会返回不相关文档给用户.



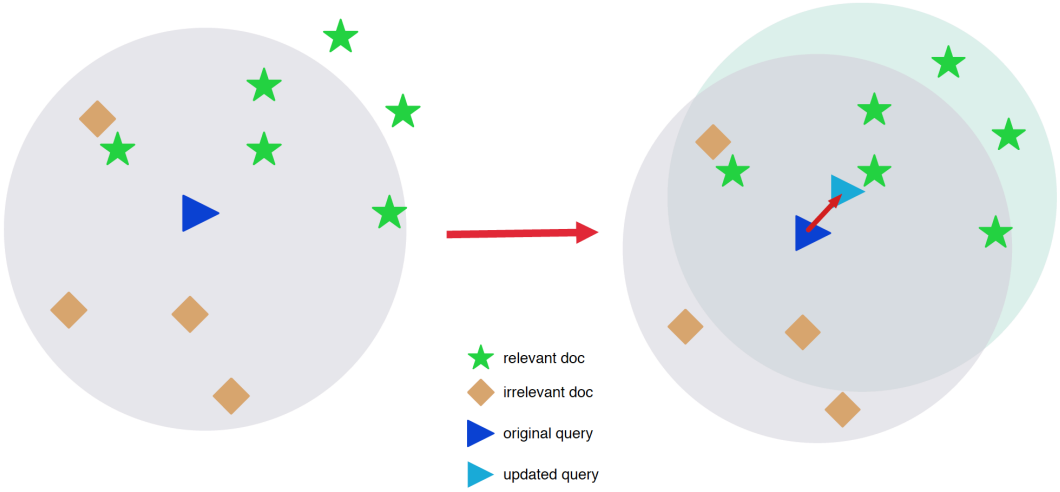
- 解决方法: 添加 ***Relevence Feedback***
  - 用相关文档的质心减去不相关文档的质心
  - 使新 query 这个向量稍微偏移一点,
    使它虽然离相关文档远了一点, 但是离不相关文档远的更多

- In practice Rocchio is often parameterised as follows:

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

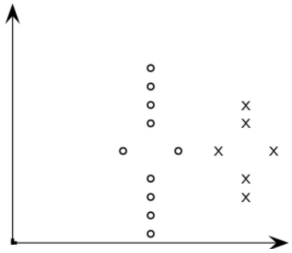
- **q0** 是原始的查询向量，**Dr** 是已知的相关文档集合（即 top results in PRF），**Dnr** 为不相关文档集合。
- **α、β** 及**γ** 是上述三者的权重。
  - Reasonable values are *a = 1.0, β = 0.75, y = 0.15*
  - 这些权重能够控制判定结果和原始查询向量之间的平衡：如果存在大量已推断的文档，那么会给β 及γ 赋予较高的权重。这些权值能够凭经验或对文本数据的认识在赋值，也能够通过实验来调试赋值。

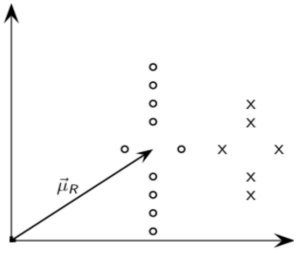
★ Example: 下图表现了添加偏移之后的效果



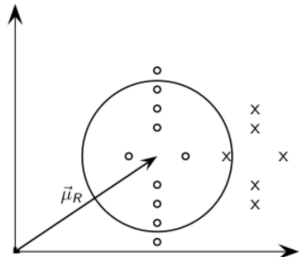
2D Rocchio Example

- For 2D examples the relevant set is generally much smaller than the non-relevant set;
- As a result we need a slightly modified rule
  - Circles represent relevant documents;
  - Xs represent nonrelevant documents

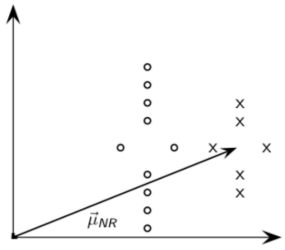




$\vec{\mu}_R$: centroid of relevant docs



$\vec{\mu}_R$:does not separate relevant/nonrelevant.



$\vec{\mu}_{NR}$: centroid of nonrelevant documents.

...

Classification Methods

- **Manual** classification
  - **Accurate** when job is done by experts
  - **Consistent** when the problem size and team is small
  - **Difficult** and **expensive** to scale
    - Means we need automatic classification methods for big problems

The Problem Statement for Classification

- Given two things:
  1. A description of an instance, ***x*∈*X***, where ***X*** is the *instance language* or *instance space*, and
  2. A fixed set of categories: ***C = {C1, C2,..., Cn}***
- Determine:
  - The category of ***x: c(x)*∈*C***, where c(x) is a ***categorization function*** whose domain is ***X*** and whose range is ***C***.
    - Functions that categorize are called "**classifiers**"

Classification Using Vector Spaces

- In vector space classification, the training set corresponds to a labeled set of document vectors
- **Premise 1:** Documents in the same class form a contiguous region of space
  - This is referred to as the ***contiguity hypothesis***
- **Premise 2**: Documents from different classes don't overlap (much)
- Learning a classifier is equivalent to building (defining) surfaces to delineate classes in the space

Ways to Measure Distance

- For normalized vectors ***Euclidean distance*** and ***cosine similarity*** correspond

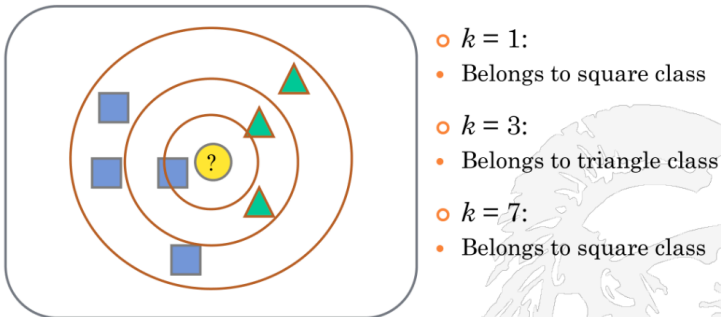
Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

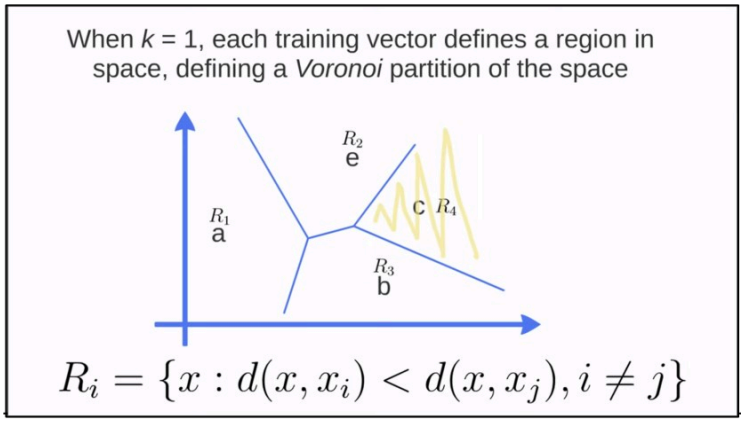
kNN Classification

- Initially we have a set of *N* documents that **have already been classified**
- **To classify a document *d***
  - locate among the ***N*** documents the ***k*** closest ones
  - from these ***k*** neighbors, pick the class that occurs most often, the majority class, and use that as the label for ***d***
- ex.
  - 

- Choosing the value of ***k***:
  - If ***k*** is too small, sensitive to noise points
  - If ***k*** is too large, neighborhood may include points from other classes
  - Choose an odd value for ***k***, to eliminate ties
- K的选择
  - k 的最佳选择取决于数据；
  - 在二元（两类）分类问题中（例如男性或女性），选择 k 为奇数会很有帮助，因为这样可以避免平局选票
  - 交叉验证是通过使用独立的数据集来验证 K 值来回顾性地确定 K 的良好值的另一种方法
  - 大多数数据集的最佳 K 一直在 **3-10** 之间
- ***K*-NN** is an example of a **non-linear** classifier; (***Rocchio*** is a linear classifier)

Voronoi Diagram (沃罗诺伊图)

- Special case (***k* = 1**) of k-NN



- A Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane

    - 根据到平面特定子集中的点的距离将平面划分为多个区域
- ***K=1* Nearest Neighbor Regions are Polygons**

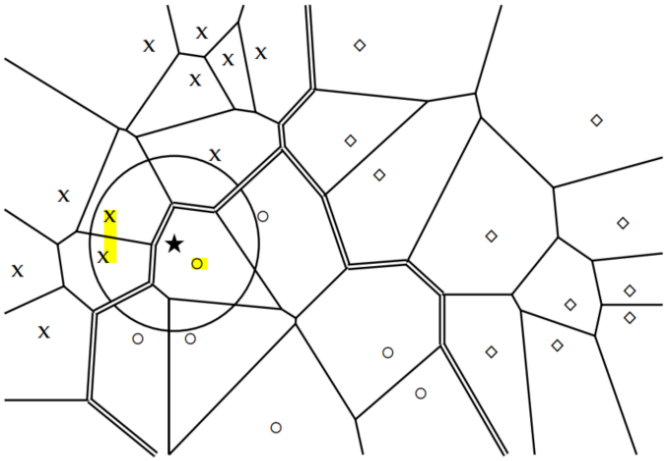


Figure 14.6 Voronoi tessellation and decision boundaries (double lines) in 1NN classification. The three classes are: X, circle and diamond.

- For ***1-NN:***
    - we assign each document to the class of **its closest neighbor**
- For ***k-NN:***
    - we assign each document to the majority class of its ***k closest neighbors*** where ***k*** is a parameter

K-NN: Final Points

- **No feature selection** necessary
- **No training** necessary
- **Scales well** with large number of classes
    - Don't need to train n classifiers for n classes
- Classes can **influence each other**
    - Small changes to one class can have ripple effect
- In most cases it's **more accurate than Rocchio**

Clustering **(for classification, 主要讨论在document领域)**

- **Clustering**: **the process of grouping a set of objects into classes of similar objects**
  - Documents within a cluster should be similar.
  - Documents from different clusters should be dissimilar.
- **Clustering is the most common form of *unsupervised learning***
  - *Unsupervised learning* = learning from raw data, as opposed to supervised learning where a classification of sification of examples examples is given a priori
- Clustering is a common and important task that finds many applications in IR and other places

What Is A Good Clustering?

- **Internal criterion:**
  - **A good clustering will produce high quality clusters in which:**
    - the **intra-class** (that is, intra-cluster) similarity is high
    - the **inter-class** similarity is low
    - The measured quality of a clustering depends on both the document representation and the similarity measure used

Three Criteria of Adequacy for Clustering Methods

1. The method produces a clustering which is **unlikely to be altered drastically** when further objects are incorporated
   - i.e. it is stable even under significant growth
2. The method is **stable** in the sense that small errors in the description of objects lead to small changes in the clustering
3. The method is **independent** of the initial ordering of the objects

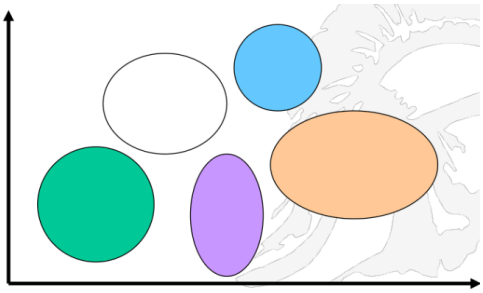
Classification vs. Clustering

- **Clustering**:
  - Unsupervised Learning
  - similarity measure
- **Classification**:
  - Supervised Learning
- *Clustering* precedes *Classification*

如何用Clustering来做Classification问题?

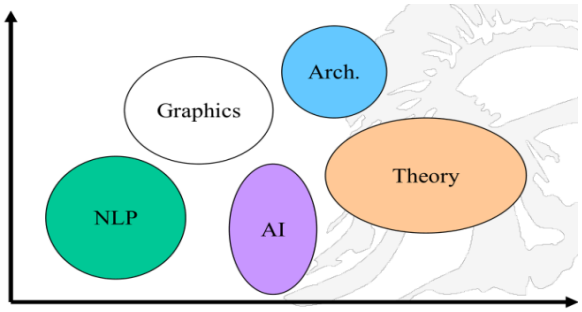
- ***Clustering*** 是unsupervised learning, 而 ***Classification*** 是supervised learning

1. 对数据进行 **Clustering**

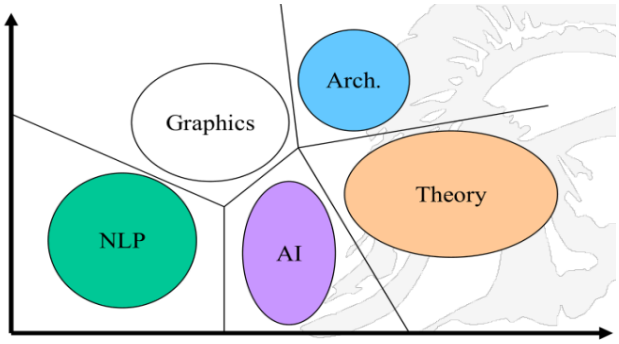


a. *How to represent the document?* → vector space; point space
b. *How to compute similarity/distance?* → cosine similarity; euclidean distance
c. *How many clusters?* → fixd priori number; data driven
d. Avoid **a cluster to be too large** / **too small** (用户得到的document过多或者过少)

2. 对每一个得到的 **cluster** 进行 **label** (总结出 **cluster** 中的通用关键词),
这样每个 **cluster** 实际上就变成了一个 **class**.
   a. ***Hard clustering***:
      - Each document belongs to exactly one cluster
      - 每个文档只能属于一个cluster (easier)
      - More common and easier to do
   b. ***Soft clustering***:
      - A document can belong to more than one cluster.
      - 每个文档可以属于多个cluster



3. Compute boundaries for the clusters that can be used as new documents appear
   - 计算出每个 cluster 的 boundary (每个cluster就变成了一个多边形区域)



4. 对于新输入的需要分类的文档, 实际上只是把它归到某一个cluster里就相当于完成了分类
   - 如果一个document恰好坐标在boundary的线上, 那它就会被同时归为多个类

Clustering Algorithms

- Two general methodologies
  - *Partitioning* Based Algorithms
  - *Hierarchical* Algorithms
- **Partitioning Based (划分聚类)**
  - Choose K and then divide a set of N items into K clusters
    - 有一堆散点需要聚类，想要的聚类效果就是“类内的点都足够近，类间的点都足够远”。
    - 首先你要确定这堆散点最后聚成几类，最后挑选几个点作为初始中心点，再然后依据预先定好的启发式算法给数据点做迭代重置，直到最后到达“类内的点都足够近，类间的点都足够远”的目标效果。
      - 如：k-means、k-medoids、k-modes、k-medians、kernel k-means
- **Hierarchical (层次聚类) – Bottom Up/Top Down**
  - **agglomerative**: pairs of items or clusters are successively linked to produce larger clusters (hierarchy produced ***bottom-up***)
    - “自底向上”的聚合 (agglomerative)
    - “自底而上”的算法开始时把每一个原始数据看作一个单一的聚类簇，然后不断聚合小的聚类簇成为大的聚类。
  - **divisive**: start with the whole set as a cluster and successively divide sets into smaller partitions (hierarchy produced ***top-down***)
    - “自顶向下”的分拆 (divisive)
    - “自顶向下”的算法开始把所有数据看作一个聚类，通过不断分割大的聚类直到每一个单一的数据都被划分。
  - 层次聚类通过对数据集在不同层次进行划分，从而形成树形的聚类结构。数据集的划分可采用“自底向上”的聚合（agglomerative）策略，也可采用“自顶向下”的分拆（divisive）策略。

Partitioning Based (划分聚类)

K-means (partitioning-based algorithm)

- 特点: k的值是提前定好的
- 步骤:
  1. 随机取k个点作为中心
  2. 计算所有点到这两个点的距离,到哪个点距离小就归为哪个簇
  3. 计算新的中心坐标 (x,y各取平均)
  4. 重复 2, 3 过程直到新的中心坐标和上一轮一模一样
- Summary Details
  - Initial centroids are often chosen randomly
    - Clusters produced vary from one run to another
  - The centroid is (typically) the mean of the points in the cluster
  - 'Closeness' is measured by cosine similarity, a variation of Euclidean distance
  - Most of the convergence happens in the first few iterations.
    - Often the stopping condition is changed to 'Until relatively few points change clusters

- 复杂度: $O(i * k * n * m)$,

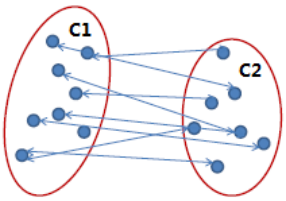
- n = number of points, k = number of clusters, i = number of iterations, m = number of attributes
- i 是 iteration 次数, k 是 cluster 个数, n 是 point 个数, m 是 attribute 数

Hierarchical Clustering (层次聚类)

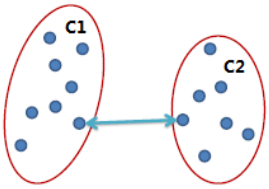
- 类型: Agglomerative (bottom up) & Divisive (top down)
- 步骤:
  - Repeatedly combine two nearest clusters

★ *How to compute the distance between two clusters?*

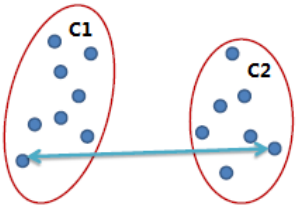
1. **Center of Gravity**
   - compute the distance between the **two centroids** of the cluster
2. **Average Link (平均连锁)**
   - Compute the **mean distance** between all pairs of points across the two clusters
   - 两个簇之间两两点之间距离的平均值，该方式可有效地排除噪点的影响。



3. **Single Link (单连锁)**
   - Compute the distance between **the two closest points** in the two clusters
   - 两个簇之间最近的两个点的距离作为簇之间的距离，该方式的缺陷是受噪点影响大，容易产生长条状的簇。



4. **Complete Link (全连锁)**
   - Compute the distance between **the two furthest points** in the two clusters
   - 两个簇之间最远的两个点的距离作为簇之间的距离，采用该距离计算方式得到的聚类比较紧凑。



A Dendrogram is Used to Display Clusters

Dendrogram(树状图):

- is a tree diagram frequently used to illustrate the arrangement of the clusters produced by hierarchical clustering

Question Answering vs. Information Retrieval

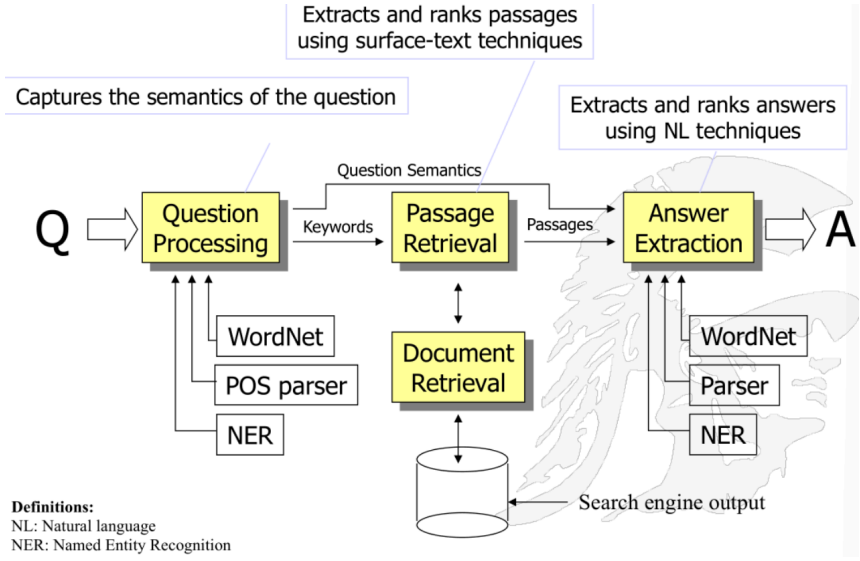
- **Question Answering** (**QA**)
  - **QA** is concerned with building systems that automatically answer questions posed by humans in a natural language
- **Information Retrieval**
  - Previously:
    - ***Document retrieval*** actually
    - Human job after that
  - Modern search engine:
    - Combine knowledge graphs, n-gram, WordNet, Inferencing…

3 Main Phases for Question/Answering

1. **QUESTION PROCESSING**
   - Detect question type (who, what, when, where, etc)
   - Identify important entities and formulate queries to send to a search engine
2. **PASSAGE RETRIEVAL**
   - Retrieve ranked documents (snippets only)
   - Break into suitable passages and match against entities
   - Passage Retrieval 相当于是 ***inverted index*** 的过程
3. **ANSWER PROCESSING**
   - Extract candidate answers (as named entities)
   - Rank candidates
     - **using evidence from relations in the text and external sources**

- 主要关注Question Processing和Answer Extraction.
- Passage Retrieval相当于是 inverted index 的过程

QA 3 Phase Block Architecture



PART 2: Passage Retrieval

- Once we have formulated queries using tools like NER, POS, variant expansion and WordNet results
- Send queries to a search engine and retrieve snippet results
- Filter the results for correct type
    - use answer type classification
    - Rank passages based on a trained classifier (application of machine learning)
    - Features:
        - Question keywords, Named Entities
        - Longest overlapping sequence,
        - Shortest keyword-covering span
        - N-gram overlap between question and passage

Capabilities for QA Systems

- **Part-of-Speech Tagging** (**POS parser**)
    - a piece of software that reads text in some language and assigns **parts** of **speech** to each word, such as noun, verb, adjective, etc.
        - ***Markov Models*** are now the standard method for part-of-speech assignment
    - 将一个问题里的每个word都标上词性, 从而判断哪些词容易成为keyword.同时, POS标记一旦手动完成, 就可以在计算语言学的上文中使用通过Markov Model求出可能的下文

  ★ Note: 这个方法准确率很低
    - 因为很多时候 **question** 里的词的顺序不是一定的. 比如有人的问题可能是 “今天天气如何?”, 而有人的问题可能是“天气怎么样今天?”.
        - 词的顺序不定性会导致 Markov Model 失效

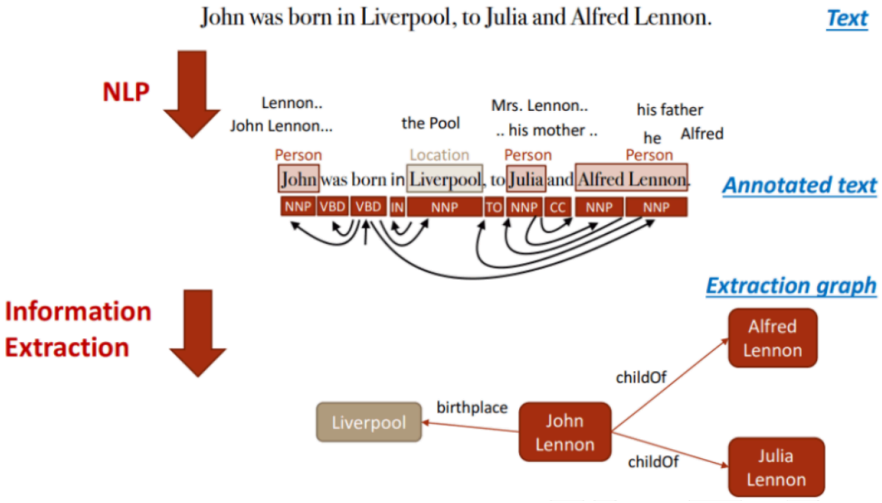
- **Named Entity Extraction** (**NER**)
    - 也叫 ***Semantic Relations*** (通常用BERT实现)
    - Software that seeks to locate and classify **named entities** in text into pre-defined categories such as the names of persons, organizations, locations…
    - 将问题中的词进行entity的分类 (可以参考下面🌰)
        - 

- **Determining Semantic Relations**

- semantic relations are concepts or meanings between entities.
  - E.g. [School] is a kind of [educational institution]
  - Opportunity to use WordNet

Answer Extraction

NLP Extraction (build Knowledge Graph)



General Keyword Selection Algorithm

1. Use the part-of-speech recognizer to identify all
   a. Nouns
   b. Verbs
   c. non-stopwords in quotations
   d. NNP (proper noun) words in recognized named entities
   e. complex nominals with their adjectival modifiers
2. Select the answer type word

Expanding the Keyword Set Using Variants

- There are 3 distinct ways to expand the keyword set determined by the keyword selection algorithm
  1. **Morphological variants** (having to do with **inflections**)
     - morphology [词法学, 形态学、构词学], inflections [词形变化]
     - invented → inventor → inventions
  2. **Lexical variants** (similar to **synonyms**)
     - Lexicon [词库]
     - killer → assassin
     - far → distance
  3. **Semantic variants**
     - like → prefer (a more **specific** or less specific term)

Appendix:

03/28/2023课堂扩展内容: Why language question answering is so easy for human but so hard for AI?

- 可以把问题转化为人、动物、AI之间理解问题的模式有什么区别

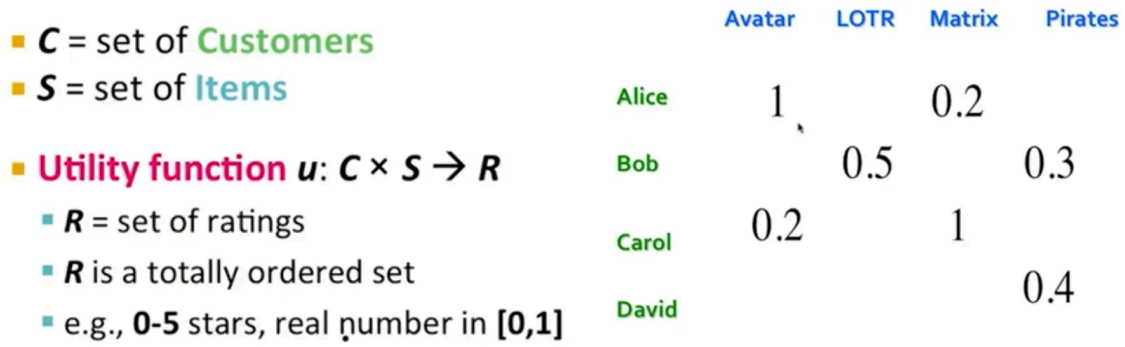
1. 信息的理解被分为两个部分: non-symbolic和symbolic. Symbolic表示看到的或者听到的信息被转化为某个我们理解的entity; non-symbolic表示个人的经历以及直觉.
2. Human是既具有symbolic又具有non-symbolic的能力的. 比如开车看到stop sign的时候如果一个人理解英语他就能知道这个符号代表着停止, 这就是symbolic. 而他学过开车所以他知道看到stop sign应该怎么做是符合交规的, 这就是non-symbolic.
3. Animal是不太具有symbolic, 只有non-symbolic的能力的. 比如狗或者猫只能在训练下听懂极少数的命令, 但是它们知道饿的时候该找什么吃.
4. AI是只具有symbolic而完全不具有non-symbolic的能力的. 你可以给一个AI对任意一个term一个全新的定义, 对它来说没有任何的区别. 它们并不能感知这些定义之后的真实意义, 比如你可以让AI理解stop就是停止, 但是停止这个动作的意义以及为什么看到stop sign要停止对于AI来说是没有意义的, 也是不被需要的.

因此AI是没有情感的因此, 当你问AI一个问题时, 它必须要和数据库里的定义有确切的联系才能让AI做出相应的回答, 而AI不会为你的问题做出任何预设 (比如你问AI“后天天气怎么样?”, AI并没有时间观念, 所以首先要查询今天的日期, 然后它需要把“天气”量化为温度、湿度、气候等指标才能对你做出回答)

Recommender System

- ***Recommendation system***: is any system which provides a recommendation / prediction/opinion to a user on items.
- Two Types of Recommendation Systems:
  - ***Content-based filtering system:***
    - uses item similarity/clustering to recommend items like ones you like
  - ***Collaborative filtering system:***
    - uses user similarity, i.e. the links between users and the item they chose as the basis of recommendations
- Commonly use ***hybrid*** systems in which both kinds of techniques are employed

Formal Model of the recommender system



- **Utility function** that looks are every customer and item pair and map to a rating, which stands as R.
- ***Utility matrix:*** 根据 utility function 得到的矩阵, 是一个 item 和 customer 的映射
- ***Prediction***: 注意到上图有很多空白部分, 预测的目的就是判断该不该推荐某物给某人. 设定一个推荐阈值很重要, 不易推荐过多结果给用户.

- 推荐系统的关键问题是找出效用矩阵中的空白值。如果符合条件，则将预测的未知数推送给用户。推荐者面临三个关键问题：
  1. **Gathering known ratings for matrix** (收集矩阵的已知评级)：
     - 如何收集效用矩阵中的数据
  2. **Extrapolating Utilities** (从已知评级推断未知评级)：
     - 我们主要对高未知评级感兴趣
  3. **Evaluating extrapolating methods** (评估外推方法)：
     - 如何衡量推荐方法的成功/性能。

- ***Gathering Ratings:***
  1. Explicit: ask people to rate items. Doesn't scale: only a small fraction of users leave ratings and reviews
  2. Implicit: learn ratings from user actions (Purchase implies high rating, what about low ratings?)

- ***Extrapolating Utilities:***
  - Key problem:
    1. utility matrix U is sparse. Most people have not rated most items
    2. Cold start: new items have no ratings, new users have no history
  - Three approaches to recommender systems:
    1. Content based
    2. Collaborative
    3. Latent based models

1.Content-based Recommender Systems (consider single user)

- 特点:
  - Use **characteristics** of an item
  - Recommend items that have **similar content to items user liked in the past**
  - Or items that **match predefined attributes of the user**
- 步骤:
  - 定义一些 features,构造 item profiles(vector) (**item-based**)
  - 用相同的 features 构造 user profiles(vector) (**user-based**)
  - 用 Cosine distance 计算 user 到 item 的距离,用分类算法匹配最近的 user-item
- 优点:
  - No need for data on other users:
    - No cold-start (for item) or sparsity problems
      (i.e., new items can receive recommendations)
  - Able to recommend to users with unique tastes
  - Able to recommend new & unpopular items:
    - No first-rater problem
      (i.e., new products never have been rated, therefore they cannot be recommended)
  - Able to provide explanations:
    - Can provide explanations of recommended items by listing content-features that caused an item to be recommended
- 缺点:
  - Finding the appropriate features can be hard
    - E.g. movie features may be obvious but what about images and music
  - Overspecialization
    - Never recommends items outside user's content profile
    - People might have multiple interests
    - Unable to exploit quality judgements of other users
  - Cold-start problem for new users
    - How to build a user profile
    - Never recommends items outside user's content profile
    - People might have multiple interests; **Unable to exploit quality judgments of other users (don't use ratings!)**

2.Collaborative filtering (consider other users)

- 特点:
  - Build a model from a **user's past behavior** (e.g., items previously purchased or rated), and **similar decisions made by other users**
  - Use the model to predict items that the user may like
  - **Collaborative**: suggestions made to a user utilizing information across the entire user base (用整个用户群体作为参考推荐东西给个体)
- Two types of Collaborative filtering
  - 两种类型 (***user-user***, ***item-item***)
- **user-user** vs. **item-item**
  - In theory, user-user and item-item are **dual approaches**
  - In practice, **item-item outperforms user-user** in many use cases
  - Items are "***simpler***" than users
    - items belong to a small set of "***genres***", but users have varied tests.
    - ***Item*** similarity is more meaningful than ***user*** similarity

Part A

Image understanding and search

- 经典数据集: **ImageNet**
- **ImageNet:**
  - ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images.

Code search

- Code search :
  - **tree + Inverted Indexing + Delta Indexing + deduplication + MinHash**)

LBS/PS

- **Location Based Search** (**LBS**), ie. **Proximity Search** (**PS**)
  - is about using **location data** where the query originates, and using that to return responses.
    - Location-based services (LBS) use real-time geodata from a smartphone to provide information, entertainment, or security. Some services allow consumers to check in at restaurants, coffee shops, stores, concerts, and other places or events.
- 概念:
  - using location data where the query originates, and using that to return responses.
- 如何快速反应动态*GPS*网络下的实时信息?
  → A Unified Approach to Spatial Proximity Query Processing in Dynamic Spatial Networks:
    - 文章主要贡献是提出了用cache存储动态网络中共用路径的状态来加速 **query** (比如两个人同时在使用google map, 分别从两个地方出发要到两个不同的地方. 但是他们各自的路径中有一条路(称为X)是都会走的, 那么在第一个人搜索的时候网络处理出X的当前情况, 第二个人在搜索的时候就可以用cache里存的X的情况, 不同再处理一次)
- 如何防止邻近服务中的用户隐私泄露? *(某些公司会通过手机定位你的确切地址,某些交友应用可以将你的定位泄露给其他用户)*
  - → Where's Wally? Precise UserDiscovery Attacks in Location Proximity Services:
    - 文章介绍了一种攻击"Wally"能利用APP中邻近服务的漏洞得到其他用户的定位, 并且提出了保护的方法, 包括对用户的位置信息进行模糊化处理 (只给APP一个框框, 表示用户位置在这个框内,而不是一个精确的点)

Similarity search (Vector DBs)

- **Vector DB:**
  - embeds into vector (XYZ...) orthogonal space, the items (data) we want to search for (using '**cosine similarity**'!).
  - 将传统的database (e.g. table) 映射到 **vector space** (每一列成为一个维度). 这样每一行, 也就是每个instance, 都会被表示为一个vector. 然后就可以在vector space上很容易用cosine similarity计算不同的instance之间的相似度
    - e.g., Pinecone

LDA, for topic modeling

- *Given a document, how to predict its dominant topic(s)?*
- **Latent Dirichlet allocation (LDA)**
  - a "generative statistical model" that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. So this is categorized as unsupervised learning.

★ Note1: **LDA** 是一个 **unsupervised learning** algorithm
(每个主题具体是什么内容要人工观察分析得到)

★ Note2: 一篇文章可能包含多个主题, 其中每个主题所占的比例不同
(通过主题词汇的统计得到比例)

ANN (Faiss)

- Exact nearest neighbors impractical
  - the **only** available method for guaranteed retrieval of the exact nearest neighbor is exhaustive search.
    保证检索精确最近邻居的唯一可用方法是穷举搜索 (由于 curse of dimensionality.)
  - Most modern-day applications have massive datasets with high dimensionality (hundreds or thousands).
    大多数现代应用程序都具有高维(数百或数千)的海量数据集
- Approximate Nearest Neighbor calcs are ***much faster*** than exact ones
  - 意义: ***KNN***的查询一个 **query** 的复杂度是 **O(n)**, 当 ***n*** 特别大的时候就会很慢. 我们希望在损失一定准确度的情况下大大提高查询的速度.
- **Approximate Nearest Neighbor** (**ANN**)
  - **Speed up** the search by **preprocessing** the data into an efficient index.
    1. **Vector Transformation**
       - applied on vectors before they are indexed, amongst them, there is dimensionality reduction and vector rotation.
         In order to this article well structured and somewhat concise, I won't discuss this.
    2. **Vector Encoding**
       - applied on vectors in order to construct the actual index for search, amongst these, there are data structure-based techniques

like Trees, LSH, and Quantization a technique to encode the vector to a much more compact form.

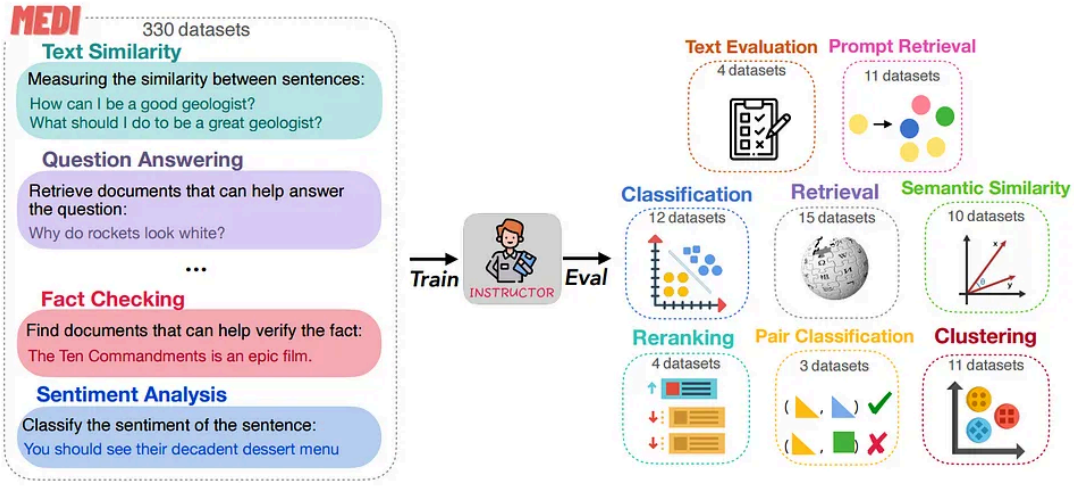
3. **None Exhaustive Search Component**
    - applied on vectors in order to avoid exhaustive search, amongst these techniques there are Inverted Files and Neighborhood Graphs.

Task-specific fine-tuning

- **'Fine-tuning'**
    - is the concept of adapting a general-purpose model, for a **specific purpose.**
        - **narrow and deep**
    - see: here

Task-specific embedding

- Task-specific embedding
    - 目标: Multiple NLP tasks can be handled by a system, if it embeds a document in a way that's related to the task
      (每一个任务的特点可能都不一样, 但是经过统一的 embed 之后可以用同一种量化标准来表示这些任务的数据. 这样就可以训练出一个模型同时可以处理多种 task)



Part B

External memory to enhance generic chat

- **ChatGPT 与 Neo4j**（a graph DB）一起使用（link）
    - rather than query Neo4j using its own ***'Cypher'*** query language, we can use natural language instead.
- **RR - Rethinking with Retrieval** (link)

    - solves reasoning-based tasks by using an external KB (knowledge base).
- **Retrieval Transformer** (link)
    - uses external memory to keep the core LLM size Down

LLM+tasks+memory -> a 'computer' system!

- LLM+tasks+memory -> **LLM + Langchain + Vector DB**
    - **LangChain**:
        - is a task programming language where we use specific commands in the form of 'templates', to compose our queries, and run() them, eg.
- **OPL (OpenAI + Pinecone + Langchain)**
    - is the name we could give to such a stack comprised of *O(penAI)+P(inecone)+L(angChain)*

  ★ Example: LLMs such as GPT-4 are 'pretrained' (that is the 'P') with a large amount of language data. For more precise answers, they need to be 'fine tuned' with specific domain-related (eg medical, legal...) language, OR, be chained to a custom DB that can be accessed by the GPT.
- In this context, *what is the 'OPL stack'? Describe in your own words, sticking to what we discussed in class (high level description is fine).*
- ★ *Ans*: OPL stands for OpenAI, Pinecone, and Langchain. OpenAI provides API access to powerful LLMs and also provides embedding models to convert text to embeddings. Pinecone provides embedding vector storage, semantic similarity comparison, and fast retrieval. Langchain allows users to build their own LLM applications, and can be used to fine-tune the models on domain-specific data or connect them to external databases.

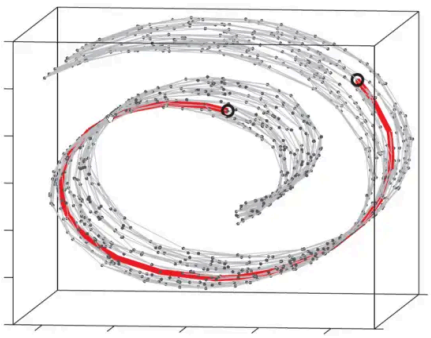
Clustering

- K-means clustering, t-SNE, HDBSCAN, UMAP
- t-SNE
    - 基本思路: 用条件分布的概率表示高维度中两点之间的相似性, 希望拟合的分布是 *t*分布 (类似正态分布)

Search

- **manifold search**: uses **manifold learning** for similarity searches
    - manifold learning: (也用于计算 **similarity**)
    - 基本思路:
        - 用非线性算法表达两点之间的距离从而降维

  ★ **Example**:
    - 如右图所示, 在一个卷起来的曲面内如果我们用两个点的空间几何距离表示他们的距离其实是不准确的. 我们应该把这个曲面展成一个平面才能表达这两个点的真实距离



- **NeRF**
  - 思路: 通过机器学习实现通过2D图像的信息生成3D模型
    - Standard LBS retrieves addresses, maps.
  - 应用: pokemon go (AR games); immersive views (Google Map)

Part C

Attention

- "A Transformer computes self-attention."
  - Step-by-Step Illustrated Explanations of Transformer

Context

- Because ***'context is everything'***, we need to address the ***'quadratic bottleneck' (***二次瓶颈***)*** of attention computation.

LoRA / QLoRA / ReFT

- **Fine-Tuning:**
  - In the realm of language models, fine tuning an existing language model to perform a **specific task** on specific data
- **Full finetuning**
  - involves optimizing or training all layers of the neural network. While this approach typically yields the best results, it is also the most resource-intensive and time-consuming.
  - 完全微调涉及优化或训练神经网络的所有层。虽然这种方法通常会产生最佳结果，但它也是最耗费资源和时间的。
- **PEFT (**Parameter-efficient finetuning**) methods:**
  - propose a efficient and cheaper alternative to full fine-tuning by updating a small fraction of weights, while using less memory and finishing training faster.

1. **LoRA** (Low-Rank Adaptation of Large Language Models, paper)
   - an improved finetuning method
   - instead of finetuning all the weights that constitute the weight matrix of the pre-trained large language model, two smaller matrices that approximate this larger matrix are fine-tuned. These matrices constitute the LoRA adapter. This fine-tuned adapter is then loaded to the pretrained model and used for inference.
   - 是一种改进的微调方法，其中不是对构成预训练大型语言模型的权重矩阵的所有权重进行微调，而是对近似该较大矩阵的两个较小矩阵进行微调。这些矩阵构成了 LoRA 适配器。然后，将此微调的适配器加载到预训练模型中并用于推理。
2. **QLoRA** (paper)
   - QLoRA is an even more memory efficient version of LoRA

        - the pretrained model is loaded to GPU memory as quantized 4-bit weights (compared to 8-bits in the case of LoRA), while preserving similar effectiveness to LoRA. Probing this method, comparing the two methods when necessary, and figuring out the best combination of QLoRA hyperparameters to achieve optimal performance with the quickest training time will be the focus here.
        - 是 LoRA 的内存效率更高的版本，
        - 其中预训练模型作为量化的 4 位权重（与 LoRA 的 8 位权重相比）加载到 GPU 内存，同时保留与 LoRA 类似的有效性。探索这种方法，在必要时比较两种方法，并找出 QLoRA 超参数的最佳组合，以最快的训练时间实现最佳性能将是这里的重点。

- **ReFT** (Representation fine-tuning, link):
    - PEFT modify **weights** of model but not the representations
        - PEFT（例如 LoRA）会修改模型的权重，但不会修改表示。

Issues

- **Bias in Search Results**
    - 问题: 搜索引擎的公司更偏向于展示和自己有经济利益的网站 (e.g. Google更喜欢展示youtube, 而MS更喜欢展示msn)

- **GDPR (General Data Protection Regulation)**
    - *What is GDPR?*
    - **Lawfulness, fairness and transparency**
        - Processing must be lawful, fair, and transparent to the data subject.
    - **Purpose limitation**
        - You must process data for the legitimate purposes specified explicitly to the data subject when you collected it.
    - **Data minimization**
        - You should collect and process only as much data as absolutely necessary for the purposes specified.
    - **Accuracy**
        - You must keep personal data accurate and up to date.
    - **Storage limitation**
        - You may only store personally identifying data for as long as necessary for the specified purpose.
    - **Integrity and confidentiality**
        - Processing must be done in such a way as to ensure appropriate security, integrity, and confidentiality (e.g. by using encryption).
    - **Accountability**
        - The data controller is responsible for being able to demonstrate GDPR compliance with all of these principles.

IP (Intellectual property)

- 四类IP:
    1. ***copyright***
        - for literary works, art, and music
    2. ***patents***
        - for inventions and processes
    3. ***trademarks***
        - for company and product names and logos
    4. ***trade secrets***
        - for recipes, code, and processes

    ★ 额外思考:
    1. AI生成的画作, 音乐如何侵犯了人类相关行业工作者的权利?
    2. Deepfake可能有什么糟糕的影响?