Information Retrieval

Johan Boye, KTH

Information Retrieval (IR)

- Att hitta **relevant** information i en **stor mängd** texter (och/eller bilder, audio, video, programkod, biomedicinsk data, …)
- Användaren ger en sökfråga
- Som svar returneras lista med (länkar till) dokument, eventuellt **sorterad efter relevans**.

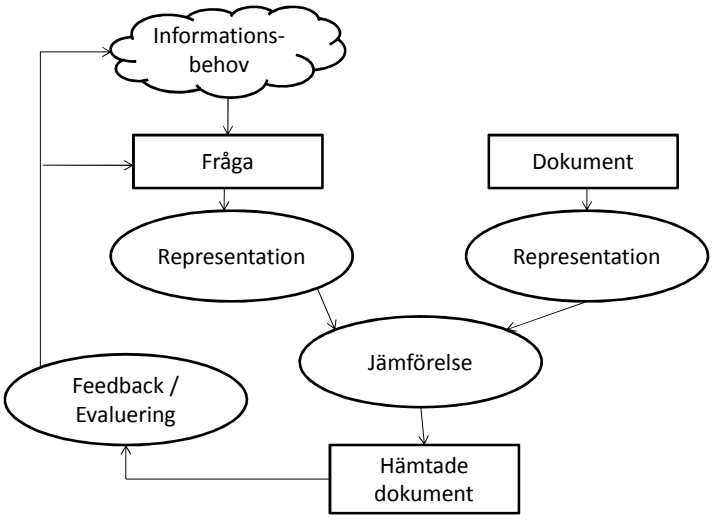
Information Retrieval (IR)

- IR handlar inte bara om att **hitta** och **ranka** texter, utan även om:
  - text**kategorisering** och –**klustring**
  - automatisk **sammanfattning**
  - ”**question answering**” (t. ex. ”Vem förlorade Krim-kriget” →”Ryssland”)
  - **multimedia-sökning**

Vad IR inte är

- Ett IR-system är **inte** ett databassystem.
- Databaser:
  - innehåller **strukturerad** data
  - tillåter **exakt** och **deterministisk** sökning
- Sökning i ett IR-system är **inexakt** i flera avseenden:
  - informationsbehov, sökfråga, resultat (vad är ett ”relevant” dokument?).

IR-processen



Utvärdering

- Precision $$p = \frac{|\, ret \cap rel \,|}{|\, ret \,|}$$

- Täckning (recall) $$r = \frac{|\, ret \cap rel \,|}{|\, rel \,|}$$

där |rel| = antal relevanta dokument (totalt)
|ret| = antal returnerade dokument

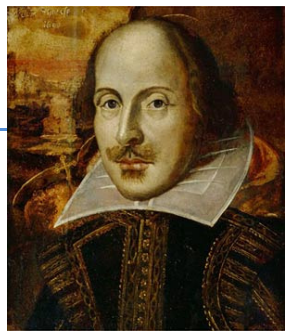
Varför studera IR?

- Kort svar: Google
  - söker bland ca 1000 miljarder dokument (!)
  - returnerar svar på ca 0,5s
  - är värderat till många miljarder dollar…
- Hur går det till?
  - hundratusentals maskiner
  - distribuerade lagrings- och sökningsmetoder
  - avancerad rankningsalgoritm (PageRank)

Hur funkar Google?

- Under huven finns vanlig IR-teknik, dvs:
  - Webben **indexeras**
  - Indexet kopplar termer till webbsidor
  - Användarens behov av information representeras av en **fråga** (query)
  - Frågor matchas mot webbsidor via indexet
  - Google försöker att returnera sidor som är **relevanta** för frågan

Exempel



- Vilka pjäser av Shakespeare innehåller orden ”**Brutus**” OCH ”**Caesar**” men INTE ”**Calpurnia**”?
- Förslag:
  - Sök efter ”**Brutus**” i alla pjäser → resultatmängd R1
  - Sök efter ”**Caesar**” i R1 → resultatmängd R2
  - Sök igenom R2 och returnera texter som **inte** innehåller ”**Calpurnia**”
- Vad är det för fel på denna lösning?

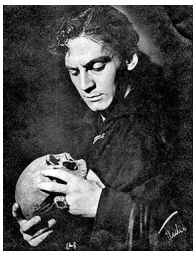
Term-dokument-matris

	Antonius och Cleopatra	Julius Caesar	Stormen	Hamlet	Othello	Macbeth
Antonius	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
nåd	1	0	0	1	1	1
medborgare	1	1	0	0	1	0

1 om pjäsen innehåller termen, 0 annars

Boolesk sökning

- Följande **bit-vektorer** representerar termerna:
  - Brutus: 110100, Caesar: 110111, Calpurnia: 010000, INTE Calpurnia 101111
  - Bitvis OCH: 110100 & 110111 & 101111 = 100100
  - Svaret på frågan blir alltså första och fjärde kolumnen i matrisen: **Antonius och Cleopatra** samt **Hamlet**



Boolesk sökning: Fördelar

- Enkel modell att förstå och implementera
- Betydelsen av en Boolesk sökfråga är exakt definierad
- Funkar bra för **vana användare** som arbetar med en **väldefinierad** samling dokument (typ bibiotekarier)

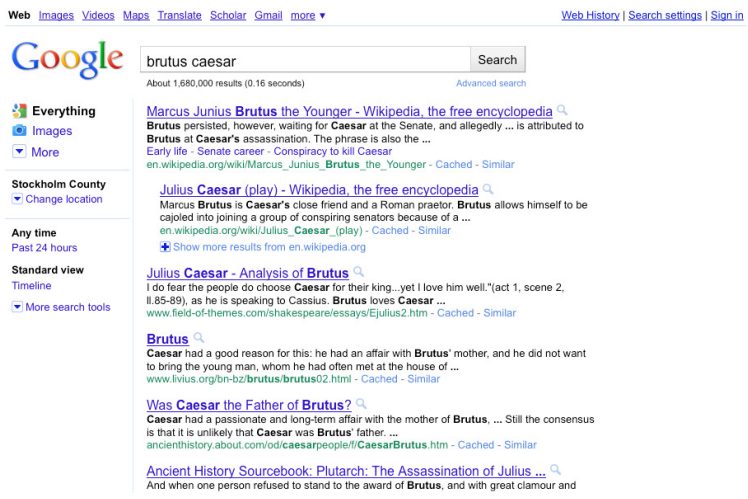
Boolesk sökning: Problem

- Många användare har problem att formulera sökfrågor
- Returnerar ofta **för många** eller **för få** resultat
  - ”zyxel P-660h” → 192 000 resultat
  - ”zyxel P-660h” ”no card found” → 0 resultat
- Krävs skicklighet att formulera en sökfråga som ger ett hanterbart antal resultat
- **Ingen rankning** av resultat
- Alla termer lika viktiga

Boolesk sökning i Google

- Google använder **inte** ren Boolesk sökning, men det finns vissa möjligheter:
  - **johan boye** vs **johan OR boye**
  - **johan boye –junge**
  - **johan boye** vs **”johan boye”**
  - Förut fanns **johan NEAR boye**, men detta verkar ha tagits bort

Rankade sökmodeller



Rankade sökmodeller

- Varje matchande dokument ges en poäng, t.ex. mellan 0 och 1
- Ju **högre poäng**, desto **bättre matchar** dokumentet frågan
- Notera: många träffar är inte längre ett problem
  - visa bara de k (=10) bästa resultaten
  - Förutsättning: Rankningsmetoden funkar

Rankade sökmodeller

- **Vektorrumsmodellen** - matchningsproblemet översätts till ett **geometriskt** problem
- **tf-idf-viktning** – tar hänsyn till söktermernas frekvens
- Vektorrum + tf-idf → modell där dokument blir rankade efter hur mycket de **liknar** sökfrågan

Rankning – fler alternativ

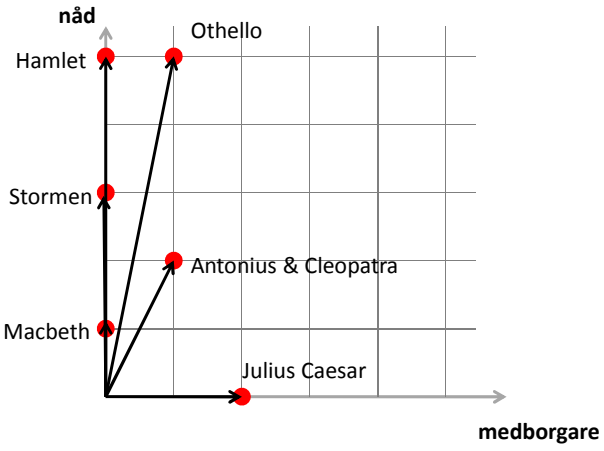
- Sidor som besöks av **många användare** hamnar högt
- Sidor som har **många viktiga in-länkar** hamnar högt (PageRank)
- **Välkända kvalitetssidor** (Wikipedia, tidningar) hamnar högt
- **Personaliserad sökning**: Rankning efter din tidigare sökningshistorik
- **Pengarna bestämmer:** Rankning efter hur mycket annonsörer betalar
- Kommersiella sökmotorer använder en blandning av flera kriterier

Matris - antal ordförekomster

	Antonius och Cleopatra	Julius Caesar	Stormen	Hamlet	Othello	Macbeth
Antonius	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
nåd	2	0	3	5	5	1
medborgare	1	2	0	0	1	0

Varje dokument är en punkt eller vektor i en term-rymd.

Dokument som vektorer



Bag-of-words-modell

- **$tf_{t,d}$** = antal gånger termen t förekommer i dokument d
  - Ett dokument ***d***→ ($tf_{1,d}$, $tf_{2,d}$, … , $tf_{n,d}$)
  - ”Kalle är bättre än Nisse” och ”Nisse är bättre än Kalle” har **samma** vektor
- Hur lika är två dokument ***d*** och ***e***?
- Första idén: **multiplicera** vektorerna **komponentvis**, **summera produkterna**

Viktning av termer

$w_{i,j} = tf_{i,j} \cdot idf_i$

- *tf* = **termfrekvens**
  - Sätt $\boldsymbol{tf_{i,j}} = n_{i,j}$ där $n_{i,j}$ är antal gånger term *i* förekommer i dokument *j*
- *idf* = **invers dokumentfrekvens**
  - Sätt $\boldsymbol{idf_i}$ = log(N/n) , där N är det totala antalet dokument, och n antal dokument där ord *i* förekommer

idf-viktning

- Notera: *idf* har ingen betydelse vid en-ords-frågor, som ”nyckfull”.
- Vid en tvåordsfråga som ”*nyckfull man*”, kommer förekomster av ”*nyckfull*” räknas mer än förekomster av ”man” när rankingen räknas ut

Matris – tf-idf-vikter

	Antonius och Cleopatra	Julius Caesar	Stormen	Hamlet	Othello	Macbeth
Antonius	0.104	0.047	0	0	0	0.100
Brutus	0.003	0.101	0	0.038	0	0
Caesar	0.041	0.038	0	0.020	0.011	0.026
Calpurnia	0	0.016	0	0	0	0
Cleopatra	0.098	0	0	0	0	0
nåd	0.0003	0	0.079	0.049	0.057	0.026
medborgare	0.0007	0.001	0	0	0.043	0

Matris - antal ordförekomster

	Antonius och Cleopatra	Julius Caesar	Stormen	Hamlet	Othello	Macbeth
Antonius	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
nåd	2	0	3	5	5	1
medborgare	1	2	0	0	1	0

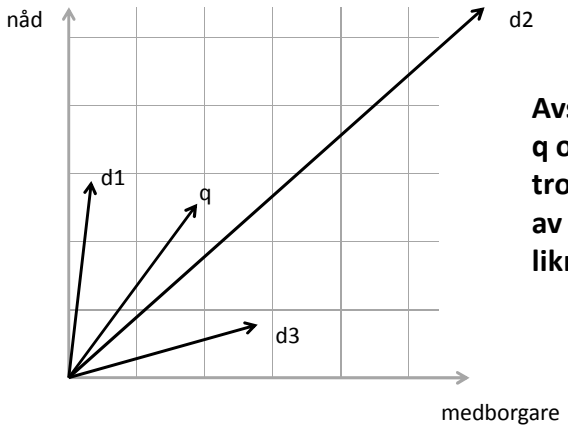
Mer om termviktning

- Stopplistor
  - Ta bort **icke-betydelsebärande** ord
  - Lingvistiskt baserat – funktionsord
  - Statistiskt baserat – vanliga ord
- Termnormalisering
  - **Trunkering** – maximal termlängd
  - **Lemmatisering** – morfologisk analys
  - **Stemming** – medelväg; suffix-borttagning

Sökfrågor som vektorer

- Representera även **sökfrågor som vektorer** i term-rymden
- Ranka dokument efter deras närhet till sökfrågan
- Närhet = avstånd?
- Närhet = vinkel?

Avståndsmått: dålig idé



Avståndet mellan q och d2 är stort, trots att fördelningen av termer i q och d2 liknar varandra.

Vinklar i stället för avstånd

- Tänk om:
  - Låt *d* vara ett dokument.
  - Lägg ihop två kopior av *d* efter varandra. Kalla detta dokument *dd*
- ”Semantiskt” har *d* och *dd* samma innehåll.
  - Avståndet mellan *d* och *dd* kan vara stort
  - Men: Vinkeln är 0, dvs maximal likhet

Cosinus-mått

- **Skalärprodukten** mellan *u* och *v*:

$$u \cdot v = \sum_{i=0}^{n} u_i v_i$$

- Det gäller också att:

$$u \cdot v = |u||v|\cos\theta$$

där $|u|$ = längden av *u*, och θ är vinkeln mellan *u* och *v*

- Alltså:

$$\cos\theta = \frac{\sum_{i=0}^{n} u_i v_i}{|u||v|}$$

Längd-normalisering

- **Dividera varje komponent** i vektorn *v* med längden på *v:*

$$|x| = \sqrt{\sum_i x_i^2}$$

- Varje (dokument-)vektor har då längd 1 (**enhetsvektorer**)
- Notera: Dokumenten *d* och *dd* är **lika** efter normalisering

Cosinus(sökfråga,dokument)

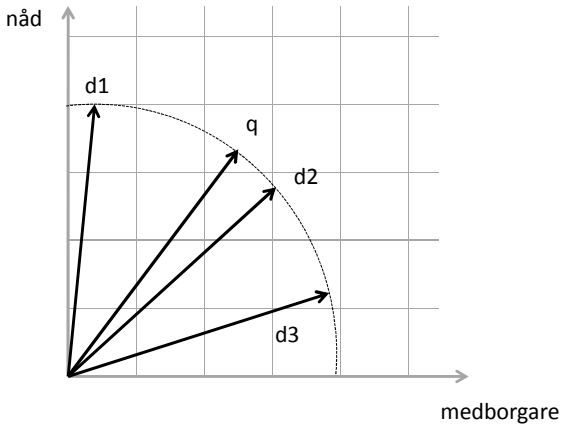
Skalärprodukt Enhetsvektorer

$$\cos(q,d) = \frac{q \cdot d}{|q||d|} = \frac{q}{|q|} \cdot \frac{d}{|d|} = \frac{\sum_{i=0}^{n} q_i d_i}{\sqrt{\sum_{i=0}^{n} q_i^2} \sqrt{\sum_{i=0}^{n} d_i^2}}$$

q_i är tf-idf-vikten av term *i* i sökfrågan

d_i är tf-idf-vikten av term *i* i dokumentet

Illustration av cosinus-mått



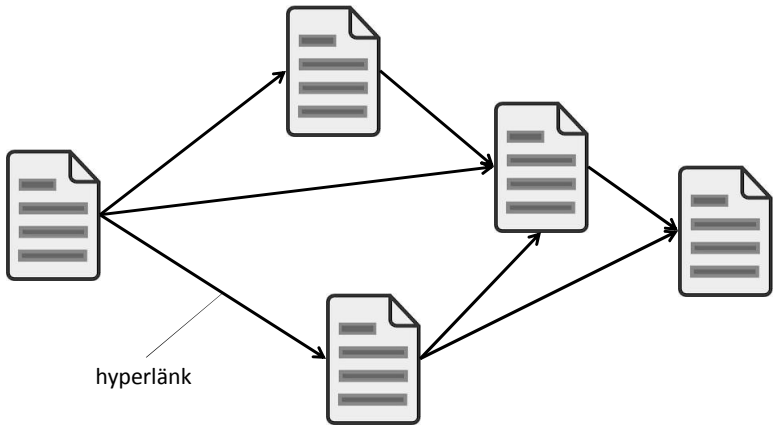
Sammanfattning

- Vektorrumsmodell-ranking:
  - Representerar varje dokument *d* som en viktad tf-idf-vektor
  - Representerar sökfrågan *q* som en viktad tf-idf-vektor
  - Beräknar cosinus-måttet [0..1] för *q* och varje *d*
  - Rankar dokumenten efter detta mått
  - Returnerar de bästa k (=10) dokumenten

Rankning av webbsidor

- Vi vill att returnerade dokument ska vara både relevanta och av god kvalitet
  - Relevans: cosinus-mått *cos(q,d)*
  - Kvalitet: oberoende av sökfrågan *g(d)*
  - *poäng(q,d) = cos(q,d)+g(d)*
- **PageRank** är ett sätt att mäta en sidas kvalitet

Webben är en riktad graf



Ranking med hjälp av länkstruktur

- **Antagande:** En länk från X till Y signalerar att Xs författare tycker att Y är en sida av god kvalitet.
  - X "lägger en röst" på Y.
- **Första förslag:** Rank = antal in-länkar

PageRank

- WWW har en speciell struktur som kan utnyttjas
  - sidor **länkar** till varandra
  - Ju fler in-länkar, desto högre rank
  - In-länkar från sidor med **hög rank** är **värda mer** än länk från sidor med låg rank
  - Denna idé ligger bakom **PageRank** (Brin & Page 1998)
  - En "random surfer" som slumpmässigt klickar på länkar kommer oftare besöka sidor med högre PageRank

PageRank – första försöket

$$PR(p) = \sum_{q \in in(p)} \frac{PR(q)}{L_q}$$

- där *p* och *q* är sidor
- *in(p)* är de sidor som länkar till *p*
- L_q är antal ut-länkar från q

- PageRank beräknas genom en iterativ algoritm

”The random surfer model”

- Antag en surfare som följer länkar **slumpmässigt**
- Länken som ska följas väljs med likformig sannolikhet
- Om surfaren kommer till en **sänka** (en sida utan länkar), börjar hon om på en **slumpmässigt vald** sida
- Då och då hoppar surfaren till en **slumpmässigt vald** sida (även om det finns länkar att följa)

PageRank – andra försöket

- Med **sannolikhet *1-c*** blir surfaren uttråkad, **slutar att följa länkar**, och **börjar om** på en slumpmässig sida
- Gissning: Google använder *c*=0.85

$$PR(p) = c\left(\sum_{q \in in(p)} \frac{PR(q)}{L_q}\right) + \frac{(1-c)}{N}$$

- Utan detta antagande kommer surfaren att fastna i en delmängd av webben.

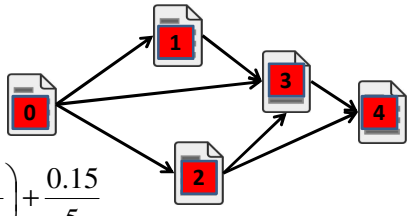
PageRank: exempel

$$PR_4 = 0.85 \cdot \left(\frac{PR_2}{2} + PR_3\right) + \frac{0.15}{5}$$

$$PR_3 = 0.85 \cdot \left(\frac{PR_0}{3} + PR_1 + \frac{PR_2}{2} + \frac{PR_4}{4}\right) + \frac{0.15}{5}$$

$$PR_2 = PR_1 = 0.85 \cdot \left(\frac{PR_0}{3} + \frac{PR_4}{4}\right) + \frac{0.15}{5}$$

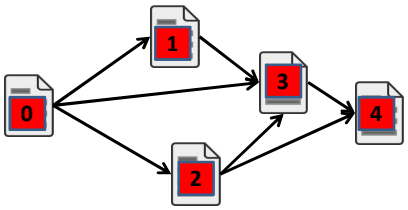
$$PR_0 = 0.85 \cdot \left(\frac{PR_4}{4}\right) + \frac{0.15}{5}$$



PageRank- tolkningar

- Popularitet / kvalitet / relativt informationsvärde
- PR_p = sannolikheten att slumpsurfaren kommer att befinna sig på sida *p* vid ett givet tillfälle
- Detta kallas den **stationära sannolikheten**

Exempel



- Stationära sannolikheter:
  (0.102, 0.131, 0.131, 0.298, 0.339)

  hittade efter 17 iterationer från startvektorn
  (0.2, 0.2, 0.2, 0.2, 0.2)

Googles ranking

- Google använder PageRank + hemliga utökningar
  - upptäcker länkfarmer
  - AdWords
  - hur sökorden är representerade på sidan (i rubriker, versaler/gemena, metadata, …)