TF-IDF

digger	2 (1/4) -> 4 (1/2)	IDF(*digger*) = 1/2
din	2 (3/8)	IDF(*din*) = 3/4
elg	4 (3/4)	IDF(*elg*) = 3/4
en	3 (3/4)	IDF(*en*) = 3/4
er	3 (3/4)	IDF(*er*) = 3/4
ingen	4 (3/4)	IDF(*ingen*) = 3/4
jeg	1 (1/4)-> 2 (1/4) -> 3 (1/4)	IDF(*jeg*) = 1/4
liker	1 (3/8)	IDF(*liker*) = 3/4
skilpadde	1 (1/8) -> 2 (1/8) -> 3 (1/4)	IDF(*skilpadde*) = 1/4

TF(*jeg*, 1) = 1
TF(*liker*, 1) = 1/2
TF(*skilpadde*, 1) = 1/2

TF(*jeg*, 2) = 1
TF(*digger*, 2) = 1/2
TF(*skilpadde*, 2) = 1/2
TF(*din*, 2) = 1/2

TF(*w*, 3) = 1 for alle *w*

TF(*w*, 4) = 1 for alle *w*

TF(*w*, *q*) = 1 for alle w

len(2) = sqrt(1/16 + 1/16 + 1/64 + 9/64) = sqrt(2/16 + 10/64) = sqrt(9/32)
len(*q*) = sqrt(1/4 + 1 + 1/16) = sqrt(21/16)

dot(2, *q*) = (1/4) * (1/2) + (1/8) * (1/4) = 1/8 + 1/32 = 5/32

cos(2, *q*) = dot(2, 1) / (len(2) * len(q)) = 5 / (32 * sqrt(9/32) * sqrt(21/16))

Faktoren len(*q*) er konstant for alle dokumenter og kan således sløyfes uten at rangeringen av dokumentene påvirkes.

INDEKSKOMPRIMERING

VB koding:
1337 = (10 * 128) + 57
10 binary = 8 + 2 = 0001010
57 binary = 32 + 16 + 8 + 1 = 0111001
Prepender kontinuasjonsbit for å få to fulle bytes: 00001010 10111001

Gamma koding:
13 = 8 + 4 + 1 = 1101
offset = 13 binary med ledende 1 fjernet = 101
length = antall bits i 101 i unary code = 1110

gamma = length konkatenert med offset = 1110101

Kurt burde normalt gå for VB koding, med mindre plassforbruk er altoverskyggende. Gamma koding komprimerer gjerne litt bedre, men forskjellen er ikke så veldig stor og VB komprimerte data er mye raskere å dekomprimere.

Komprimerte data gir mindre data å lese fra disk, når/om vi først må gå til disk. Når mer data passer i minne og kan vi oftere serve resultater uten å måtte gå til disk. Caching gevinster.

Entropien er maksimal for en uniform distribusjon. Med N elementer får den da verdien log2(N).

K-GRAM RANKING

La A = 3gram(*foobar*) = {*foo*, *oob*, *oba*, *bar*}
La B = 3gram(*foodbar*) = {*foo*, *ood*, *odb*, *dba*, *bar*}

jaccard(A, B) = card(A AND B) / card(A OR B) = 2/7

La *x* angi en vilkårlig posisjon i dokumentet.

P(treff i posisjon x) = k/N
P(ikke treff i posisjon *x*) = $1 - k/N$
P(ikke treff i posisjon *x* for noen av de *h* treffene) = $(1 - k/N)^h$
P(treff i posisjon *x* for minst ett av de *h* treffene) = $1 - (1 - k/N)^h$

EVALUERING AV RELEVANS

p@20 = 6/20 = 0.3

R p@1 = 1/1	r@1 = 1/8
R p@2 = 2/2	r@2 = 2/8
N p@3 = 2/3	r@3 = 2/8
N p@4 = 2/4	r@4 = 2/8
N p@5 = 2/5	r@5 = 2/8
N p@6 = 2/6	r@6 = 2/8
N p@7 = 2/7	r@7 = 2/8
N p@8 = 2/8	r@8 = 2/8
R p@9 = 3/9	r@9 = 3/8
N p@10 = 3/10	r@10 = 3/8
R p@11 = 4/11	r@11 = 4/8
N p@12 = 4/12	r@12 = 4/8
N p@13 = 4/13	r@13 = 4/8
N p@14 = 4/14	r@14 = 4/8
R p@15 = 5/15	r@15 = 5/8

N p@16 = 5/16 r@16 = 5/8
N p@17 = 5/17 r@17 = 5/8
N p@18 = 5/18 r@18 = 5/8
N p@19 = 5/19 r@19 = 5/8
R p@20 = 6/20 r@20 = 6/8

Interpolering: Se etter største p(r) der r >= 0.33. Siden 33% av 8 = 2.64 er det fra @9 og utover. Største p er p@11 = 4/11 = 0.364.

Opplysningen om det er indeksert i alt 10000 dokumenter er ikke relevant i denne sammenhengen. ☺

NDCG står for ”normalized discounted cumulative gain”. Lettest å ta tak i når man leser det bakfra: Bør forklare hva som ligger i hhv ”cumulative gain”, ”discounted” og ”normalized”. En svakhet ved målet er at det blir lurt av (relevante) duplikater i resultatsettet.

SUFFIX ARRAYS

d1: *hermetikk*
d2: *mett*

ermetikk (1, 1)
etikk (1, 4)
ett (2, 1)
hermetikk (1, 0)
ikk (1, 6)
k (1, 8)
kk (1, 7)
metikk (1, 3)
mett (2, 0)
rmetikk (1, 2)
t (2, 3)
tikk (1, 5)
tt (2, 2)

For å finne alle forekomster av en substreng *met* kan vi nå gjøre binærsøk for å finne først match (*metikk* = (1, 3)) og scanne frem til og med *mett* = (2, 0). De aller flinkeste vil kanskje også fortelle om hvordan bruk av *lcp* kan brukes for å gi en liten speed-up.

YMSE OM ORDLISTER

Finne overlapp: Aho-Corasick algoritmen eller lignende, med litt NLP-awareness rundt hvor tokens starter/slutter. Datastruktur: En trie eller lignende som tillater en ”trie walk”, gjerne en automat der også andre ting enn prefiks er delt (for bedre komprimering og dermed muliggjøring av enda større ordlister.)

Edit distanse: Datastruktur for ordlista som over. Muligens med mulighet for å ha mer meta-data per node som kan brukes for å prune søkerommet raskere. Søk i denne ”trie-walkable” strukturen som beskrevet i Zhang/Merrett paper. Beregning per kolonne i edit tabellen kan gjøres litt raskere vha Ukkonens cutoff og/eller bit-parallelisme teknikker, avhengig av hvordan edit tabellen representeres.