

TextMatcher

Speciale af

Søren Dalby

Datanomuddannelsen
Lyngby Handelsskole
Programkonstruktion II
Foråret 2002

Revisioner

Dato	Ændringer
2002-06-09	Grundlagt
2002-06-10	Diverse tastefejl rettet
2002-06-16	Enkelte formuleringer og kommafejl er rettet
2002-06-27	Telefonnummer fjernet fra hoved – produktnavn (fra testmateriale) fjernet fra konklusion

Forord	4
Indledning	5
Generel sprogbrug.....	6
Illustrationer	6
Ordlister	7
Identificering og søgning af nøgleord	8
Identificering af nøgleord.....	8
Inddeling af vigtige ord i klasser.....	9
Stopliste.....	12
Søgninger	13
Automatisk dokumentklassifikation via klyngedannelser	14
Fravalg.....	16
TextMatcher kravspecifikation	17
Afgrænsninger	17
TextMatcher Arkitektur	18
Krav til TextMatcher som serverprogrammel	20
Sammenligningsmodul og Lagringsmodul	20
Kommunikationsmodul	22
Teknologivalg.....	24
Implementeringsplatform.....	24
Appendiks A WebCrawler (ordbogsgenerering)	26
Konklusion	28
Sammenligning 1 – Store dokumenter	28
Sammenligning 1 – Mindre dokumenter	28
Litteratur.....	30
Bilag.....	30

Forord

Jeg har længe været fascineret af datalingvistiske problemstillinger, som på jævn dansk betyder maskinel behandling af naturlige sprog. Min første indgangsvinkel til problematikken var, da jeg udviklede et tekstbehandlingsprogram i starten af 1990'erne. Da skulle jeg udvikle et orddelingsmodul og en stavekontrol (selv om sidstnævnte kun blev ved tanken).

For nylig var jeg involveret i nogle overvejelser om et system til elektronisk publicering af bladudgiveres artikler, og her var et af de centrale temaer en søgemaskine, som kunne finde artikler vha. søgeord. I denne forbindelse, fik jeg en ide om at identificere søgeord ved at fjerne almindelige danske ord (registreret i en ordbog), og at denne kunne opbygges ved at skanne Internettet.

Samtidig har jeg længe reflekteret over, hvordan to tekster egentligt kunne betragtes som relateret og jeg begyndte at koble internetordbogen sammen med denne problematik og det stod klart for mig, at jeg havde lyst til at udforske dette – for mig – ukendte land i mit datanomspecial.

Der er lagt uhyre mange timer i denne opgave – mange flere timer end hvis jeg havde kastet mig ud i et område med samme akademiske niveau, men hvor jeg var på kendt grund. Det har været anstrengelsen værd. Jeg har lært flere teorier og gjort mig flere erkendelser om dette komplekse område og er gået ud af denne opgave en del klogere, end da jeg bevægede mig ind den.

Jeg håber at den måske også kan bringe lys over hele eller dele af dette komplekse område for dem, der læser denne rapport.

Søren Dalby
Hillerød d. 9. juni 2002

Indledning

Tekstsøgning er med Internettet's fremkomst blevet en vital del af den måde, IT bruges på af såvel privatpersoner som virksomheder og institutioner, da det er en meget effektiv måde at fremsøge informationer på Internettet. Dertil kommer, at virksomheder og institutioner tit har brug for at søge i informationer internt.

De traditionelle systemer til tekstsøgning kaldes søgemaskiner (eller søgerobotter). De har gjort store fremskridt i de seneste år, og de tillader brugere, at angive et eller flere søgeord, hvorefter de fremviser de sider, som indeholder et eller flere af disse ord.

Men der er et andet aspekt af tekstsøgning, der ikke er dækket ind af denne traditionelle fremgangsmåde: At kunne forespørge om et dokument har relaterede dokumenter i den afsøgte informationsmængde. Hensigten med denne opgave er at efterprøve en række teorier, der understøtter denne proces. Der er dels tale om teorier hentet fra den datalingsvistiske teori og samt specialiseringer af- og supplementter til disse.

Det er en langt mere kompleks problemstilling end traditionel nøgleordssøgning, da identifikation af ordsæt, som indeholder beskrivende attributter (nøgleord) for denne tekst også skal detekteres maskinelt. Endeligt skal dette sæt sammenlignes med et tilsvarende sæt af nøgleord for en anden tekst og afgøre, om de sammenlignede tekster kan betragtes som relateret. Definitionen af *relation mellem dokumenter* er afhængigt af konteksten og skal derfor kunne være en varierende parameter til processen.

Opgaven undersøger Luhn's teori om, at ords hyppighed i en tekst kan bestemme et ords vigtighed for teksten, og om denne proces (samt sammenligning af ord mellem dokumenter) forbedres, ved at reducere ord til deres stamme. Det undersøges desuden om maskinel reduktion af ord til stamme er præcis nok til at være gavnlige for processen, når en algoritme til orddeling (udviklet af undertegnede i et tekstbehandlings-program) medvirker til en større præcision af denne reduktion

Endeligt undersøges det, om en ordbog, baseret på ords hyppighed på det danske domæne af Internettet, kan tjene som opslagsværk for ord, der - i kraft af deres hyppighed - kan betragtes som så almindelige i dansk sprogbrug, at de ikke er meningsbærende for teksten.

Opgaven er afgrænset til, kun at omhandle behandlingen af tekster, der er formuleret på dansk.

Validiteten af ovenstående efterprøves i en implementering af programmet TextMatcher, der løser denne opgave. Systemets arkitektur og indhold er også detaljeret beskrevet.

Efter råd fra vejleder, er implementeringen af algoritmerne (TextMatcher) ikke vedlagt som bilag.

Generel sprogbrug

Det er tilstræbt at holde sproget i et nemt forståeligt dansk. EDB termer er oversat til dansk med mindre, at det er skønnet, at ordet mister sin indlysende betydning for læseren, hvis den danske term bruges.

Forklaringer, der er af mere sekundær betydning, er skrevet som fodnoter, så læseren nemmere kan se bort fra disse.

Begrebet *nøgleord* og *vigtige ord*¹ beskriver ord, der har en væsentlig betydning for den tekst, som de indgår i – i modsætning til eksempelvis alm. danske ord som "og", "du", "i" osv., der indgår i de fleste dansksprogede tekster.

Læseren bør endvidere konsultere ordlisten på næste side, hvis der forekommer tekniske EDB termer, som kræver uddybning.

Illustrationer

Alle illustrationer er taget fra bogen INFORMATION RETRIEVAL.

¹ Grunden til, at der bruges to termer, skyldes, at nøgleord er det mest præcise i opgavens sammenhæng, men meget af det konsulterede materiale benytter den engelske term *important word*, hvilket er direkte oversat til *vigtigt ord* i de sammenhænge, hvor der refereres til baggrundsmaterialet.

Ordliste

Binær kode	Term for et program i eksekverbar form i modsætning til de kildetekster, som ligger til grund for programmet
Caching	Program konstruktion hvor data, der skal skrives til- og læses fra disken, opbevares i den interne hukommelse for intensive områder. En oversættelse til dansk (mellemlager) synes at være uheldig, så den oprindelige engelske term er benyttet.
JVM	Akronym for <i>Java Virtuel Maskine</i> . En JVM er et program, som kan afvikle den binære kode, der er genereret af en Java-oversætter. Et Java program afvikles m.a.o. på en JVM på samme måde som andre systemer afvikles på Windows, UNIX eller lignende platforme.
Maskine	Ordet Maskine benyttes som synonym for computer.
RDBMS	Relational Database Management System - et databasesystem som kan lagre og hente data via sproget SQL. Der er en række kriterier for, om et databasesystem kan betragtes som relationel, men det er med vilje udeladt her. RDBMS bruges i øvrigt også som akronym for <i>Remote Database Management System</i> , som betyder at databasesystemet afvikles på en anden maskine end klientprogrammet.
Server	Begrebet server dækker over maskiner, hvis formål er at afvikle programmer, der betjener andre programmer (i modsætning til programmer, der direkte betjenes af mennesker). En oversættelse til dansk (tjener-maskiner) synes at være uheldig, så den oprindelige engelske term er benyttet.
Transaktions database	Hvis en database er i stand til at betragte en serie af operationer som enhed, der enten er gennemført eller ikke – dvs. hvis en af operationerne fejler, bringes databasen tilbage til den tilstand, som herskede før operationens begyndelse – er det en transaktionsdatabase.
Tråde	Når et program afvikles, betegnes dette som en proces. En proces kan have flere samtidige afviklinger af programkode, og hver enkelte afvikling kaldes en tråd. Dette har mange fordele frem for blot at starte flere processer af samme program.

Identificering og søgning af nøgleord

I dette afsnit gennemgås en række lingvistiske teorier. Der bliver taget stilling til hver enkel teori, og hvis disse ikke er implementeret i TextMatcher efter deres præcise forskrifter, er den tillempede version beskrevet og argumenteret for.

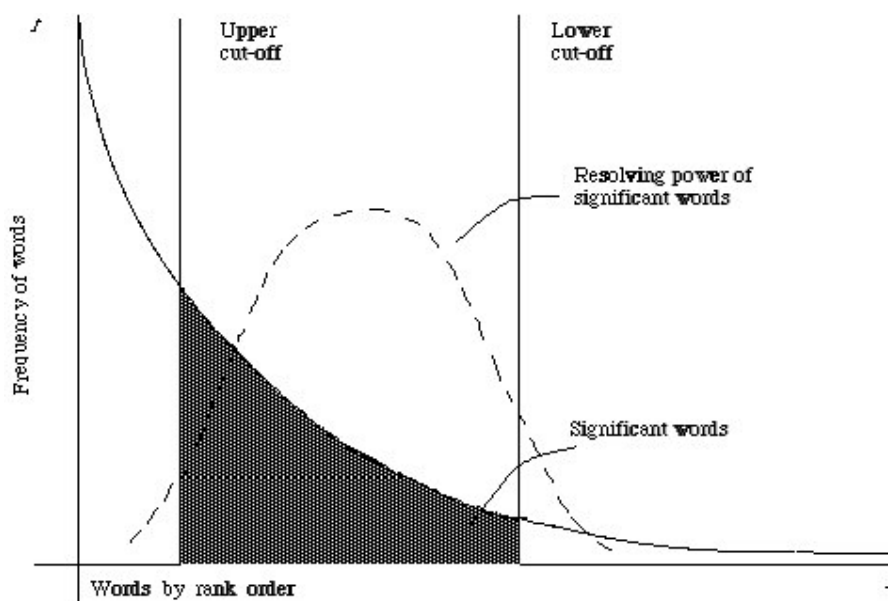
Endeligt nævnes teorier, der har været under overvejelse, men som af den ene eller anden årsag er valgt fra.

Identificering af nøgleord

Dette afsnit handler om at detektere ord, der bør betragtes som en del af det sæt, der beskriver et dokument i forhold til andre dokumenter. Ord, som er en del af dette sæt, kaldes i denne redegørelse for nøgleord eller vigtige ord, og sættet kaldes for for nøgleordssættet.

Luhn har formuleret en teori om, at et ords hyppighed i en tekst, er bestemmende for et ords vigtighed. Et af hans tidligste værker fremstiller følgende hypotese: "It is here proposed that the frequency of word occurrence in an article furnished a useful measurement of word significance".

Det skal ikke forstås således, at ord enten skal være meget hyppige eller det modsatte. Luhn definerer en øvre og nedre grænse for ords hyppighed. Ord under den nedre grænse anses for ubetydelige, og ord over den øvre grænse blev anset for almindelige og derfor ikke meningsbærende. Dette er illustreret i følgende graf:



Ord, hvis hyppighed har samme afstand til øvre og nedre grænse, er de mest betydende og at dette falder til noget nær 0 i nærheden af øvre eller nedre grænse. Det er ikke muligt, at definere øvre og nedre grænse ud fra faste kriterier - disse kan kun fastsættes ud fra eksperimenter.

Luhn supplerede listen af vigtige ord med den sætning i teksten, der havde flest forekomster af vigtige ord, men denne algoritme er ikke inkluderet i TextMatcher.

Salton og Yang har arbejdet videre med Luhns metoder og lavet undersøgelser af ords hyppighed i såvel enkelte dokumenter som grupper af dokumenter. De fandt ud af, at hvis ordet havde en høj hyppighed i mange dokumenter, var dette et mindre vigtigt ord for det enkelte dokument. Et ord bør m.a.o. kun betragtes som et vigtigt ord, hvis det samtidigt ikke betragtes som et vigtigt ord for størstedelen af det samlede dokumentmateriale.

De konkluderede også, at hvis tolerancetærskelen er høj (lavt sat nedre grænse og højt sat øvre grænse) vil et givent ord blive fundet i flere dokumenter, men desværre vil præcisionen af, om et ord er vigtigt eller ej, også være mindre præcis.

Ord skal altså ligge inden for en ramme, som kan variere i størrelse, men som tager udgangspunkt i midten af frekvensområdet og som ikke har høj forekomst i alle dokumenter. Hvis rammen bliver for stor, vil flere ikke-nøgleord blive betragtet som medlem af nøgleordsættet. Omvendt, hvis rammen er for lille, vil visse nøgleord ikke blive betragtet som medlem af dette sæt.

Implementeringen af denne algoritme i TextMatcher viser, at denne algoritme er meget præcis for større tekster, men til gengæld falder i præcision, jo mindre teksten er.

Inddeling af vigtige ord i klasser

C. J. van RIJSBERGEN [1] anviser, at man med fordel kan reducere vigtige ord til stammen af ordet. Alle vigtige ord med samme stamme siges at tilhøre samme klasse. Denne proces kaldes derfor klassificering.

At kunne bestemme et ords stamme med 100% sandsynlighed, kræver en præcis grammatisk analyse af ordet, hvilket ikke er muligt maskinelt uden en ordbog, der dækker alle ord i sproget, deres bøjninger samt alle undtagelser for disse bøjninger (uregelmæssige ord). En sådan ordbog har desværre ikke været tilgængelig i dette projekt.

Derfor er en metode benyttet, som prøver at fjerne alle standard-ender fra ordet. Af de standardender, der passer til slutningen af ordet, vælges den længste.

For at bestemme om fjernelse af en endelse er lovlig, analyseres stammen. Den skal typisk bestå af mindst to bogstaver² og indeholde en vokal.

Endvidere kan en algoritme, som jeg har udtænkt i forbindelse med udviklingen af et orddelingsmodul i et tekstbehandlingsprogram, benyttes til at bestemme om en endelse kan fjernes.

Den baserer på ideen om, at visse bogstaver kun kan optræde efter et andet bogstav, hvis bogstaverne forekommer i hver sin stavelse. Eksempelvis kan bogstavet D kun efterfølge B, hvis de optræder i hver sin stavelse (eksempelvis ab-di-ce-re). Det samme gør sig gældende for alle dobbeltkonsonanter.

Stammen kan derfor analyseres for, om der optræder en vokal både før og efter en sådan bogstavskombination. Den samlede liste, som jeg analyserede mig frem til, ser således ud:

Bogstav	Efterfulgt af ét af følgende bogstaver
B	CDFGHKMNPQVXZ
C	BDFKMNPXT
D	BCFGKLMNPQRTXZ
F	BCDGKMNPQTVXZ
G	BCDFKMNPQTXZ
H	BCDFGKLMNPQRTXZ
J	BCDFGKLMNPQ
K	BCDFGMPQXZ
L	BCQTXZ
M	BCDFGHKNPQR
N	BCJMPQRVXZ
P	BCDGHKMNPQVXZ
Q	BCDFGHJKLMNPRTUVXZ
R	BCDFGJMNPQRVXZ
S	BCDFG
T	BCDFKLMNPQXZ
W	BCDFGHJKLMNPQRTVXZ
X	BCDFGHJKLMNPQRTVXZ
Z	BCDFGHJKLMNPQRTVXZ

Denne algoritme påstås ikke at være perfekt, men den har vist sig særdeles effektiv ved test, hvor en række orddelinger blev testet i tekstbehandlingsprogrammet³.

² Det betyder at ordet *å* (de to bække løber sammen til en *å*) eller *ø* (Bornholm er en dejlig *ø*) vil ikke blive klassificeret. Ordet *i* (eksempelvis *Det er min fødselsdag i dag* eller *Kommer i til min fødselsdag*), vil aldrig have en endelse, så det vil ikke volde problemer.

³ hvilket kan efterprøves i via orddelingsmodulet i tekstbehandlingsprogrammet Kvik-Tekst 3.0, som kan downloades fra <http://home19.inet.tele.dk/tjj/kt01.htm>

Hvis denne stammevalideringsalgoritme ikke kan godkende ordet efter at endelsen er fjernet, afvises denne endelse og der forsøges med en ny endelse. Da de længste endelser prøves først, vil dette give den størst mulige reduktion.

De endelser, som TextMatcher opererer med, følger endelserne beskrevet i afsnittet *Grammatisk oversigt* i *Politikens nudansk ordbog* (bilag A) for substantiver, adjektiver og verber. Dertil kan føjes, at hvis ordet minus standard-endelse ender på en dobbeltkonsonant, fjernes den ene af konsonanterne, da både adjektiver og verber kan tilføje dobbeltkonsonanter mellem stammen og endelsen (eksempelvis *langsomm-ere*).

Stamme-analysen er ikke perfekt. Dette illustreres godt med ordet *Komet*. Dette er stammen af et substantiv, som betegner et objekt i universet, men regelsættet vil reducere ordet til *kom*, hvilket vil blive opfattet som et verbum (*at komme*) i imperativ. Denne fejl er alvorlig og kunne undgås, hvis en ordbog var tilgængelig⁴, men det var desværre ikke tilfældet.

Endeligt er der foretaget nogle valg omkring sammensatte ord og verbalsubstantiver. Verbalsubstantiver omlægges til verbet's stamme, da det er vurderet, at sammenligningen på "verbumformens" stamme, er ønskelig. Dette kan også afstedkomme fejl, da ord, hvis stamme ender på *n*, ikke får tilføjet *ning* men blot *ing* og derfor vil fjernelsen af endelsen *ning* være forkert (eksempelvis *beregning* reduceres til *bereg*, når *ning* fjernes, hvilket er forkert).

De beskrevne fremgangsmåder har en høj fejlfrekvens, men det skal ses i lyset af, at fejlen vil ske på begge sider af lighedstegnet⁵ og dermed vil den ovenstående fejl, hvor *beregning* reduceres til *bereg* ikke være fatal, hvis alle bøjninger af ordet vil resultere i stammen *bereg* og hvis ingen andre ord reduceres til dette. Til gengæld må eksemplet, hvor *komet* omsættes til *kom*, betragtes som en alvorlig fejl, da ordet skifter betydning. Denne type fejl er desværre umulige at undgå uden en omfattende ordbog.

Tests i TextMatcher giver anledning til at konkludere, at klassificeringen (reduktionen til stamme) er præcis nok til at kunne bruges i denne sammenhæng og at stammevalideringen vha. algoritmen fra Kvik-Tekst højner kvaliteten af klassificeringen ganske betragteligt.

Det er en nærliggende tanke, at en så usikker fremgangsmåde som den implementerede klassificering burde kunne deaktiveres i TextMatcher. Det er dog vurderet, at fremgangsmåden vil være ubrugelig uden denne klassificering, så det må betragtes som obligatorisk (eller det mindste onde om man vil).

⁴ En ordbog kunne afsløre, at dette verbum tilføjer en ekstra konsonant mellem stamme og endelse i perfektum participium

⁵ Med dette menes, at denne fejlagtige stamme vil optræde på samme måde i de to sæt af ord, som ligger til grund for en sammenligning mellem 2 dokumenter.

Stopliste

Stoplister indeholder ord, som skal vælges fra, ud fra en forudgående viden om, at disse ord er så hyppige, at det ikke giver mening at måle på dem. Disse indgår m.a.o. automatisk i en gruppe af ord, som vælges fra, fordi de ligger over den øvre den grænse.

Stoplister har en meget høj vægt i den valgte løsningsmodel, da den inkluderer ordfrekvens fra en skanning af 113.000 danske internetsider⁶. Denne statistik er så omfattende, at den har større betydning for resultatet, end C. J. van RIJSBERGEN [1] ligger op til.

Denne automatiske indsamling af ord, har den fordel, at den registrerer ord i alle de bøjninger, som forekommer hyppigt på Internettet. Da det danske sprog har mange uregelmæssigheder, har dette primitive princip en fordel frem for et system, der kun har ordstammer registreret og kombinerede disse med grammatiske regler.

Stoplisterne er delt op i en obligatorisk ordbog og en tillægsordbog. I TextMatcher kan tillægsordbogen vælges fra af brugeren.

Den obligatoriske ordbog indeholder ord, som pr. definition ikke er meningsbærende (eksempelvis *af, at, der, de, og, som* og *der*).

Tillægsordbogens indeholder ord, hvor det i højere grad er en vurderingssag, hvorvidt ordet skal fjernes eller ej. En kritisk gennemgang af tillægsordbogens indhold afslører, at den er "ujævn" - forstået således, at det er svært for en bruger at vide, om et givent ord er med eller ej. Det påvirker også mængden af ord i nøgleordssættet, og netop denne indgår i sammenligningsalgoritmen (der beskrives senere) som divisor.

Ordene i tillægsordbogen reduceres til stammen, mens ordene i den obligatoriske ordbog checkes som de er. Dette synes at modstride ovenstående argument om uregelmæssige bøjninger, og det er også korrekt, at eksempelvis *burde* ikke kan reduceres korrekt til *bør*, men denne relation kan alligevel ikke ske uden en grammatisk ordbog og størstedelen af ordene, vil få en mere præcis og sammenlignelig form.

Teorien bag fremgangsmåden, hvor ordbogen blev dannet på grundlag af ord indhentet fra Internettet, var faktisk, at alle hyppige ord kunne registreres i en form, så frasorteringen af disse, kunne ligge til grund for en identificering af nøgleord alene. Dette har vist sig meget forkert. Dertil er menneskers måde at formulere tekst på for nuanceret. Men selvsamme konklusion hjælper også positivt. Den selvsamme rigdom i ord medfører nemlig, at de ord som to tekster har tilfælles typisk er beskrivende ord (såsom fag-termer, personer, virksomheder, produktnavne og lign) . Så TextMatcher har ved denne fejlagtige hypotese – i en helhedsbetragtning - vundet på gyngerne hvad der er tabt på karrusellen.

⁶ Her af blev kun 1/3 (maskinelt) betragtet som sider med sammenhængende dansk tekst og indholdet de andre 2/3 sider blev derfor ignoreret.

Tillægsordbogen bør forbedres, så den strukturelt afspejler et frekvensniveau for medtagne ord. Dens nuværende form påvirker resultatet positivt men uforudsigeligt.

Denne konklusion må dog ikke opfattes som generel for ordbogsprincippet, men kun for de faktiske informationer, som ordbogen består af som følge af fremgangsmåden med Internetskanningen.

Søgninger

Når der er udtrukket en serie af nøgleord i to tekster, skal en algoritme afgøre, om disse to tekster er at betragte som sammenhængende.

C. J. van RIJSBERGEN [1] anviser følgende algoritme, hvor Q er sættet af vigtige ord i den ene tekst og D er sættet af vigtige ord i den anden tekst:

$$M = \frac{|D \cap Q|}{|Q| + |D|}$$

Resultat er baseret på den delmængde af vigtige ord, som de 2 dokumenter har til fælles set i forhold til den samlede sum af vigtige ord i D og Q.

Resultat M skal være over en vis grænse før der konkluderes, at der er en relation mellem D og Q.

Den implementerede algoritme er inspireret af ovenstående, idet den betragter fællesmængden af søgeord i forhold til en tilfældig foreningsmængde af søgeord, men vægten af det enkelte søgeord, er bestemt ved ordets vigtighed⁷ i begge tekster. Disse multipliceres med hinanden og adderes til den samlede score. Det betyder, at hvis ordet opfattes som vigtigt i begge tekster, får det stor indvirkning på den samlede score, hvis ordet er vigtigt i den ene og ikke-vigtigt i det andet, får det en middel indflydelse på den samlede score etc.

⁷ Efter Luhn's algoritme som betragter ordets vigtighed ud fra ordets frekvens

I pseudokode ser det således ud

```
Dn = first stem in D
while more elements in D
    does a matching stem exist in Q
        points = points + importance( matching stem( Qn ) ) * importance( Dn )
    Dn = next stem in D

score = ( points * 1000 ) / ( elements in D + elements in Q )
```

Sammenligningsalgoritmen

Årsagen til at resultatet multipliceres med 1000 er, at denne heltalsberegning ikke ville være tilstrækkelig præcis, hvis decimalerne blot blev afskåret.

I TextMatcher kan brugeren bestemme grænsen for resultatet – det vil sige om et givent resultat skal opfatte dokumenterne som relateret eller ej.

Implementeringen af denne algoritme i TextMatcher viser, at denne algoritme er meget præcis for større tekster, men til gengæld falder i præcision, jo færre ord der er i teksten.

Automatisk dokumentklassifikation via klyngedannelser

Automatisk dokumentklassifikation via klyngedannelser registrerer mønstre mellem dokumenter.

Princippet er beskrevet af C. J. van RIJSBERGEN [1] og baserer sig på, at alle dokumenter sammenlignes og en score beregnes for hver sammenligning.

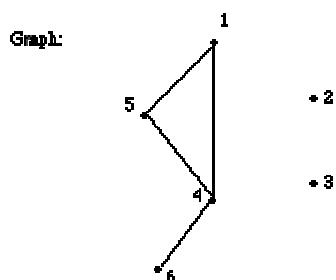
Helt specifikt sammenlignes hvert dokument med alle andre dokumenter. Der laves en scoreliste og kun sammenligninger med en score over en vis grænse, betragtes som medlem af klyngen. I nedenstående illustrerede eksempel vises en sammenligning af 6 dokumenter. M fastsættes til en grænse (threshold) på 0.89 og alle sammenligninger over denne grænse forbindes i en graf, da dokumenterne som sammenligningerne baserer sig på, betragtes som en klynge.

Objects: {1,2,3,4,5,6}

Similarity matrix:

1						
2	.6					
3	.6	.8				
4	.9	.7	.7			
5	.9	.6	.6	.9		
6	.5	.5	.5	.9	.5	
	1	2	3	4	5	6

Threshold: .89



Eksemplet viser at forholdet mellem dokumenterne

- 1 og 5
- 1 og 4
- 5 og 4
- 4 og 6

ligger over grænsen på 0.89. Derfor knyttes de sammen i en graf, der kan ligge til grund for en søgning.

En klynge kan da tilknyttes en profil af søgeord og aktiveres som et samlet hele.

Dokumentklassifikation via klyngedannelser er implementeret i TextMatcher for at generere en rapport over scorefordeling, som tjener til at hjælpe med at fastslå det korrekte niveau for sammenligning. Det kunne have været implementeret som en alternativ søgestrategi, men det er valgt fra pga. tidspres.

Der foreligger derfor ikke tests, som kan ligge til grund for egentlige konklusioner, om hvorvidt dokumentklassifikation via klyngedannelser ville påvirke søgningsresultatet og tidsforbruget i TextMatcher. Da klyngen kun registrerer de ord, som klyngens dokumenter har til fælles, er det nærliggende at forvente en forringelse af søgeresultatet. Til gengæld må det forventes, at ydeevnen ved denne søgestrategi vil være betydeligt højere end den implementerede

fremgangsmåde. Det forudsætter dog, at der er et tidsvindue med få eller ingen forespørgsler, hvor disse klynger kan dannes.

Fravalg

C. J. van RIJSBERGEN [1] refererer til (men uddyber ikke) andre og mere effektive algoritmer end de benyttede, men disse baserer sig på indgående statistisk materiale og omfangsrige ordbøger og er valgt fra på grund af dette.

Latent Semantic Indexing/Analysis synes derudover at være en effektiv og ofte benyttet konstruktion, men er valgt fra, da det ikke var muligt at finde materiale, der gav en praktisk og anvendelsesorienteret forklaring.

TextMatcher kravspecifikation

TextMatcher skal kunne honorere et krav om at tekster skal kunne sammenlignes ud fra brugerdefinerede søgningstrategier.

Som i andre søgnings-problematikker kan søgningstrategien enten være stram eller løs og TextMatcher forsøger at honorere følgende betragtninger:

At hvis strategien er stram, opnås en snæver selektering af relaterede dokumenter, hvilket betyder, at alle selekterede dokumenter er relateret, men til gengæld risikeres det, at ikke alle relaterede dokumenter medtages.

At hvis strategien er løs, opnås en bred selektering af relaterede dokumenter, hvilket betyder, at alle relaterede dokumenter sandsynligvis er selekteret, men til gengæld risikeres det, at ikke-relaterede dokumenter medtages i selektionen.

Derfor er det nødvendigt, at brugeren selv kan sætte kriterierne for denne selektion og at faciliteter til at finde det rigtige niveau tilbydes.

Det mest almindelige brugsmønster er, at en bruger forespørger om et dokument har relaterede dokumenter og få disse retur. Som led i en typisk forespørgsel vil det forespurgte dokument fremover indgå i det materiale, som der søges i. TextMatcher skal dog kunne honorere kravet om, at søgninger kan foretages uden at dokumenter suppleres, og at dokumenter kan suppleres uden at en søgning foretages.

TextMatcher skal kunne afvikles på en servermaskine og leve op til de krav, som stilles til et sådan system. TextMatcher skal have et kommunikationsmodul, der kan modtage dokumenter fra og aflevere dokumenter til de programmer, som TextMatcher servicerer (klientprogrammer). TextMatcher skal kunne betjene flere samtidige klientprogrammer og disse skal ikke nødvendigvis være placeret på samme maskine som TextMatcher.

Afgrænsninger

TextMatcher mangler faciliteten til at sortere resultatsættet (et eller flere dokumenter) af en søgning efter graden af lighed med det forespurgte dokument.

Desuden er det erkendt, at den nuværende implementering af TextMatcher ikke fungerer optimalt på meget små dokumenter. Dette forventes dog muligt at forbedre ved en specialisering af sammenligningsalgoritmen samt forbedring af tillægsordbogen.

TextMatcher Arkitektur

Da såvel kommunikationsmodulet⁸ som sammenligningsmodulet⁹ og lagringsmodulet¹⁰ kan antage mange former, er det vigtigt at have en løs kobling mellem modulerne, så de kan skiftes uafhængigt af hinanden. Desuden er der lagt vægt på, at sammenligningsmodulet kan fungere uafhængigt af denne kontekst.

Sammenligningsmodulet består bl.a. af nøgleordsmodulet¹¹, der igen har statistikmodulet¹² og klassificeringsmodulet¹³ som delmængde.

For at skabe en kobling mellem dét, som kommunikationsmodulet har afleveret og dét, som sammenligningsmodulet skal bruge, er en komponent, som kan læse den afleverede fil og omsætte det til en liste af ord, nødvendig. Dette gøres via ordudtrækningsmodulet, som i den nuværende implementering kun kan læse tekstfiler.

Klassificeringsmodulet er eksplicit skrevet til det danske sprog og ordbøgerne er også specifikke for det danske sprog.

Den løse interne kobling mellem modulerne (de er ikke binært adskilt, men hele tanken bag implementeringen har netop været at holde dem adskilte) gør, at en videreudvikling af TextMatcher kunne bestå af en lille kerne, som læser en konfigurationsfil, som angiver hvilken kommunikationsform, filtyper, lagringsform og sprog en given installation af TextMatcher består af¹⁴.

Denne samlede arkitektur er illustreret ved følgende simplificerede diagram:

⁸ det modul som kan modtage forespørgsler og sende svar

⁹ det modul som kan finde relaterede dokumenter

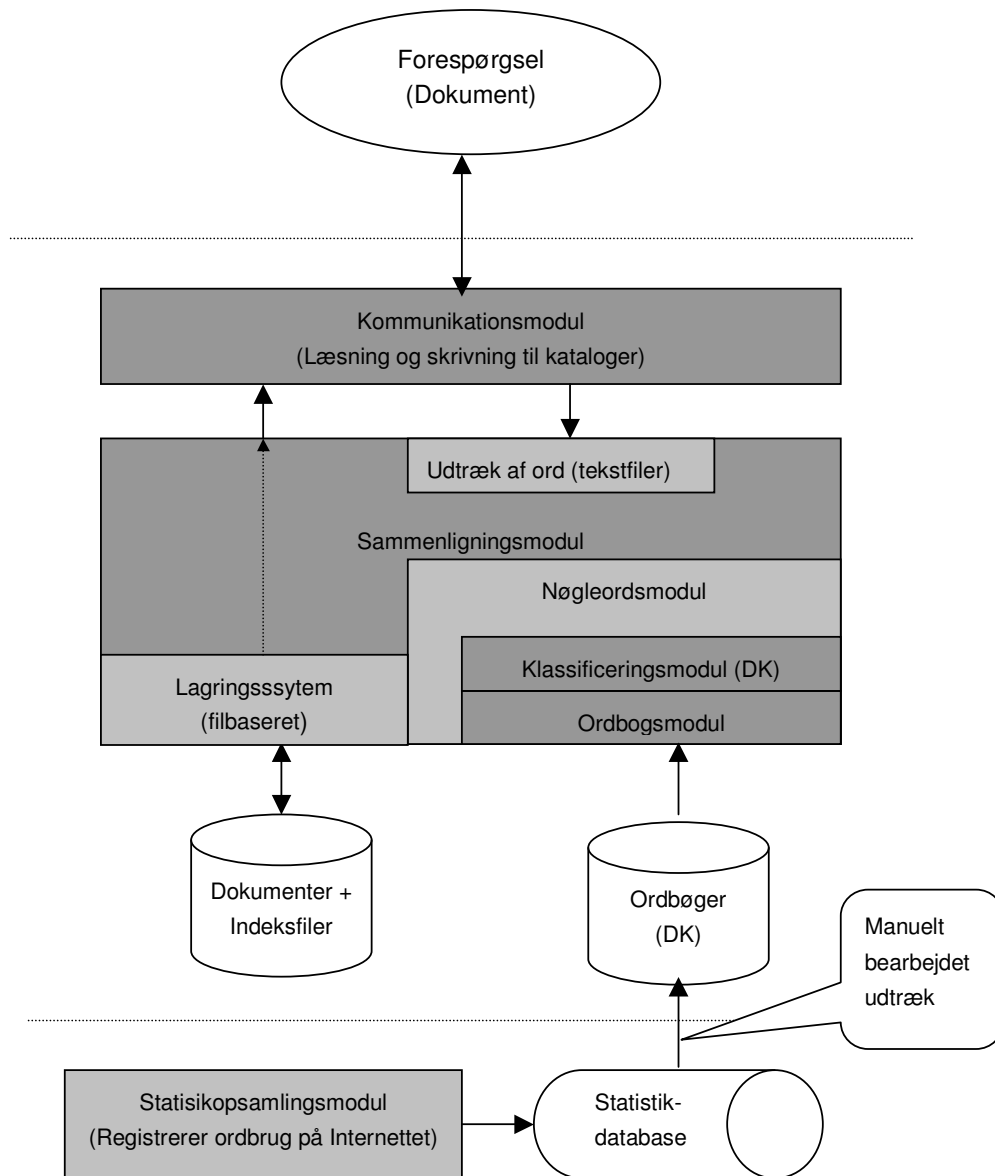
¹⁰ det modul som lagre og tilvejebringe dokumenter

¹¹ det modul som kan identificere og udtrække nøgleord

¹² som fjerner ord, der ikke er meningsbærende, via 2 ordbøger

¹³ der reducerer ord til dettes stamme

¹⁴ Der er en stram kobling mellem lagringsmodulet og brugen af flade tekstfiler i den nuværende implementering, men denne kan elimineres via en vis indsats. Dette er valgt fra pga. tidspres.



Krav til TextMatcher som serverprogram

TextMatcher er skrevet til at køre på servermaskiner, og det stiller visse krav.

Det kan afvikle samtidige arbejdsopgaver i et konfigurerbart antal tråde og kan liste aktivitetsbeskrivelser (logninger) i en konfigurerbar detaljeringsgrad. Alle fejl bliver udførligt beskrevet og hvis programmet lukkes ned som følge af en fatal fejl, afsluttes programmet med en fejlkode, som kategoriserer¹⁵ fejlen for de omgivelser, som kaldte programmet. Der er lagt vægt på diagnosticering af fejl, så operatøren ved, hvordan en fejlsituation skal håndteres.

TextMatcher har en konsol-facilitet, så operatøren stoppe-, genstarte-, og skifte logningsniveau for- en kørende version af TextMatcher.

Endeligt checker TextMatcher for om systemet allerede er kørende – og vil i så fald ikke starte op.

Hvis TextMatcher detekterer, at sidste afvikling af programmet ikke blev lukket kontrolleret ned, tvinger TextMatcher operatøren til at køre en re-indeksering (mere herom senere) af data.

Sammenligningsmodul og Lagringsmodul

Når relaterede dokumenter for et dokument skal findes, sammenlignes dette med alle registrerede dokumenter sekventielt og sammenligningsalgoritmen (beskrevet i kapitlet *Søgning*) benyttes til at vurdere om dokumenterne er relateret.

Denne sekventielle søgning, hvor sammenligningsalgoritmen aktiveres for hvert registrerede dokument, er relativ tung for meget store mængder af dokumenter, men der synes ikke at være alternativer. Normalt benyttes søgefaciliteterne i relationelt databasesystem (RDBMS), men det synes ikke at byde på nogen muligheder i denne sammenhæng. Selv om mange af disse har muligheden for at udlede fællesmængden af to sæt ved kommandoen UNION, løser dette ikke problemet mht. at scoreberegningen skal foretages. Som akademisk betragtning kunne sammenligningen løses i en RDBMS, men afviklingshastigheden ville gøre dette helt ubrugeligt.

Selv om det ikke er et funktionelt krav i sammenligningsmodulet, er det af hensyn til afviklingshastigheden nødvendigt, at processen, hvor ord udtrækkes af teksten, hvor ord reduceres til stammen og hvor visse ord fjernes via ordbøger mm., gøres én gang for alle og lagres. Denne proces kaldes indeksering.

Ved ændringer af visse parametre i konfigurationsfilen (se næste afsnit) eller i tilfælde, hvor indeks kan være blevet inkonsistente med det oprindelige datasæt som følge af strømsvigt

¹⁵ Eksempelvis *Fil ikke fundet, Fejl ved læsning/skrivning af fejl, Fejl i ordbøger, Fejl i data* osv.

eller maskinnedbrud¹⁶, indeholder TextMatcher en re-indekseringsfacilitet, der re-genererer indeks på basis af dokumenterne.

Ved søgning indgår 3 parametre i sammenligningen af dokumenter, som brugeren skal tage stilling til:

- 1) Nedre grænse for ordhyppighed (se Identificering af nøgleord)
- 2) Øvre grænse for ordhyppighed (se Identificering af nøgleord)
- 3) Scoregrænse (threshold) for om et dokument er relateret eller ej.

Da parameter 1 og 2 har indflydelse på den proces, der selekterer ord til indeksfilen, checker TextMatcher for, om disse er ændret i konfigurationsfilen, og kræver en reindexering i så fald. Dette gøres via en sammenligning med en systemkonfiguration, der angiver, hvad de gældende grænser var ved seneste kørsel.

Det implementerede lagringsmodul benytter filer til at lagre dokumenterne og tilhørende indeksfil. Det er skønnet, at det ikke var attraktivt at involvere et RDBMS for en simpel lagringsproblematik, da dette også stiller krav til viden om og licenser¹⁷ til sådanne systemer. Visse virksomheder vil dog nok foretrække registrering i et RDBMS, da data i så fald kan indgå i en samlet strategi for sikkerhedskopier af denne. Desuden giver transaktionsdatabaser¹⁸ en yderligere fordel, da konsistens mellem dokumenter og indeks kan sikres af disse.

Når et dokument omsættes til en liste af ord, udregnes en signatur. Når et dokument ønskes tilføjet, checkes det om et dokument med samme signatur allerede er registreret. I så fald bliver dokumentet ikke tilføjet.

Ved tilføjelse af et nyt ord, trækker TextMatcher et sekvensnummer, og benytter dette til navngivning af filen.

Implementeringen af TextMatcher læser alle indeksfiler ved opstart og deres indhold bevares i en struktur i den interne hukommelse, men mængden af ord, som optræder i denne liste er begrænset selv for store dokumenter.

Hvis der er tale om installationer, der har tusindvis af dokumenter registreret og som hyppigt forespørges, stiller TextMatcher visse krav til såvel mængden af intern hukommelse som regnekræften i centralenheden,

¹⁶ Det er egentligt kun nødvendigt, hvis afbrydelsen af TextMatcher rammer et kritisk (og meget snævert) tidsvindue, hvor en opdatering af indeks skrives til disk, men implementeringen af TextMatcher følger et forsigtighedsprincip, hvor risikoen for, at data måske er kompromitteret, medfører en reindexering.

¹⁷ Der eksisterer gratis RDBMS systemer såsom MySQL og Postgress

¹⁸ I langt de fleste tilfælde er et RDBMS og en transaktionsdatabase én og samme ting, men der eksisterer RDBMS'er som ikke er transaktionsdatabaser og transaktionsdatabaser som ikke er RDBMS'er

Hvis mængden af forespurgte dokumenter er lille og/eller frekvensen af forespørgsler er lav, kan TextMatcher baseres på en løsning, hvor indeksdata læses og skrives via filer i stedet for den interne hukommelse. Det er i den forbindelse ikke muligt at bruge caching mekanismer, da alle registrerede dokumenter skal konsulteres for hver forespørgsel.

Det ville være relevant og nærliggende at implementere begge løsninger og måle på forskellen, men dette er valgt fra pga. tidspres.

I parentes skal det nævnes, at dokumentets oprindelige identifikation¹⁹ gemmes i indeksfilen. Det er en vigtig detalje, da denne information ikke kan genskabes via dokumentet, da dokumentet lagres via et sekvensnummer. I en ekstrem fejlsituation kan denne information altså mistes, men reindexering læser de gamle indeksfiler for denne information, før nye indeksfiler skrives. Rapporteringsmodulet bruger denne id, men den er ikke vital for TextMatchers kernefunktionalitet. Navngivningen af en relateret fil, som returneres til brugeren i den implementerede kommunikationsmodul, bruger også denne id som delkomponent i navngivningen af filnavnet, men det er ikke vitalt, da sekvensnummeret på dokumentet er unikt og er primært implementeret for at lette test-situationer.

Kommunikationsmodul

Kommunikationen er implementeret som et meget simpelt system, der læser en forespørgsel i form af en dokument fra et filkatalog og afleverer de relaterede dokumenter i et andet filkatalog. Denne model er valgt, fordi den er simpel at implementere og fordi de fleste programmer kan interagere med andre programmer på denne måde.

Der checkes for, om filen som indeholder forespørgslen er mere end 5 sekunder gammel, så det ikke risikeres, at en fil læses af TextMatcher, mens den er ved at blive skrevet.

For at undgå konflikter ved flere samtidige forespørgsler, vil alle afleverede dokumenter (resultatdokumenter) være navngivet efter det indgående dokument efterfulgt af resultatdokumentets sekvensnummer og evt. resultatdokumentets oprindelige id.

Mens resultatdokumentet skrives, navngives det med et punktum først i filnavnet og omdøbes til at det rigtige navn, når skrivningen er færdig. Det skal derfor dokumenteres, at filer, der begynder med punktum, ikke skal læses i resultatkataloget²⁰.

Et mere ambitiøst kommunikationsmodul kunne eksempelvis udveksle XML data over HTTP. Det er ved at være standard at implementere systemer i Java, udveksle data i XML formatet og lade den fysiske kommunikation foregå over HTTP. Det skyldes at XML skaber portable data²¹ og HTTP kan bruge Internettet som bæremedie uden at blive blokeret af

¹⁹ i det implementerede kommunikationsmodul vil dette være filens navn, men andre løsninger vil ikke kunne basere sig på dette

²⁰ Dette er standard for fillæsninger i programmer, der afvikles under UNIX platformen.

²¹ Dette er en meget simplificeret sandhed. XML skaber en fælles referenceramme for formatering af data, men der skal foreligge egentlige udvekslingsstandarder, før dette er helt korrekt.

sikkerhedssystemer – såkaldte *firewalls*. En standard for denne kommunikationsform (med tilhørende gratis implementering) kaldes SOAP og den ligger til grund for bl.a. Web Services, som er en meget omfattende standard for, hvordan systemer skal interagere med hinanden. TextMatcher ville være en perfekt Web Services komponent, men det er valgt fra pga. hensynet til kompleksitet.

Hvis TextMatcher ønskes benyttet uden integration med andre systemer, er det oplagt at lave et kommunikationsmodul, der kan læse dokument fra Internet mail via protokollen POP3 og sende relaterede dokumenter tilbage til afsenderen af denne mail. Dette vil kunne finde praktisk anvendelse uden nogen form for systemintegration og Internettet vil kunne bruges som infrastruktur.

Teknologivalg

Implementeringsplatform

TextMatcher er implementeret i Java, der er en standardplatform (et programmeringssprog og afviklingsmiljø) fra Sun Microsystems samt en lang række andre udbydere.

Java er meget velegnet til denne type opgaver, da :

- Sproget har et passende abstraktionsniveau og der medfølger megen funktionalitet i afviklingsmiljøet i form af klasse-biblioteker mm.
- Ved udarbejdelsen af programmer med flere tråde, er mange komplekse problemstillinger løst i JVM'en, som selve programmet ellers skal tage højde for.
- Samme system kan afvikles på mange forskellige operativsystemer, da systemet afvikles på en JVM.
- Der er implementeret en lang række af standardteknologier til denne platform og mange af disse er gratis.
- Mange udviklere behersker dette sprog, da det er meget udbredt på såvel undervisningsinstitutioner som i virksomheder.
- Java er ekstremt stærk hvad sene bindinger af binær kode angår. Det tidligere formulerede fremtidsperspektiv om, at TextMatcher kunne indlæse sine moduler via en konfigurationsfil, er i stor grad baseret på Java's faciliteter på dette område²²

Der er også ulemper ved Java platformen:

- Programmer, udviklet på Java platformen, er ofte langsommere end tilsvarende løsninger udviklet i sprog, der genererer kode, som afvikles direkte på maskinens centralenhed.
- Da en JVM er en standard, der benyttes på tværs af maskinelle arkitekturer og operativsystemer, lider Java standarden til en vis grad af, at kun mindste fællesnævner mellem disse gælder.
- Det er en forudsætning, at en JVM er installeret på maskinen (disse medfølger dog ofte fra leverandøren af operativsystemet)
- Selv om der er lavet JVM'er til stort set alle bredt tilgængelige operativsystemer, divergerer disse i hastighed og faciliteter²³
- Det er nemt at konvertere binær kode til information, der gør andre programmører i stand til at gennemskue programmets konstruktion. Dette er en stor ulempe, da kommercielle interesser kan kompromitteres – eksempelvis ved at den binære kode, som en programudbyder har indsat til at checke et brugercertifikat, kan gennemskues eller helt fjernes.

²² Windows software kan dog noget tilsvarende, hvor såkaldte *Dynamic Link Libraries* kan bruges på samme måde.

²³ Eksempelvis er IBM meget bagud med JVM frigelser til deres operativsystemer i forhold til Java standarderne. Kvaliteten er dog eksemplarisk (når de endeligt frigives).

Der er andre problemer forbundet med klient-programmel skrevet i Java, men da projektet er et program beregnet til server-maskiner, vil det ikke blive uddybet her.

Selv om ulemperne ved Java platformen er betydelige, er det den bedste platform til udvikling af serverprogrammel. Java har dog fået følge af .Net fra virksomheden Microsoft og denne antages at dele mange af Java's fordele og ulemper.

Appendiks A WebCrawler (ordbogsgenerering)

Dette afsnit handler om det system, der er udviklet som led i dette speciale, til at skabe datagrundlaget for ordbøgerne.

Dette program hører til i den perifere ende og det er derfor begrænset til en teknisk redegørelse – især med fokus på de uventede problemer.

Det var langt det mest komplekse og tidskrævende i denne opgave, da der var nogle overraskende problemstillinger i udviklingen af dette.

Basalt set lever det op til de samme krav til serverprogrammer, som er beskrevet for TextMatcher.

For at opnå en tålelig ydeevne, skulle programmet være flertrådet, da størstedelen af tiden i afviklingen går med at vente på svar fra servere på internettet. Således kørte dette program 20 samtidige jobs under skanningsprocessen.

Programmet baser sig på en mySQL database og alle registrerede data blev lagret i tabeller i denne.

Programmet skulle løse følgende opgaver:

- 1) Læs en side fra internettet (fra .dk domænet)
- 2) Detektér om det er en HTML side.
- 3) Analyser siden og detekter danske ord i den returnerede information. Dette krævede bl.a. oversættelse af såkaldte entitetskoder (bogstaver som åæøéá osv. er angivet via koder).
- 4) Detekter om mængden af ord udgør en sammenhængende tekst eller om der eksempelvis er tale om enkeltstående kommander eller lign.
- 5) Check for ofte benyttede engelske ord – visse sider under .dk domænet er engelsk sprogede.
- 6) Registrer alle ord. Hvis ordet ikke eksisterer i forvejen, bliver det indsat i tabellen ellers opdateres en frekvenstæller for dette ord.
- 7) Detekter referencer (URL'er) til andre sider og lagre dem til gentagelse af skridt 1

Selve tekstanalysen var relativ kompleks, men forudsigelig i udviklingstid. Problemet bestod i at punkt 1 kræver kald af rutiner, som kan blokere uendeligt²⁴. Dette skabte et behov for, at hver job (tråd) også fik en partnertråd, som løbende overvågede om denne var i live, og forsøge at lukke den ned, hvis en blokering blev detekteret. Desværre var dette ikke altid muligt, så programmet blev nødt til at få en uskon tilføjelse: Hvis det faktiske antal af tråde oversteg en vis grænse (beregnet på basis af det logiske antal tråde), blev programmet genstartet. Genstarten blev foretaget ved at programmet lukkede ned med en bestemt

²⁴ Typisk vil operationer, der ikke kan gennemføres, have et tidsudløb (timeout). Det var ikke tilfældet (i Javas implementering af) denne operation.

returværdi, som den kaldende scriptfil (eller batch fil) reagerede på ved at genstarte programmet.

Det samlede samarbejde mellem de to tråde (den som læste og den som overvågede) krævede en del synkronisering, som indebar nye udfordringer.

En anden overraskelse var, at hjemmesider med indeks over andre hjemmesider (som jubii.dk og yahoo.dk) laver en masse krumspring for, at de netop ikke skal kunne skannes på denne måde. Det lykkedes aldrig at omgå disse til trods for ihærdige forsøg og det var derfor nødvendigt manuelt at indsætte de sider, som indeholdte de egentlige referencer til andre hjemmesider.

Konklusion

Konklusionen på de anvendte teorier er meget positiv for større tekstmængder, men kvaliteten daler i takt med at det forsøges at sammenligne mindre dokumenter.

Sammenligning 1 – Store dokumenter

Her blev en kravspecifikation og en designspecifikation (for samme produkt) på henholdsvis 14.584 karakterer og 20.123 karakterer (inklusive blanktegn) sammenlignet.

Resultatet var mere end blot tilfredsstillende. Sættet af klassificerede ord, som teksterne havde tilfælles, var faktisk beskrivende for dokumenternes fælles tema og pointfordelingen var også god.

Enkelte fejl blev dog detekteret. Den mest kritiske var, at produktnavnet (som naturligvis var meget vigtigt) blev reduceret til to forskellige stammer og disse to stammer fik derfor hver deres pointsum.

Ud fra denne sammenligning kan det konkluderes, at TextMatcher og de algoritmer som anvendes i dette system, fuldt ud lever op til forventningerne og kunne være et værdifuldt aktiv i situationer, hvor denne type dokumenter skal kunne sammenlignes.

Sammenligning 1 – Mindre dokumenter

I denne sammenhæng er en række supportsager blevet sammenlignet og her er resultatet svingende. I sager med ren dansk tekst med 350 karakterer eller derover var resultatet brugbart.

Kvaliteten af resultatet dalede dog i takt med at teksterne blev mindre og i et konkret tilfælde med 61 karakterer, var resultatet decideret forkert.

Ud fra disse sammenligninger kan det konkluderes, at TextMatcher og de algoritmer som anvendes i dette system, delvist lever op til forventningerne i sammenligninger af supportsager. Ved tekster på 350 karakterer og derover er resultatet brugbart, men ved helt små tekster på ned til 60 karakter, er resultatet decideret misvisende.

Årsagen til denne sammenhæng skal findes i at de enkelte ord i teksten på godt og ondt får større betydning for sammenligningen i takt med, at teksten bliver mindre. Alene problemstillingen om, hvorvidt ét enkelt ord er med i tillægsordbogen eller ej kan have vital betydning.

Den overordnede konklusion er, at TextMatcher og de algoritmer som anvendes i dette system og som er gjort rede for i dette speciale, som helhed er anvendelig for sammenligning af dansksprogede tekster af en vis størrelse.

De gennemførte tests har givet anledning til at antage, at følgende forbedringsforslag vil påvirke resultatet yderligere i en positiv retning:

- En bedre og mere struktureret tillægsordbog
- En undtagelsesordbog for klassificeringer som desuden angiver den korrekte stamme
- En positivliste over ord, hvis tilstedeværelse i to sammenlignede dokumenter, skal tildeles en høj pointscore
- At benytte en specialiseret sammenligningsalgoritme for små tekster.

Litteratur

INFORMATION RETRIEVAL [1]

C. J. van RIJSBERGEN B.Sc., Dip. NAAC, Ph.D., M.B.C.S., F.I.E.E., C.Eng.,
F.R.S.E.

POLITIKENS NUDANSKE ORDBOG [2]

14. UDGAVE (vedlagt som bilag)

AN INTRODUCTION TO LATENT SEMATIC ANALYSIS [3]

Landauer, Foltz og Laham 1998

USING LATENT SEMANTIC INDEXING FOR INFORMATION FILTERING [4]

Foltz

Bilag

Afsnittet *Grammatisk oversigt fra Politikens nudansk ordbog*