UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

BLAŽ NOVAK

ODKRIVANJE TEMATIK V ZAPOREDJU BESEDIL IN SLEDENJE NJIHOVIM SPREMEMBAM

DIPLOMSKA NALOGA
na univerzitetnem študiju

Mentor:
akad. prof. dr. Ivan Bratko

Somentorica:
doc. dr. Dunja Mladenić

Ljubljana, 2008

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

specifikacija:

Tema te diplomske naloge je problem nenadzorovanega sprotnega sledenja strukturiranim tematikam in odkrivanja novih dogodkov v neomejenih tokovih besedil. V nalogi izdelajte pregled obstoječih pristopov k temu problemu ter predlagajte nov pristop z uporabo metod strojnega učenja. Novi pristop implementirajte in njegovo delovanje prikažite ter ovrednotite na realnih podatkih, kot so npr. novice. Pozornost posvetite tudi učinkovitosti pristopa.

Kazalo

Povzetek

Razvoj informacijske tehnologije v zadnjih desetletjih je s seboj prinesel svojevrsten izziv, saj lahko količina dostopne informacije hitro preseže človeške sposobnosti sledenja. Spletni dnevniki so najnovejši in tudi najbolj obsežen vir takšne informacije.

V tej diplomski nalogi pristopim k problemu sprotnega spremljanja velikih količin besedil z vidika organiziranja novonastale informacije v smiselne skupine.

Prvi del naloge predstavi trenutno stanje na področjih, ki so potencialno pomembna za reševanje zadanega problema, in razloge za neustreznost posameznih metod.

Glavni prispevek diplomske naloge je nov algoritem, ki združuje razne ideje, opisane v prvem delu. Spada v družino algoritmov za hierarhično razvrščanje, s tem da obdeluje podatke sprotno, in ne vseh naenkrat. Algoritem je sposoben postopnega dodajanja in odstranjevanja učnih primerov, model pa se spremeni po vsakem koraku, tako da bolje ustreza strukturi, prisotni v trenutno aktualnem naboru dokumentov.

Na simuliranih učnih množicah, ki sem jih pridobil iz korpusa besedil *Reuters Corpus Volume 1*, sem izvedel nekaj poskusov, da bi preučil lastnosti pristopa. Ugotovil sem, da osnovne predpostavke o časovni zahtevnosti in sposobnosti prilagajanja držijo, in pa da se postopek obnese presenetljivo dobro na raznovrstnih vhodnih podatkih.

Ključne besede: razvrščanje, rudarjenje v podatkovnih tokovih

Abstract

A challenge created by the recent development in information technology is that people are often faced with an overwhelming amount of information available to them, with blogs presenting the latest and most abundant source of such information.

In this thesis, I approach the problem from a standpoint of organizing the newly created information into sensible groups.

The first part of the thesis is an overview of the existing state of the art in the areas relevant to the problem and an analysis of shortcomings of different methods.

The main contribution is the development of a new algorithm that pieces together various ideas presented in the first part. It is an online hierarchical clustering algorithm that is capable of incremental model updates that support not only adding but also the removal of documents. The structure of the model is adapted after each step to better reflect the structure of the currently observed world. The model can also be optimized while waiting for new events.

Some experiments to test the properties of the new algorithm were performed using simulated data streams created from the *Reuters Corpus Volume 1* dataset. I have found that the basic assumptions about time complexity and the ability to adapt the model are correct and that the algorithm performs surprisingly well for a range of different inputs.

Keywords: clustering, stream mining

1

Uvod

1.1 Motivacija

Hiter razvoj komunikacijskih tehnologij in pojav blogov[1] v zadnjih letih sta omogočila običajnim ljudem, da se iz tradicionalnih porabnikov informacij prelevijo v ustvarjalce le-teh in zaobidejo monopol založnikov, ki so v preteklosti obvladovali pretok pisane besede.

Popularnost novega medija, skupaj z nižanjem vstopnega praga za objavljanje svojega bloga, sta povzročila, da se količina tako objavljene informacije povečuje iz dneva v dan: junija leta 2008 je bilo objavljenih med deset in dvajset člankov vsako sekundo — oziroma med enim in dvema milijonoma vsak dan — na približno petnajst milijonih neodvisnih blogov, z upoštevanjem neizvirne vsebine[2] pa se ta številka povzpne na sto do dvesto na sekundo.

Velika količina dostopnih informacij lahko privede do *informacijske preobremenjenosti,*[3] ko posameznik ni več sposoben slediti izbrani tematiki. Situacija je podobna na področju novinarskih člankov[4] in pa v finančnih domenah, kjer je seznanjenost s trenutnim stanjem kritična za sprejemanje kvalitetnih odločitev.

Namen te diplomske naloge je predstavitev algoritma, ki bi lahko s pomočjo avtomatske organizacije olajšal pregled in uporabo velike količine novonastajajočih člankov.

Od rešitve se pričakuje samodejno odkrivanje strukture v množici takšnih člankov. Ker se svet, ki ga besedila opisujejo, nenehno spreminja, se mora temu ustrezno prilagajati tudi izdelana struktura, zaradi velike količine podatkov pa mora biti

[1] Beseda izvira iz angleške besede 'weblog', ki pomeni spletni dnevnik. Ustaljenega slovenskega prevoda še ni, nekateri predlagajo uporabo besede 'spletnik'.

[2] Ljudje vprašljivih moralnih standardov objavljajo tuje članke in ostale prosto dostopne kose informacije z namenom povečevanja števila obiskov na svojih straneh, kjer obiskovalce pričaka obilica oglasov za ročne ure in medicinske rešitve mentalnih težav.

[3] Ang. 'information overload'.

[4] Pogostost takšnih objav na spletu je trenutno med eno in dvema na sekundo.

pristop tudi zelo učinkovit v smislu asimptotske zahtevnosti.

Ob pregledu obstoječe literature nisem našel algoritma, ki bi ustrezal vsem zahtevam, zato tu predlagam metodo, ki po mojem vedenju do sedaj še ni bila uporabljena in ni opisana v znanstveni literaturi, ter predstavljam njene osnovne lastnosti.

1.2 Struktura diplomske naloge

Problem, opisan v zgornjem odstavku leži med več precej nepovezanimi podpodročji: *nenadzorovanim strojnim učenjem*, *rudarjenjem v podatkovnih tokovih* in *odkrivanjem in sledenjem tematikam*. V naslednjem poglavju zato najprej sledi kratek povzetek različnih algoritmov iz literature, ki sem jih pregledal in ki po določenih lastnostih ustrezajo zgornjim zahtevam, ter razlogi, zakaj sem se odločil, da v celoti niso primerni za reševanje izbranega problema.

Nato sledi opis izbrane predstavitve podatkov in načinov njihove predobdelave. Uporabljeni so povsem standardni in uveljavljeni pristopi, zato jim nisem posvečal velike pozornosti, nekaj več poudarka je le na merah podobnosti, saj jih uporabljam na nekoliko neobičajen način. V istem sklopu je tudi opis uporabljenega korpusa besedil, na katerih sem preizkušal smiselnost posameznih pristopov in možnosti.

Predlagana rešitev spada v družino algoritmov za hierarhično razvrščanje,[5] katerih namen je razbitje množice dokumentov na smiselne skupine in podskupine.

Vsak dokument je obravnavan kot nedeljiva celota, postopek pa jih na podlagi izbrane mere podobnosti uvrsti v končno število diskretnih razredov, pri čemer en dokument pripada natanko enemu razredu. Za razliko od običajnega — *ploskega* razvrščanja, kjer so si razredi med seboj enakovredni, obstaja v razbitju, ki ga določi takšen algoritem, hierarhična struktura: razredi imajo lahko podrazrede; interpretacija tega pa je, da dokument, ki spada v nek razred, istočasno spada tudi v vse nadrazrede, pri čemer je podobnost med posameznimi dokumenti, ki so si v takšni hierarhiji bolj oddaljeni, nižja.

Ker se struktura sveta iz trenutka v trenutek spreminja, je model definiran za nek dan interval v času. Ko se pojavi nov dokument ali pa vsebina nekega dokumenta zastara in ni več relevantna za opis sveta, se mora model ustrezno prilagoditi novim podatkom.

Predlagani algoritem vzdržuje drevo, ki predstavlja hierarhično strukturo, tako, da ob vsakem dodajanju ali pa zastaranju dokumenta popravi le majhno okolico poti od dokumenta do korena drevesa.

V poglavju 4 so opisani načini ocenjevanja rezultatov, v jedru diplomske naloge pa podrobno opisujem predlagano metodo v poglavju 5 in eksperimentalne rezultate v poglavju 6. Zaključki in ugotovitve so podane v poglavju 7.

[5] Druge obstoječe slovenske besede za isto stvar so ‘grozdenje’, ‘grupiranje’, ‘particioniranje’; angleški izraz pa je *clustering*.

2

Pregled obstoječih pristopov

Kot sem omenil že v uvodu, so za reševanje zadanega problema in pa nabiranje idej načeloma zanimivi algoritmi treh delno prekrivajočih se skupin, vsi pa spadajo v družino algoritmov nenadzorovanega strojnega učenja.

Prva skupina so algoritmi za razvrščanje, druga skupina algoritmi za obdelavo podatkovnih tokov in tretja skupina algoritmi razviti v okviru delavnic na temo sledenja tematikam v dokumentih.

2.1 Razvrščanje

Za razliko od nadzorovanega strojnega učenja, kjer so podatki, na podlagi katerih naj se računalnik nauči modela sveta, vnaprej označeni z neko lastnostjo, ki je zanimiva za uporabnika — na primer z oznako razreda v primeru uvrščanja v vnaprej določene skupine, ali pa z realnim številom v primeru regresije — in je naloga algoritma, da se nauči preslikave iz podatkov v množico oznak, pri nenadzorovanem strojnem učenju takšne oznake niso na voljo.

Naloga algoritma za nenadzorovano učenje je, da na podlagi podatkov poišče posplošitev, ki jih dobro opisuje in katere predstavitev je običajno bolj preprosta kot pa podatki sami. Ker so za učenje potrebne predpostavke o svetu, je tako kot pri večini algoritmov za nadzorovano (atributno) učenje tudi pri teh osnovna predpostavka, da so si podatki, ki so si med seboj podobni po določenih lastnostih, podobni tudi po ciljni oznaki — le da v tem primeru množica oznak še ni znana.

Način določanja stopnje podobnosti med učnimi primeri je odvisen od izbrane metode. Običajno je podobnost kar eksplicitno definirana s funkcijo, ki preslika dva primera v neko realno število. Pravokotni shemi realnih števil, kjer m_{ij}-ti element opisuje podobnost med i-tim in j-tim učnim primerom, rečemo *matrika podobnosti* (‘*similarity matrix*’). Algoritmi za združevalno razvrščanje pogosto delujejo kar na takšni matriki.

Ena od pogosteje uporabljanih družin algoritmov nenadzorovanega strojnega

učenja so algoritmi za razvrščanje (*clustering*).[1]

Razvrščanje je družina postopkov, ki množico učnih primerov razbijejo na več podmnožic. Posamezni množici takšne particije rečemo grozd, skupina oziroma *cluster*.

Cilj takšnega algoritma je, da je v posameznem grozdu veliko primerov, ki so si med seboj po izbrani meri čimbolj podobni, da je v grozdih čimmanj primerov, ki so si med seboj močno različni, istočasno pa tudi, da so si grozdi med seboj čimbolj različni. *Notranja podobnost* (podobnost med primeri v enem grozdu) in *medgrozdna podobnost* (podobnost med grozdi nekega razbitja) nista enolično določeni, različne izbire imajo lahko za posledico različna optimalna razbitja. Končno razbitje je posplošena izjava o (ne)podobnosti med posameznimi učnimi primeri in s tem opis strukture sveta, iz katerega ti primeri izvirajo.

Algoritme za razvrščanje lahko v grobem razdelimo v 2 skupini: na postopke, ki poiščejo *trdo razbitje*, in postopke, ki poiščejo *mehko razbitje*. Trdo razbitje množice je razbitje, kjer je vsak učni primer element natanko enega grozda, pri mehkem razbitju pa lahko posamezen element pripada več — lahko tudi vsem — grozdom, vendar z različnimi stopnjami pripadnosti.

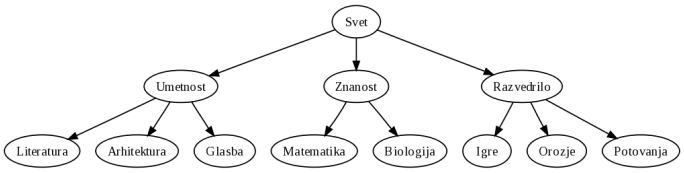
Ena izmed glavnih težav pri razvrščanju v diskretne grozde je izbira števila grozdov, kar običajno označimo s k. Z večanjem števila grozdov se za dano učno množico njihova velikost v povprečju zmanjšuje, posledično pa se njihova notranja podobnost povečuje: če je število grozdov enako številu učnih primerov, so posamezni grozdi seveda povsem čisti; takšen model pa s stališča uporabe ni preveč koristen.

Če mehko razbitje interpretiramo v smislu verjetnosti pripadnosti učnega primera posameznim grozdom, si lahko za določanje števila grozdov pomagamo s principom najkrajšega opisa, kjer večje število grozdov sicer lahko pomeni krajše opise za posamezne primere, vendar dolžina opisa samih grozdov preprečuje prekomerno povečevanje števila le-teh. Za takšen pristop pa je potrebna dobra ocena verjetnosti posameznih primerov glede na izbran model, kar pa je trenutno za tekstovne učne primere še neobvladljiv problem.

Če je rezultat algoritma namenjen za neposredno interpretacijo s strani uporabnika, lahko algoritem večkrat zaženemo z različnim k, izrišemo graf neke mere razbitja v odvisnosti od števila grozdov in prepustimo končno odločitev uporabniku; v takšnem grafu si za število grozdov izberemo točko kjer se krivulja splošči. Več na temo izbire števila grozdov je opisal Manning s sodelavci, [12], v okviru zadane naloge pa tak pristop ni možen, saj se algoritem ne izvaja v prisotnosti uporabnika.

Razbijanje množice glede na znano ocenjevalno funkcijo lahko obravnavamo kot optimizacijski problem. Število možnih različnih razbitij množice velikosti n na k podmnožic je neobvladljivo veliko že za, s stališča obdelave besedil, majhne množice (Stirlingovo število druge vrste, $S(n, k)$), zato je preizkušanje vseh možnosti neizve-

[1]Slovenski izraz 'razvrščanje' nima nič skupnega s *klasifikacijo* (*classification*), ki pomeni uvrščanje primerov v razrede z vnaprej znano odločitveno funkcijo.



Slika 2.1: Primer dendrograma, ki prikazuje nekaj grozdov z vrha taksonomije spletnih strani DMoz [1, 7]

dljivo, preiskovani prostor pa za netrivialne ocenjevalne funkcije nima lepe globalne strukture. Posledica teh dejstev je, da od rešitev ne smemo pričakovati globalne optimalnosti. Mere za ocenjevanje kvalitete razbitja so podrobno opisane v poglavju 4.

Algoritme za trdo razvrščanje lahko nadalje razdelimo v dve skupini: takšne, ki razbijejo množico na skupino enakovrednih grozdov, in pa takšne, pri katerih je razbitje strukturirano. Ker je najbolj pogosto uporabljan algoritem za nehierarhično razvrščanje opisan kasneje v poglavju 5.2.1, so tu opisani le postopki za hierarhično razvrščanje.

2.1.1 Hierarhično razvrščanje

Hierarhično razvrščanje se od običajnega razlikuje po tem, da vsi grozdi razbitja med seboj niso enakovredni. Algoritem za hierarhično razvrščanje zgradi drevo razbitij. Na najvišjem nivoju je množica, ki vsebuje vse primere. Ta množica je razbita na določeno število podmnožic, vsaka izmed teh podmnožic je razbita še na manjše podmnožice, itn. Takšno drevo razbitij lahko predstavimo grafično (primer je na sliki 2.1) in mu rečemo dendrogram (iz grščine, *risba drevesa*), običajno ime za takšno razbitje pa je taksonomija. Relacija urejenosti med grozdi v taksonomiji je ‘podtip–nadtip’, zato se tak način organizacije primerov lepo sklada s človeškim načinom razmišljanja. Primeri hierarhičnih razbitij so znani že dolgo časa, na primer Linnéjeva taksonomija, knjižnične klasifikacijske sheme, ...

Glede na način grajenja drevesa lahko zopet ločimo dva tipa algoritmov za hierarhično razvrščanje.

Razvrščanje z razbijanjem

Algoritmi v tej skupini pričnejo z množico, v kateri so vsi učni primeri. To množico razbijejo na nekaj podmnožic in postopek ponavljajo rekurzivno na nastalih podmnožicah, dokler ni v vsakem končnem razredu le še po en primer, ali pa dosežejo nek drug vnaprej zastavljen cilj, na primer najmanjšo velikost grozda. V vsakem koraku razbijanja se uporabi običajen algoritem za razvrščanje, z vsemi prej opisanimi kvalitetami in slabostmi — predvsem zahteva po znanem k.

Združevalno razvrščanje

Algoritmi za razvrščanje z združevanjem gradijo drevo 'od spodaj navzgor'. Vsak učni primer najprej dodelijo v svoj grozd, nato pa v vsakem koraku izberejo dva najbolj podobna grozda in ju združijo. To ponavljajo, dokler ne ostane le še en grozd, ki zdaj vsebuje vse primere, zaporedje združevanja pa opisuje strukturo.

Obstaja več smiselnih definicij podobnosti med dvema grozdoma, ki so podrobneje opisane v poglavju 3.2.

2.2 Rudarjenje v podatkovnih tokovih

Vsi do sedaj opisani algoritmi so namenjeni le razvrščanju vnaprej znane množice elementov, zato kot taki niso primerni za reševanje zadane naloge, sledeči algoritmi pa so namenjeni prav obdelavi podatkovnih tokov. Podatkovni tok je dolgo — teoretično neskončno — zaporedje učnih primerov, ki prihajajo v sistem zaporedno. Čas prihoda je lahko ekspliciten, v obliki datuma, ali pa impliciten, le kot urejenost glede na predhodnike.

Časovna zahtevnost obdelave posameznega učnega primera v takem primeru ne sme presegati $O(1)$, saj bi se v nasprotnem primeru na vhodu začela nabirati vrsta še ne obdelanih primerov.

Algoritmi za obdelavo podatkovnih tokov delujejo po principu *obdelave v enem prehodu*, kar pomeni, da nimajo dostopa do poljubno starih podatkov. Z vsakim podatkom se lahko ukvarjajo le v trenutku, ko ta podatek prispe v sistem, ali pa največ konstantno število časovnih enot kasneje.

Čeprav so ti algoritmi v osnovi namenjeni obdelavi podatkov, ki prihajajo v sistem kot zaporedje, pa jih lahko uporabimo tudi za obdelavo množice podatkov, ki je prevelika za običajne algoritme za razvrščanje. V takšnih situacijah lahko z več zaporednimi prehodi čez podatke izboljšamo kvaliteto modela (npr. algoritem v [18]).

2.2.1 Spreminjanje tematik

Spreminjanje tematik (*topic drift*) je pojav, povezan s podatkovnimi tokovi. Opisuje situacijo, ko vir, ki generira podatke, ni stacionaren, ampak se spreminja tekom časa.

Model podatkovnega toka, ki je bil zgrajen na preteklih učnih primerih, tako ne ustreza več sedanjim primerom. S tem pojavom je povezanih več izzivov: zaznavanje sprememb, ustrezno popravljanje modela, tako da ustreza trenutnemu stanju, sledenje spremembam z namenom modeliranja dinamike sprememb in pa predstavitev spreminjanja končnim uporabnikom.

Drseče okno

Najpreprostejši način obravnave spreminjajočih podatkov je uporaba drsečega okna. V vsakem trenutku opazujemo le majhno število učnih primerov iz bližnje preteklosti, preostale pa preprosto zavržemo. Takšno okno lahko nato obravnavamo kot običajno množico učnih primerov in nad njo izvajamo algoritme iz poglavja 2.1. Posledica je seveda to, da v sistemu ni več nobenih podatkov o preteklosti izven okna, kar pa lahko delno popravimo tako, da hranimo povzetke podatkov, ki smo jih zavrgli.

Zgodovino lahko vzdržujemo tudi na več nivojih razločljivosti; v [6] predlagajo avtorji uporabo hierarhije modelov. Sistem, ki ga opisujejo, uporablja poljubno mnogo modelov z drsečim oknom, pri čemer ima vsak naslednji model višje v hierarhiji širše okno. Da pa se prednost pristopa ne bi izgubila, je vhod v modele s širokim oknom le še povzetek — nekakšno povprečje — več zaporednih točk. S pravilno izbiro agresivnosti združevanja je prostorska zahtevnost vseh modelov v hierarhiji približno enaka.

Večinoma se uporablja okno konstantne velikosti, v [17] pa je predlagano prilagajanje velikosti okna glede na trenutno velikost ocenjenega spreminjanja tematik.

Staranje

Namesto vzdrževanja okna nespremenljive velikosti lahko starejšim podatkom ob prispetju novih zmanjšamo vpliv na model. Na tak način se lahko izognemo ostrim prehodom med časovno zaporednimi modeli, ki bi lahko nastali, če v podatkih ni zadostne redundance, ki bi zgladila vhodno porazdelitev.

V večini člankov, ki uporabijo tak pristop, je uporabljeno računsko preprosto eksponentno staranje, saj tako ni potrebna hramba izvirnih učnih primerov, uporaba linearnega staranja je zelo redka. Dober primer takšnega pristopa je v [3]; algoritem vzdržuje množico grozdov, ob prihodu novega primera pa zmanjša vpliv vsem dosedanjim primerom. Ker je funkcija staranja eksponentna v odvisnosti od starosti, ob vsakem dogodku pomembnost starih primerov (oziroma njihovega povzetka) preprosto pomnoži z $e^{-\beta\Delta t}$, kjer je β hitrost staranja, Δt pa čas od zadnjega spreminjanja uteži. Če je Δt_i čas med dvema zaporednima popravljanjema uteži in $t = \sum_i \Delta t_i$ starost nekega primera, velja $e^{-\beta t} = e^{-\beta \sum_i \Delta t_i} = \prod_i e^{-\beta \Delta t_i}$, zato se vsi primeri starajo na enak način, ne glede na to, kdaj in kolikokrat se uteži popravljajo.

2.2.2 Algoritmi

BIRCH

BIRCH [18] (*Balanced Iterative Reducing and Clustering using Hierarchies*) je en izmed zgodnjih poskusov algoritmov za inkrementalno razvrščanje. Cilj BIRCH-a

je grajenje taksonomije v enem samem prehodu čez podatkovno bazo, ne obravnava pa odstranjevanja zastarelih podatkov in spreminjanja tematik.

V vsakem vozlišču drevesa, ki predstavlja hierarhijo, hrani algoritem opis povzetka vseh otrok tega vozlišča, t.i. *clustering feature*, ki je trojica $(\mathrm{LS}, \mathrm{SS}, n)$. n je število vseh učnih primerov v poddrevesu, $\mathrm{LS} = \sum_{j=1}^{n} X_j$ njihova vsota in $\mathrm{SS} = \sum_{j=1}^{n} X_j^2$ vsota njihovih kvadratov. Kvadrat vektorja je tu definiran kot vektor, katerega komponente so kvadrati komponent originalnega vektorja.

Algoritem v vseh korakih ohranja uravnoteženo drevo, kjer so listi drevesa trenutni grozdi, notranja vozlišča pa potencialni grozdi, ki bi nastali z združevanjem listov. Ker je namen algoritma učinkovito razvrščanje podatkov v podatkovnih bazah, je število listov parameter, ki je izbran tako, da je velikost vozlišča enaka velikosti strani v podatkovni bazi.

Za vsak prispeli učni primer se s preprostim rekurzivnim iskanjem določi najboljše vozlišče, nakar se vozlišča, ki imajo več otrok, kot jih gre na eno stran, razbije na več manjših. Takšno razbitje poveča število otrok starša tega vozlišča, zato je potrebno potencialno razbijanje ponoviti tudi na staršu in tako navzgor po drevesu vse do vrha.

Po končani prvi fazi predlagajo avtorji še en prehod čez podatke, kjer se drevo lokalno popravi z odstranjevanjem primerov, ki najbolj kvarijo model, in potencialnim združevanjem določenih vozlišč. Alternativna možnost je tudi uporaba običajnega algoritma za razvrščanje nad listi takšnega drevesa.

CluSteam

Algoritem *CluStream* [2] je že v osnovi zasnovan za razvrščanje podatkov v podatkovnih tokovih in za obvladovanje težav, povezanih s količino podatkov in razvojem ter spreminjanjem tematik v toku.

Za razliko od ostalih algoritmov ne izdela uporabnega modela v enem prehodu, ampak razbije delo na dve podnalogi. Postopek, ki obdeluje podatkovni tok, vzdržuje množico statistik, ki povzemajo lastnosti podatkov v toku, uporabnik pa lahko naknadno iz teh statistik zgradi želeno razbitje za izbrano časovno obdobje.

Mikrogrozd (skupek dokumentov, tako kot grozd, vendar ni namenjen končni uporabi) za množico d-dimenzionalnih učnih primerov (za opis predstavitve besedila v obliki vektorjev glej poglavje 3.1) $X_1 \ldots X_n$ s pripadajočimi časi prihoda $t_1 \ldots t_n$ je $(2d+3)$-dimenzionalna n-terica $(\mathrm{CF}_2^{\mathrm{x}}, \mathrm{CF}_1^{\mathrm{x}}, \mathrm{CF}_2^{\mathrm{t}}, \mathrm{CF}_1^{\mathrm{t}}, n)$. Podobno kot pri BIRCHu je tudi to povzetek vseh primerov, ki jih zaobjema mikrogrozd.

$\mathrm{CF}_1^{\mathrm{x}} = \sum_{i=1}^{n} X_j$ je po vzoru BIRCHa vsota primerov, $\mathrm{CF}_2^{\mathrm{x}} = \sum_{i=1}^{n} X_j^2$ vsota njihovih kvadratov, $\mathrm{CF}_1^{\mathrm{t}} = \sum_{i=1}^{n} t_j$ vsota časov prihodov, $\mathrm{CF}_2^{\mathrm{t}} = \sum_{i=1}^{n} t_j^2$ vsota kvadratov časov prihodov in n število primerov, ki jih ta mikrogrozd vsebuje. Mikrogrozdi so med seboj enakovredni.

Centroid grozda lahko iz takšnega opisa preprosto izračunamo kot $\mathrm{CF}_1^{\mathrm{x}}/n$, z uporabo enakosti $D(X) = E[X^2] - E[X]^2$ pa dobimo tudi standardno deviacijo

podatkov v mikrogrozdu. Ker imamo enake podatke za čas, je mikrogrozd definiran s pozicijo in razpršenostjo v času in prostoru.

Ker za vse komponente opisa mikrogrozda velja aditivnost, lahko mikrogrozde združujemo s preprostim seštevanjem opisov.

$$(\mathrm{CF}^{\mathrm{x}}_{1,2}, \mathrm{CF}^{\mathrm{x}}_{1,1}, \mathrm{CF}^{\mathrm{t}}_{1,2}, \mathrm{CF}^{\mathrm{t}}_{1,1}, n_1) + (\mathrm{CF}^{\mathrm{x}}_{2,2}, \mathrm{CF}^{\mathrm{x}}_{2,1}, \mathrm{CF}^{\mathrm{t}}_{2,2}, \mathrm{CF}^{\mathrm{t}}_{2,1}, n_2) = \tag{2.1}$$

$$(\mathrm{CF}^{\mathrm{x}}_{1,2} + \mathrm{CF}^{\mathrm{x}}_{2,2}, \mathrm{CF}^{\mathrm{x}}_{1,1} + \mathrm{CF}^{\mathrm{x}}_{2,1}, \mathrm{CF}^{\mathrm{t}}_{1,2} + \mathrm{CF}^{\mathrm{t}}_{2,2}, \mathrm{CF}^{\mathrm{t}}_{1,1} + \mathrm{CF}^{\mathrm{t}}_{2,1}, n_1 + n_2) \tag{2.2}$$

Sprotni del CluStreama vzdržuje vnaprej določeno število mikrogrozdov. Vsak učni primer, ki pride v sistem, se primerja z vsemi trenutnimi mikrogrozdi. Če se nahaja v bližini katerega izmed njih, se v najbližjega tudi doda, sicer pa se ustvari nov mikrogrozd. Odločitev, ali se primer nahaja v bližini mikrogrozda, je odvisna od tega, ali odstopa od centroida manj kot za nek konstanten faktor od povprečnega odstopanja ostalih članov mikrogrozda.

Ko število mikrogrozdov preseže določeno zgornjo mejo, je potrebno enega od starejših odstraniti iz sistema. Če je možno katerega od obstoječih prepoznati kot nekoristne podatke, na primer če vsebuje le en dokument in ni bil narejen v bližnji preteklosti, ga lahko preprosto zbrišemo, drugače pa je potrebno z uporabo lastnosti 2.1 dva starejša mikrogrozda združiti v enega ali pa najstarejšega popolnoma zavreči.

Na tak način se vzdržuje stalno število mikrogrozdov, ki približno opisujejo podatkovni tok; količina je odvisna od strojnih virov.

Tako nastali mikrogrozdi se periodično shranjujejo na sekundarne pomnilne medije. Glavna ideja je, da je število takšnih posnetkov stanja v času veliko za bližnjo in manjše za daljno preteklost, z več vmesnimi stopnjami gostote.

Ko si uporabnik zaželi pregled nad stanjem v določenem časovnem intervalu, sistem z linearno kombinacijo shranjenih posnetkov rekonstruira približno stanje v tistem trenutku in združi mikrogrozde na podlagi uporabnikovih zahtev v končno sliko.

Delni k–means z zlivanjem in LOCALSEARCH

Delni k-means z zlivanjem [13] definira problem razvrščanja kot zaporedje operacij nad tokom podatkov, čeprav v osnovi ni zasnovan za obdelavo tokov, ampak velikih statičnih množic satelitskih meritev. Izpeljan je iz znanega algoritma k-means [11], za detajlen opis glej poglavje 5.2.1).

Sestavljen je iz dveh operatorjev nad tokom podatkov: delnim razvrščanjem in zlivanjem. Vhodne podatke najprej razdeli v množice tolikšne velikosti, da jih še lahko naloži v delovni spomin, nakar posamično razbije vsako množico v grozde z uporabo običajnega k-means algoritma. Rezultat razbitja pa je poleg grozdov tudi množica centroidov – po en za vsak grozd – ki jim je prirejena utež glede na število podatkovnih točk, ki pripadajo ustreznemu grozdu.

Druga stopnja algoritma sprejme kot vhod množico uteženih centroidov iz prve stopnje. Te centroide potem z uporabo k-meansa razvrsti v množico grozdov, kot da

bili običajne podatkovne točke, le da pri izračunu centroidov *teh* grozdov nimajo vsi prvostopenjski centroidi enake pomembnosti, ampak se upoštevajo glede na njihovo težo — število podatkovnih točk, ki jih zastopajo.

Takšnega postopka ni mogoče uporabiti na podatkovnem toku, ker mora prva stopnja obdelati celoten vhod, preden lahko začne delo druga stopnja, zato predlagata avtorja kompromisno rešitev. Vedno, ko prva stopnja obdela množico podatkovnih točk in preda opis grozdov in pripadajočih centroidov, jih druga stopnja doda v množico svojih delovnih centroidov in z algoritmom k-means nekatere od njih združi, tako da je končno število centroidov v drugi stopnji vedno enako.

Podoben pristop predlagata tudi avtorja algoritma LOCALSEARCH [8].

2.3 Odkrivanje in spremljanje tematik

Področje, ki se ukvarja z zasledovanjem tematik v besedilih se v angleščini imenuje *topic detection and tracking* in izvira iz serije delavnic TDT1999 – TDT2004 [4]. V okviru teh delavnic so definirali razna podpodročja, kot npr. odkrivanje tematik, sledenje znani tematiki, zaznavanje prve novice v zaporedju sorodnih, itn. Pričakoval sem, da bo s tega področja največ literature, na katero bi se lahko navezal, vendar se je izkazalo ravno nasprotno.

Cilji, definirani v okviru teh delavnic so zelo specifični, saj se meri točnost na majhnem številu dobro definiranih zgodb, velikost učnih in testnih množic pa je tako majhna, da večina pristopov, ki sem jih preučil, temelji na nekakšni modifikaciji metode najbližjih sosedov: vsak nov dokument se primerja z vsemi v modelu, razlike med posameznimi pristopi pa so v načinih uteževanja atributov, načinih zanemarjanja starejših dokumentov — uporabe drsečega okna — in odločanja o pragu podobnosti, ki ga mora nov primer preseči (glede na množico starejših), da se še smatra kot da pripada v neko starejšo zgodbo.

Ker metoda najbližjih sosedov ni primerna za reševanje zadane naloge, podrobnosti algoritmov s tega področja ne bom opisoval.

3

Podatki

3.1 Predstavitev podatkov

Večina omenjenih algoritmov, kot tudi ta v jedru diplome, ni odvisnih od načina predstavitve podatkov, dokler obstaja mera podobnosti, ki dvema primeroma dodeli realno vrednost z nekega intervala, ki je majhna za primere, ki so si po nekem izbranem principu različni, in visoka za takšne, ki so si podobni.

Na področjih strojnega učenja in iskanja informacij (*information retrieval*) se je uveljavil pristop, ki temelji na vektorskih prostorih, tim. *vector space model*. V [15] ga je Salton opisal za namene iskanja dokumentov.

Vsak dokument d je predstavljen kot t–dimenzionalni vektor $\mathbf{d} \in \mathbb{R}^t$. Komponente i–tega dokumenta $\mathbf{d}_i = (d_{i1}, d_{i2}, \ldots, d_{it})$ predstavljajo prisotnost posameznih različnih besed, ki se lahko pojavijo v korpusu, v tem dokumentu. Takšnemu načinu predstavitve pravimo tudi *vreča besed* (*bag of words*), ker vrstni red besed v dokumentu zanemarimo in opazujemo le njihove pojavitve.

Najpreprostejša oblika takšne predstavitve so vektorji *TF*, kjer j-ta komponenta vektorja $\mathbf{d}_i$ predstavlja število ponovitev besede t_j v i-tem dokumentu:

$$\mathrm{TF}(d)_{i,j} = n_{i,j} \tag{3.1}$$

Z ne preveliko izgubo točnosti lahko to predstavitev še poenostavimo tako, da je zaloga vrednosti komponent le $\{0, 1\}$, tako da 0 pomeni, da se beseda v dokumentu ne pojavi nikoli, in 1, da se pojavi vsaj enkrat.

$$\mathrm{TF}_{\mathrm{bin}}(d)_{i,j} = \begin{cases} 0 & \text{za } n_{i,j} = 0 \\ 1 & \text{za } n_{i,j} > 0 \end{cases} \tag{3.2}$$

Salton je v [15] ugotovil tudi, da je koristnost posameznih besed s stališča njihove uporabnosti za razločevanje tematik odvisna od tega, v kolikšnem deležu dokumentov se posamezna beseda ponovi: najboljše so tiste, ki niso ne prepogoste in tudi

ne preveč eksotične. Da se zmanjša vpliv besed, ki se pojavljajo v velikem številu dokumentov (kot so na primer vezniki, medmeti, ...) na mere podobnosti, lahko posamezne komponente vektorja *TF* pomnožimo z obratno vrednostjo števila dokumentov (označeno z IDF_j, *inverse document frequency*), v katerih se ta beseda pojavi vsaj enkrat.

$$\mathrm{IDF}_j = \frac{\text{število vseh dokumentov}}{\text{število dokumentov, ki vsebujejo besedo } j} \tag{3.3}$$

V uporabi so tudi drugačne definicije IDF uteževanja, na primer '$\log \mathrm{IDF}_j$', kjer vzamemo logaritem razmerja dokumentov iz enačbe 3.3. Poskus 6.5.1 kaže vplive različnih načinov uteževanja na kvaliteto razbitja.

TFIDF vrednost j–te komponente vektorja d_i je

$$\mathrm{TFIDF}_{i,j} = TF_{i,j} \cdot IDF_j \tag{3.4}$$

Tako dobljen opis TFIDF običajno še normaliziramo na enotsko dolžino glede na normo L_2 ($\sqrt{\sum_{j=1}^{n} d_{ij}} = 1$), s čimer odstranimo vpliv različnega števila besed v dokumentih.

3.2 Mere podobnosti

Tako opisane normalizirane dokumente si lahko predstavljamo kot da ležijo na hiperkrogli z radijem 1. Podobnost med dvema dokumentoma lahko zdaj definiramo kot kosinus prostorskega kota $\phi_{i,j}$ med dvema vektorjema i in j, ki ta dokumenta predstavljata:

$$\cos\phi_{i,j} = \frac{\mathbf{d_i} \cdot \mathbf{d_j}}{||\mathbf{d_i}||\,||\mathbf{d_j}||} \tag{3.5}$$

Če so dokumenti normalizirani, je to kar $\mathbf{d_i} \cdot \mathbf{d_j}$. Takšno mero podobnosti bom označeval s $\cos(d_i, d_j)$.

Povzetek $\mathbf{p}$ grozda G definirajmo kot

$$\mathbf{p} = \sum_{d \in G} \mathbf{d}, \tag{3.6}$$

centroid $\mathbf{c}$ pa kot

$$\mathbf{c} = \frac{1}{|G|} \sum_{\mathbf{d} \in G} \mathbf{d}, \tag{3.7}$$

kar je povprečje vseh vektorjev v grozdu.

V predlaganem algoritmu so lahko elementi grozda poleg dokumentov tudi drugi grozdi; v tem primeru je povzetek grozda neutežena vsota po vseh dokumentih, ki se nahajajo v poddrevesu, centroid pa povprečje tako definiranega povzetka.

Podobnost med grozdoma G_1 in G_2 s pripadajočima centroidoma c_1 in c_2 in pa med grozdom G in posameznim dokumentom $\mathbf{d}$ lahko definiramo na več načinov. Izračun podobnosti med grozdom in dokumentom lahko obravnavamo kot primer, ko imamo namesto dokumenta grozd, v katerem je le en dokument.

3.2.1 Ekstremne podobnosti

Za podobnost med grozdoma si lahko izberemo kar podobnost med dvema najbolj podobnima (*single link clustering*) ali pa najbolj različnima (*complete link clustering*) dokumentoma d_i in d_j, $d_i \in G_1$, $d_j \in G_2$. Kljub uporabi v ostalih vedah se za obdelavo besedil ti meri ne uporabljata.

3.2.2 UPGMA — metoda aritmetičnega povprečja neuteženih parov

UPGMA (*Unweighted Pair Group Method with Arithmetic Mean*) je povprečna podobnost med vsemi dokumenti iz obeh grozdov, vključno z medsebojnimi podobnostmi dokumentov iz posameznih grozdov, le podobnost dokumentov samih s seboj ni upoštevana.

$$\mathrm{UPGMA}(G_1, G_2) = \frac{1}{(|G_1| \cdot |G_2|)(|G_1| \cdot |G_2| - 1)} \sum_{d_i \in G_1 \cup G_2} \sum_{d_j \in G_1 \cup G_2, d_i \neq d_j} d_i \cdot d_j \quad (3.8)$$

Ta mera se veliko uporablja pri razvrščanju z združevanjem, tako na besedilih kot tudi v računski biologiji. Izračun podobnosti grozdov predstavlja daleč najbolj procesorsko časovno potratno operacijo (glej 6.7). Zahtevnost naivne implementacije na podlagi enačbe 3.8 je za primerjavo dveh grozdov $O((|G_1| + |G_2|)^2)$ z zelo velikim konstantnim faktorjem, ki vsebuje množenje dveh vektorjev. Na srečo je vsota skalarnih produktov enaka skalarnemu produktu vsot:

$$\sum_{\mathbf{d_i} \in G_1} \sum_{\mathbf{d_j} \in G_2} \mathbf{d_i} \cdot \mathbf{d_j} = \left(\sum_{\mathbf{d_i} \in G_1} \right) \cdot \left(\sum_{\mathbf{d_j} \in G_2} \right), \quad (3.9)$$

tako da lahko enačbo 3.8 prepišemo v

$$\mathrm{UPGMA}(G_1, G_2) = \quad (3.10)$$

$$= \frac{1}{(|G_1| \cdot |G_2|)(|G_1| \cdot |G_2| - 1)} \left[\left(\sum_{d \in G_1 \cup G_2} \mathbf{d} \right)^2 - (|G_1| + |G_2|) \right] \quad (3.11)$$

$$= \frac{1}{(|G_1| \cdot |G_2|)(|G_1| \cdot |G_2| - 1)} \left[\left(\sum_{d \in G_1} \mathbf{d} + \sum_{d \in G_2} \mathbf{d} \right)^2 - (|G_1| + |G_2|) \right], \quad (3.12)$$

odštevanje ($|G_1| + |G_2|$) pa je popravek za izključitev samopodobnosti, ki so 1.

Ker je cena za vzdrževanje pravilnega povzetka grozda tekom algoritmov za združevanje zanemarljiva, je tako zahtevnost izračuna podobnosti le en izračun norme L_2 vsote dveh vektorjev.

V eksperimentih (glej 6.4) se je kljub precej boljšim rezultatom v [16] ta mera obnesla slabše kot podobnost med centroidi, ki je opisana v naslednjem razdelku. Razlika ni tako presenetljiva, saj avtorji v [16] uporabljajo mero le za razvrščanje z združevanjem, algoritem ki je predstavljen tu, pa za razvrščanje z razbijanjem.

3.2.3 Razdalja med centroidi

Podobnost med centroidoma grozdov

$$\begin{aligned}
\mathrm{CENT}(G_1, G_2) &= c_1 \cdot c_2 && (3.13)\\
&= \left(\frac{1}{|G_1|}\sum_{d_i \in G_1} d_i\right) \cdot \left(\frac{1}{|G_2|}\sum_{d_j \in G_2} d_j\right) && (3.14)\\
&= \frac{1}{|G_1| \cdot |G_2|}\left(\sum_{d_i \in G_1} d_i\right) \cdot \left(\sum_{d_j \in G_2} d_j\right) && (3.15)\\
&= \frac{1}{|G_1| \cdot |G_2|}\sum_{d_i \in G_1}\sum_{d_j \in G_2} \mathbf{d_i} \cdot \mathbf{d_j}, && (3.16)
\end{aligned}$$

ki je skalarni produkt nenormaliziranih centroidov je, kot UPGMA, izračunljiva brez primerjanja vseh dokumentov, ki pripadajo grozdoma. Kot je razvidno iz enačbe 3.16, je podobnost med centroidoma enaka povprečni podobnosti med dokumenti iz posameznih grozdov, vendar brez upoštevanja podobnosti med dokumenti, ki pripadajo istemu grozdu.

Razdalja $\mathrm{CENT}(c_i, c_j)$ ni nujno enaka $\cos(c_i, c_j)$, saj centroidi grozdov v splošnem niso enotske dolžine.

3.3 Jezikovna predobdelava podatkov

Kljub uporabi IDF uteževanja se je izkazalo za koristno, da se določene neinformativne besede odstrani iz podatkov še pred pretvorbo v TFIDF obliko. Za vse eksperimente sem uporabil spisek 571 angleških besed, ki se običajno uporabljajo v ta namen; izvirajo pa iz sistema SMART[5].

Ker je namen naloge ločevanje tematik, ne pa podrobno razumevanje besedila, lahko različne pojavne oblike iste besede spremenimo v osnovno obliko in s tem povečamo podobnost pri dokumentih, ki uporabljajo podobne, a ne povsem enake besede. Za ta korak obdelave vhodnih podatkov sem uporabil Porterjev krnilnik [14], ki je del modula NLTK za programski jezik Python.

3.3.1 Primer podatkov

Za demonstracijo pretvorbe dokumenta iz začetne oblike do končne TF predstavitve tu ves postopek pokažem na delu ene izmed novic iz korpusa RCV1 opisanega v naslednjem razdelku (novica št. 3541).

Izvirna novica

A group of 56 Chinese writers, former officials and academics has petitioned Communist Party chief Jiang Zemin to save cultural relics from the gigantic Three Gorges dam which would flood huge tracts of land. Ancient tombs and temples were threatened by a dearth of government funds and official under-reporting of those relics that warranted saving, according to the letter, a copy of which was made available to Reuters on Tuesday. "The hearts of relic protection departments are like burning fire with 10,000 worries and misgivings"it said of the lack of funds. About 130 historical sites, some of them dating to the Stone Age, could be flooded as soon as next year. The sites include Qing dynasty (1644-1911) temples, an entire street from the Ming Dynasty (1368-1644) and stone carvings from the Han Dynasty (206 BC-AD 220). Relocation of many relics had been delayed due to lack of funds, the letter said, adding that the damage would worsen unless government funds for relocation were provided.

Novica po odstranitvi pogostih besed

group 56 chinese writers , officials academics petitioned communist party chief jiang zemin save cultural relics gigantic gorges dam flood huge tracts land . ancient tombs temples threatened dearth government funds official - reporting relics warranted saving , letter , copy made reuters tuesday . "hearts relic protection departments burning fire 10 , 000 worries misgivings "lack funds . 130 historical sites , dating stone age , flooded year . sites include qing dynasty (1644 - 1911) temples , entire street ming dynasty (1368 - 1644) stone carvings han dynasty (206 bc - ad 220). relocation relics delayed due lack funds , letter , adding damage worsen government funds relocation provided .

Novica po Porterjevem krnjenju

group chines writer offici academ petit communist parti chief jiang zemin save cultur relic gigant gorg dam flood huge tract land ancient tomb templ threaten dearth govern fund offici report relic warrant save letter copi made reuter tuesday heart relic protect depart burn fire worri misgiv lack fund histor site date stone age flood year site includ qing dynasti templ entir street ming dynasti stone carv han dynasti bc ad reloc relic delay due lack fund letter ad damag worsen govern fund reloc provid

Novica kot TF vektor

'depart':1, 'offici':2, 'lack':2, 'damag':1, 'ancient':1, 'group':1, 'han':1, 'writer':1, 'due':1, 'includ':1, 'threaten':1, 'communist':1, 'save':2, 'warrant':1, 'tuesday':1, 'relic':4, 'fire':1, 'dam':1, 'copi':1, 'zemin':1, 'protect':1, 'fund':4, 'qing':1, 'worri':1, 'gorg':1, 'report':1, 'made':1, 'provid':1, 'histor':1, 'heart':1, 'ad':2, 'petit':1, 'misgiv':1, 'cultur':1, 'site':2, 'dearth':1, 'ming':1, 'street':1, 'year':1, 'gigant':1, 'huge':1, 'worsen':1, 'parti':1, 'reuter':1, 'delay':1, 'flood':2, 'dynasti':3, 'bc':1, 'chines':1, 'burn':1, 'carv':1, 'templ':2, 'entir':1, 'letter':2, 'tract':1, 'date':1, 'stone':2, 'land':1, 'jiang':1, 'age':1, 'chief':1, 'govern':2, 'reloc':2, 'tomb':1, 'academ':1

3.4 Predstavitev podatkov

Vektorji TF ali TFIDF imajo vrednost večine komponent 0, saj je v posameznem dokumentu prisotnih le nekaj besed iz nabora vseh možnih. Hranjenje vseh ničelnih vrednosti bi bilo izredno potratno, zato je v uporabi predstavitev z *redkimi vektorji*. Redki vektor je v spominu predstavljen kot vektor parov (i, v), kjer predstavlja i indeks neke neničelne komponente vektorja, ki ga opisujemo, v pa njegovo vrednost.

3.5 Eksperimentalni material

Vse poskuse navedene v poglavju 6 sem izvedel nad korpusom besedil RCV1 (*Reuters Corpus Volume 1*, [10]). Zbirka besedil obsega približno 810.000 novic o dogodkih med 20. avgustom 1996 in 19. avgustom 1997, kar je okrog 2.300 novic na dan. Velika večina teh novic je ročno razporejena v plitvo taksonomijo 103 kategorij. Dve največji skupini kategorij pokrivata novice o ekonomiji in podjetjih, sledi še skupina kategorij ki pokriva tržne novice, in pa večje število manjših kategorij raznovrstnih novic, od mode do športa.

V tabeli 3.1 je statistika števila dokumentov za kategorije, ki jih uporabljam v eksperimentih. Nekateri dokumenti lahko pripadajo tudi več kategorijam istočasno, zato so tudi tu šteti večkrat.

Kategorija	velikost	opis
CCAT	381327	Industrija
GCAT	239267	Splošno
ECAT	119920	Ekonomija
M14	85440	Trgovanje z dobrinami
GDIP	37739	Diplomacija
GSPO	35317	Šport
GVIO	32615	Nasilje
GCRIM	32219	Kriminal
M143	21957	Trgovanje z energijo
G15	20672	Evropska Unija
GDIS	8657	Naravne nesreče
GENV	6261	Okolje
GHEA	6030	Zdravje
GENT	3801	Zabava
GREL	2849	Religija
GSCI	2410	Znanost
GFAS	313	Moda

Tabela 3.1: Število dokumentov po posameznih kategorijah korpusa RCV1

4

Ocenjevanje razbitja

Ocenjevanje kvalitete razbitja je odvisno od uporabnika, zato obstaja mnogo različnih mer. Najbolj očitno jih lahko razdelimo v dve skupini glede na to, ali uporabljajo podatke, ki algoritmu za razvrščanje med izvajanjem niso na voljo.

Meram, ki na nek način formalizirajo zahtevo, naj si bodo dokumenti v grozdih čimbolj podobni, dokumenti v različnih grozdih pa čimbolj različni in ne uporabljajo dodatnih podatkov rečemo *notranje mere* (*internal quality measure*). Mere, ki primerjajo kvaliteto razbitja z nekim vnaprej določenim razbitjem (t.i. *golden standard*) so *zunanje mere* (*external quality measure*). Z njimi lahko primerjamo razbitje z neko referenco, ki jo lahko za določeno množico dokumentov sestavi človek.

4.1 Notranja podobnost

Najpreprostejša mera notranje podobnosti je kar utežena povprečna kompaktnost grozdov, če je kompaktnost definirana kot povprečna podobnost med dokumenti v grozdu G s centroidom c in povzetkom p:

$$\begin{aligned} \text{KOMP}(G) &= \frac{1}{|G|^2} \sum_{d_i \in G, d_j \in G} \cos(d_i, d_j) && (4.1) \\ &= \frac{1}{|G|} \left(\sum_{d \in G} \mathbf{d} \right) \cdot \frac{1}{|G|} \left(\sum_{d \in G} \mathbf{d} \right) && (4.2) \\ &= \mathbf{c} \cdot \mathbf{c}, && (4.3) \end{aligned}$$

torej skalarni produkt centroida s seboj, oziroma kvadrat L_2 norme centroida, $\sum_{c_i} c_i^2$.

Navadno (makro)povprečje tako definiranih kompaktnosti grozdov se je v eksperimentih izkazalo za težavno, saj imajo majhni grozdi enak vpliv na skupno oceno kot veliki, predstavljajo pa le majhen del populacije. Če povprečje utežimo z velikostjo grozdov, dobimo:

$$\text{INTRA}(G_1, G_2, \ldots, G_n) = \frac{1}{|G|} \sum_{G_i} |G_i| \cdot \text{KOMP}(G_i) \quad (4.4)$$

$$= \frac{1}{|G|} \sum_{G_i} |G_i| \cdot c_i \cdot c_i \quad (4.5)$$

V primeru hierarhičnega razbitja se centroid izračuna kot je opisano v 3.2.

4.2 Zunanje mere

V uporabi je množica mer za ocenjevanje običajnih razbitij, splošne mere za hierarhična razbitja pa so precej bolj redke.

4.2.1 Mera F_1

Za ocenjevanje kvalitete hierarhičnega razbitja, predlagata Larsen in Aone v [9] uporabo razširjene mere F_1. V kontekstu nadzorovanega strojnega učenja ocenjuje mera F_1 kompromis med sposobnostjo algoritma, da kot pozitivne označi le primere, ki so v resnici pozitivni (natančnost), in pa sposobnostjo, da kot pozitivne označi čimveč primerov, ki so v resnici pozitivni (priklic).

Razširjena F_1 mera predpostavlja, da je za vsako od znanih tematik $t \in T$, ki se pojavljajo v razbitju, neko vozlišče najbolj reprezentativno; da celotno poddrevo tega vozlišča predstavlja to tematiko. Definirana je kot

$$F_1 = \frac{\sum_{t \in T} (|T| \cdot F_1(t))}{\sum_T |T|}, \quad (4.6)$$

torej uteženo povprečje F_1 ocen po posameznih tematikah. F_1 ocena za tematiko t je

$$F_1(t) = \frac{2 \cdot \text{točnost}(t) \cdot \text{priklic}(t)}{\text{točnost}(t) + \text{priklic}(t)}, \quad (4.7)$$

kjer sta točnost in priklic, merjeni v nekem grozdu g

$$\text{točnost}(t) = \frac{\text{število dokumentov, prirejenih tematiki } t}{\text{število vseh dokumentov v grozdu}} \quad (4.8)$$

$$\text{priklic}(t) = \frac{\text{število dokumentov, prirejenih tematiki } t}{\text{število vseh dokumentov prirejenih tematiki } t \text{ v celotnem razbitju}} \quad (4.9)$$

Pri štetju števila dokumentov v nekem grozdu se upoštevajo tudi dokumenti iz podgrozdov. Za vrednost $F_1(t)$ v enačbi 4.6 se uporabi najvišja vrednost za tematiko t po vseh grozdih v drevesu.

4.2.2 Povprečna dolžina opisa podatkov

Pri uporabi zgornje mere sem opazil težavo, ki izvira iz njenih predpostavk. Ker se v končnem izračunu uporablja le po ena vrednost na tematiko, izvirajo te vrednosti običajno iz samega vrha drevesa — iz otrok korena — zato se posplošena F_1 mera ne spremeni, če se popolnoma spremeni podstruktura vozlišča, ki je bilo izbrano kot vozlišče za neko tematiko. Poleg tega se zaradi omejitev na razvejanost drevesa lahko zgodi, da pride ena tematika v več vej drevesa že na najvišjem nivoju, pri ocenjevanju pa se uporabi le ena izmed teh vej.

Za lažje spremljanje razvoja eksperimentov sem si zato definiral še svojo mero:

$$\mathrm{DL} = -\frac{\sum_{d \in D} \log_{|T|} p_{t(d)}}{|D|}, \tag{4.10}$$

kjer je D množica vseh dokumentov v drevesu, T množica resničnih tematik v korpusu (tako je DL vedno z intervala $[0, 1]$), $p_{t(d)}$ pa verjetnost razreda t, ki mu pripada dokument d v grozdu G, kateremu je d prirejen: $p_{t(d)} = \frac{n_t(G)}{n(G)}$. V tej enačbi je $n_t(G)$ število dokumentov v grozdu, ki pripadajo resnični tematiki t — pri izračunu se štejejo tudi dokumenti, ki pripadajo podgrozdom grozda G; enako za $n(G)$.

Avtorji v [16] uporabljajo za ocenjevanje razbitja predvsem entropijo, vendar le na končnih grozdih, ker je namen članka primerjava nehierarhičnih razvrščanj.

Poskusil sem, kako se tak pristop obnaša, če izmerim entropijo kar na vseh grozdih in jo seštejem, vendar je mera izredno nestabilna, saj majhne spremembe globine drevesa prinašajo velike spremembe entropije, ker se posamezni primeri upoštevajo večkrat, četudi se drevo samo ne spremeni veliko. Normalizacija ni smiselna, saj se nekateri dokumenti štejejo večkrat.

Povprečna dolžina opisa podatkov temelji na podobnem razmisleku kot entropija; v primeru da razbitje nima hierarhije, sta meri kar enaki.

Tudi ta mera ima težave (nekatera drevesa, ki imajo boljšo strukturo dobijo slabšo oceno); vendar se je v poskusih dobro ujemala z vizualno oceno kvalitete drevesa: bolj čista drevesa so imela v večini primerov nižjo povprečno dolžino opisa. Boljša mera bi morala upoštevati tudi ceno razbitij. Popravek mere, da bo bolj ustrezala predstavi o kvalitetnih drevesih je priložnost za nadaljnje raziskave.

V nekaterih eksperimentih imenujem to mero *nečistoča drevesa*.

5

Predlagana metoda

The only way to make sense
out of change is to plunge into it,
move with it, and join the dance.
∼ Alan Watts

Algoritmi opisani v poglavju 2 imajo vrsto pomanjkljivosti. Nekateri niso primerni za obdelavo podatkovnih tokov, drugi ne omogočajo prilagajanja spremembam izvora, ali pa predpostavljajo lastnosti izvora podatkov, ki jim pisana besedila ne ustrezajo.

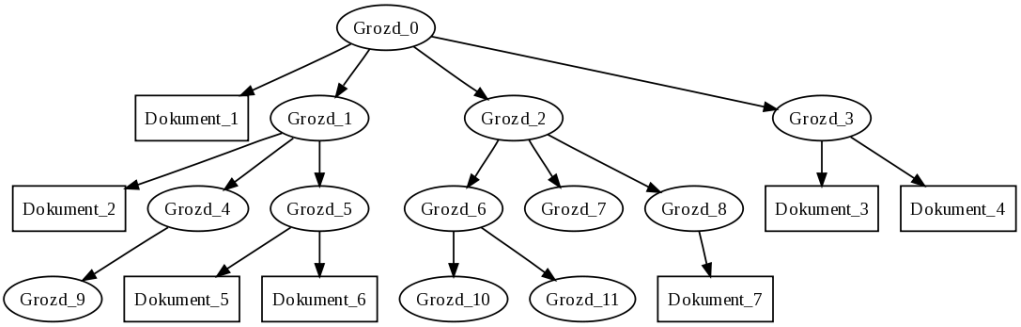
V tem poglavju predstavim svoj predlog algoritma, ki deluje na tokovih tekstovnih podatkov in uspešno prilagaja model spremembam izvorne distribucije tekom časa. Asimptotska zahtevnost operacij dodajanja in odstranjevanja učnih primerov iz modela je linearna v odvisnosti od globine drevesa, oziroma približno $O(\log n)$, kjer je n število učnih primerov trenutno v drevesu, če podatki niso izrojeni.

5.1 Opis metode

Algoritem je načeloma izpeljan iz BIRCHa (2.2.2), nekatere ideje pa sem pobral iz drugih opisanih algoritmov.

Deluje na *neskončnem* zaporedju podatkovnih točk, ki so vedno elementi vektorskega prostora $\mathbb{R}^n$, n-terice realnih števil. Z vsakim podatkom je povezan čas prihoda v sistem. Ta je lahko impliciten, v tem primeru je časovna oznaka kar zaporedna številka podatka, ali pa ekspliciten v obliki sekund od določene časovne prelomnice. Podatke, ki trenutno sestavljajo model, določi drseče okno, katerega velikost je lahko določena z absolutno omejitvijo na število primerov v modelu, ali pa z najvišjo dovoljeno starostjo podatka.

Podatkovni model Ena od slabosti algoritma CluStream (2.2.2) je časovna zahtevnost za dodajanje oziroma odstranjevanje primera iz modela, ki je linearna v



Slika 5.1: Primer podatkovnega modela

odvisnosti od števila *vseh* grozdov v modelu.

Model, ki ga vzdržuje predlagani algoritem, je opisan z drevesom. Vozlišča v drevesu predstavljajo grozde — skupine podobnih dokumentov — in pa dokumente. Dokumenti, ki so podlaga za ustvarjanje modela, so prirejeni enemu od vozlišč, ki predstavljajo grozde. Takšno vozlišče ima lahko poleg dokumentov za otroke tudi druge grozde. Poleg otrok vsebuje grozdno vozlišče tudi *povzetek*, ki je vsota vseh podatkovnih točk, ki jih ta grozd pokriva, tako neposrednih kot tudi tistih, ki so dodeljene kateremu od njegovih (potencialno neposrednih) potomcev.

Primer podatkovnega modela je na sliki 5.1. V vseh ilustracijah so dokumenti predstavljeni s pravokotnikom, grozdi pa z elipso.

Dogodki v modelu Dva osnovna dogodka v modelu sta prihod in zastaranje primera, ki se razlikujeta le po prvem koraku, vsi nadaljnji pa so povsem enaki.

Ob *prihodu* nove podatkovne točke se najprej poišče vozlišče v drevesu, ki je najbolj podobno učnemu primeru. Po eksperimentiranju z merami (6.4), opisanimi v 3.2, se je za najbolj primerno mero izkazala kosinusna razdalja do centroida grozda (3.2.3). Za iskanje uporabim snopovno iskanje z ozkim snopom (širina snopa od 1 do 5), da se izognem nekaterim lokalnim minimumom. V poglavju 6.5.2 so ilustrirane razlike glede na uporabljeno širino snopa. Ko je določeno najboljše vozlišče, se mu priredi točka in rekurzivno popravi povzetek vsem vozliščem na poti do korena drevesa.

Popravljanje povzetkov ob dodajanju pomeni preprosto prištevanje podatkovnega vektorja k vektorju povzetkov v ustreznem vozlišču.

Ob dodajanju v model se točke istočasno dodajo tudi v vrsto FIFO, kjer čakajo na zastaranje. Ko je izpolnjen pogoj za zastaranje, se točka odstrani iz pripadajočega vozlišča v drevesu; prav tako kot pri dodajanju se popravijo vsi povzetki do korena drevesa.

V naslednjem koraku se delno popravi struktura drevesa, da bolje ustreza novim

podatkom.

Popravki strukture Algoritem *CluStream* deli grozde, ki so po neki meri preveliki, na več manjših. Odločitev o deljenju je odvisna od čistosti grozda, ker pa so vsi grozdi enakovredni, zadostuje primerjanje z eno vnaprej določeno konstanto.

Ker predlagani algoritem vzdržuje drevo, bi moralo biti takšnih konstant več — po ena za vsak nivo — in če naj bo razvejanost smiselno omejena, tudi različne za posamezne veje v drevesu (razen za primere zelo lepo porazdeljenih podatkov).

Vsi poskusi z uporabo ene same meje kompaktnosti (enačba 4.1, tu uporabljena za ocenjevanje primernosti razbitja) za določeno globino v drevesu, omejitvijo na relativno kompaktnost otrok glede na starša, ali pa omejitvijo na relativno kompaktnost glede na ostala vozlišča na isti globini so hitro pripeljali do izrojenih dreves, ki niso bila več primerna s stališča asimptotske zahtevnosti operacij nad njimi, je pa en izmed takšnih poskusov uporabljen za primerjavo v poskusu 6.7.

Tekom časa se lahko več tematik zlije v eno, zato mora takšnemu obnašanju slediti tudi model. CluStream to rešuje z zlivanjem mikrogrozdov, odločitev o zlivanju pa temelji na navzgor omejenem številu mikrogrozdov v sistemu: zlijeta se najbolj podobna, ko je ob nastanku novega grozda presežena dovoljena meja.

V drevesnem modelu je možen še en tip spremembe: vertikalna migracija grozdov po nivojih drevesa, ko neka tema ni več podtema neke druge, ampak je njej enakovredna, ali pa ji postane nadrejena.

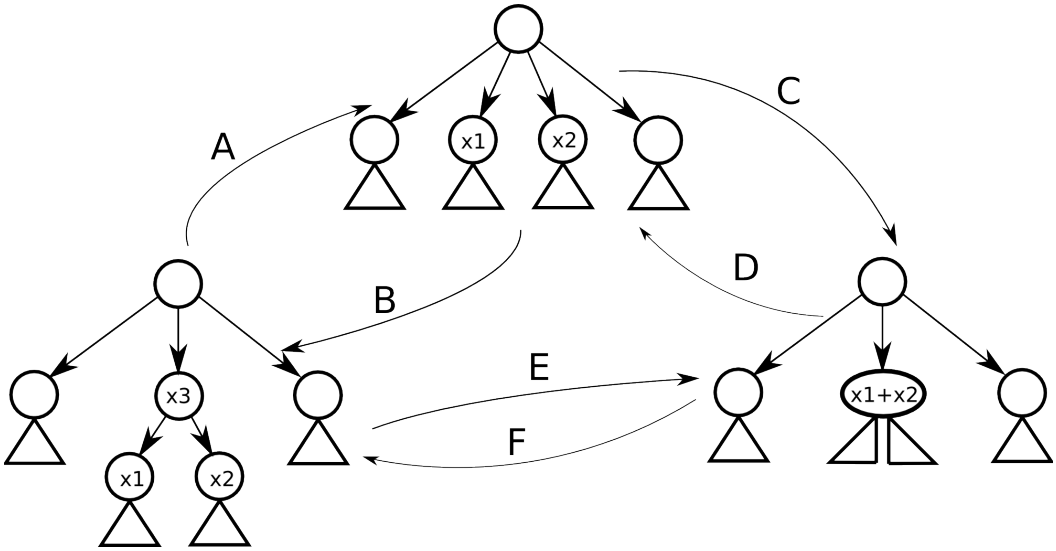
Lokalne transformacije nad drevesom, ki so potrebne za predelavo kakršnegakoli drevesa v poljubno drugo obliko, so ilustrirane na sliki 5.2, njihov pomen pa je opisan v tabeli 5.1. Od treh parov zadostujeta le poljubna dva, tretjega pa lahko simuliramo z zaporedjem drugih dveh.

Oznaka transformacije	Razlog za transformacijo
A	podobnost med x_1 in x_2 je primerljiva s podobnostjo med brati x_3
B	podobnost med x_1 in x_2 je signifikantno večja kot podobnost z ostalimi brati
C	vsota x_1 in x_2 nima notranje strukture v primerjavi z ostalimi brati
D	grozd $x_1 + x_2$ ima notranjo strukturo, primerljivo z ostalimi brati
E	struktura grozda x_3 ni očitna
F	$x_1 + x_2$ ima notranjo strukturo, vendar bolj fino kot je med njegovimi brati

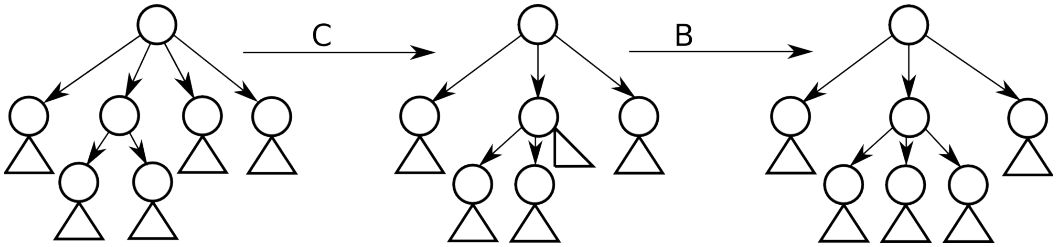
Tabela 5.1: Pomeni ilustriranih transformacij

Postopek spuščanja grozdov po drevesu — potrebno zaporedje navedenih trans-

formacij za dosego cilja — ilustrira slika 5.3, postopek dvigovanja pa je obraten.



Slika 5.2: Lokalne transformacije drevesa



Slika 5.3: Migracija med nivoji

Namen ilustracije teh transformacij je razmislek, da lahko s kombinacijo takšnih transformacij spremenimo drevo v poljubno drugo — če ne drugače, po en učni primer naenkrat, tako da vsak primer ali grozd z zaporedjem teh operacij prenesemo na želeno mesto.

Namesto dveh simetričnih parov operacij predlagam uporabo ene same, ki ob prisotnosti idealne odločitvene funkcije zadostuje za izvedbo katerekoli od zgornjih in posledično za poljubno transformacijo drevesa. Poleg posplošenega pristopa ter hitrejše konvergence k lokalnemu optimumu tudi poenostavi odločanje na eno samo vprašanje.

Namesto odločanja o združevanju ali pa razbijanju dveh otrok nekega vozlišča se lahko vprašamo o poljubni kombinaciji združitev ali razbitij teh otrok, z drugimi besedami o novi porazdelitvi vnukov med otroke, z dodatno možnostjo spremembe števila otrok.

Dvigovanje in spuščanje po drevesu je samodejno: če takšna porazdelitev uvrsti en sam element v svoj grozd, se ta element dvigne za eno nivo in postane otrok vozlišča, nad katerim izvajamo trenutno operacijo. Spuščanje je ravno obratno: vse otroke vozlišča, ki *niso* grozdi, obravnavamo za potrebe ponovne porazdelitve enako kot vnuke, s čimer že v osnovi začnejo na enem nivoju nižje (če pa taka razdelitev ni primerna, ob končanju prerazporejanja zgornje pravilo poskrbi da ponovno postanejo otroci tega vozlišča).

Ko se otroci nekega vozlišča spremenijo zaradi prerazporeditve vnukov, ponovimo postopek na njegovem staršu in tako navzgor vse do korenskega vozlišča. Če bi želeli še hitrejšo konvergenco v lokalno optimalno stanje drevesa, bi lahko takšno obdelavo izvedli tudi na novonastalih vnukih, vendar bi na tak način v vsakem koraku spreminjali celotno drevo, kar pa ni učinkovito s časovnega vidika.

Algoritem 1 Dodajanje točke v drevo

Vhod: Drevo D, Širina iskalnega snopa B

```
1  v = PoišciPrimernoVozlišče(širina_snopa = B)
2  dodaj točko v vozlišče v
3  dokler v obstaja, ponavljaj:
4     popravi povzetek v
5     preuredi otroke v
6     v = v.starš
```

Algoritem 2 Preurejanje otrok vozlišča

Vhod: Vozlišče v

```
1  zberi množico vnukov vozlišča v:
      vsi otroci otrok
      otroci, ki niso tipa grozd
2  pozabi otroke
3  izračunaj možna razbitja množice vnukov
4  izberi razbitje z ustrezno kompaktnostjo
5  ustvari nove otroke in jim priredi vnuke
6  če ima kakšen otrok le enega potomca, izbriši otroka in posvoji potomca
```

Odločanje o velikosti razbitja Odločanju o številu otrok, ki je samo po sebi težak problem že za nehierarhično razvrščanje, sem posvetil precej časa. Povprečna čistost grozdov v razbitju se povečuje s številom otrok, zato lahko formuliramo problem kot iskanje zgornjega še sprejemljivega števila grozdov.

V končni verziji algoritma zgornja meja števila otrok ni fiksna, ampak je omejena njihova velikost — dokler niso velikosti vseh novonastalih otrok pod neko konstanto, ki določa zgornji faktor razvejanosti drevesa, jih razbijamo po prioritetnem redu, ki da prednost večjim vozliščem. To da spodnjo (in potencialno tudi zgornjo) mejo na število otrok. Tudi če so otroci že dovolj majhni, pa še ni doseženo njihovo dovoljeno število, razbijanje ponavljamo, vendar je zdaj prednost vozlišč v čakalni vrsti za razbitje glede na njihovo kompaktnost. To da zgornjo mejo na število otrok, ki pa ni nujno višja kot tista, dobljena iz pogoja za največjo velikost posameznih otrok, saj je lahko število otrok že preseženo zaradi izpolnjevanja tega pogoja. Če spodnja in zgornja meja nista enaki, je s stališča zagotavljanja neizrojenega drevesa sprejemljiva katerakoli možnost med obema vrednostma. V eksperimentih sem v takem primeru izbral takšno število otrok, da je bilo razmerje med kompaktnostjo starša in popvrečno kompaktnostjo otrok (po enačbi 4.4), ter pa razmerjem med povprečno kompaktnostjo otrok in povprečno kompaktnostjo vnukov čimbolj podobno — z drugimi besedami, kompaktnost po hierarhiji naj narašča čimbolj enakomerno.

Ker število otrok vozlišča ni navzgor omejeno, bi se lahko zgodilo, da se število otrok v korenu povečuje čez vse smiselne meje. Da se izognem takšni situaciji, ima tudi korensko vozlišče starša, nad katerim izvedemo enako operacijo razbijanja kot pri ostalih vozliščih. V primeru, da ima po prerazporejanju vnukov (torej otrok korenskega vozlišča) več kot enega otroka, postane to vozlišče nov koren drevesa in dobi novega skritega starša.

5.2 Implementacija

Algoritem sem implementiral v jeziku C++, testno okolje pa v Pythonu.

Osnovna podatkovna struktura v algoritmu je izvedena kot drevo (slika 5.1) z dvema tipoma vozlišč; en tip predstavlja grozde, drug tip pa posamezne dokumente. Grozdi imajo lahko poljubno število otrok, ki so lahko kateregakoli tipa. V njih se nahaja tudi vsota vseh dokumentov (povzetek), ki so njihovi posredni ali neposredni potomci.

5.2.1 Učinkovita implementacija k-meansa

Za nerazvejano razvrščanje v koraku 3 postopka za preurejanje otrok, sem uporabil algoritem k-means.

k-means [11] je iterativni postopek za izračun razbitja množice na grozde. Število želenih grozdov k sprejme kot parameter. V začetnem koraku si izberemo k dokumentov, ki zdaj predstavljajo centroide začetnih grozdov, nato pa izmenično ponavljamo dva koraka. V prvem koraku vse dokumente priredimo grozdu, katerega centroid jim je najbližji; tako dobimo neko razbitje. V drugem koraku izračunamo nove centroide teh grozdov po enačbi 3.7. Ta dva koraka ponavljamo, dokler se prirejanje še spreminja, oziroma dokler sprememba kvalitete razbitja ne pade pod neko

vnaprej določeno mejo — tako se izognemo pretiranemu popravljanju razbitja brez velikega učinka. Običajno se navzgor omeji tudi samo število iteracij, s čimer dobimo časovno zagotovilo, kljub potencialnemu slabšemu razbitju. V splošnem (če vedno enako razrešujemo dodeljevanja v centroide, kjer je več enako dobrih možnosti) algoritem k-means konvergira k lokalnemu optimumu.

Algoritem 3 k-means

Vhod: Množica točk D, Število grozdov k

```
1  izberi množico centroidov C: k točk iz D
2  vsako točko v D priredi najbližjemu centroidu v C
3  če nek centroid nima prirejene nobene točke:
4     priredi mu točko iz D, ki leži najdlje od svojega centroida
5  izračunaj novo množico centroidov C kot množico povprečij grozdov
6  če še ni izpolnjen ustavitveni pogoj, se vrni na korak 2
```

Obstaja veliko načinov za izboljšanje tega osnovnega algoritma. Po ugotovitvah iz [9] sem povzel preprosto, a zelo učinkovito spremembo: namesto da se centroidi izračunajo šele, ko so na novo dodeljeni vsi dokumenti, se ob vsaki spremembi pripadnosti grozdu popravita centroida grozdov, med katerima je prešel dokument.

Skladno z ugotovitvami v [9] je tudi moja tako popravljena implementacija k-meansa v veliki večini primerov konvergirala že po enem prehodu čez vse učne primere, medtem ko je implementacija, ki temelji na algoritmu 3, za primerljivo kvaliteto potrebovala od pet do deset prehodov.

Začetne centroide sem poiskal z naključnim vzorčenjem vseh dokumentov. Med šestimi vzorci dokumentov sem izbral takšnega, ki je imel najmanjšo notranjo podobnost (4.1) — najbolj različne dokumente.

Ker k-means konvergira v lokalen optimum, sem za vsako razbijanje množice poskusil več možnih razbitij. Zgornja meja števila poskusov je bila deset, vendar sem, če se po dveh zaporednih poskusih utežena kompaktnost razbitja ni zmanjšala, postopek prekinil. V povprečju so se pri zaporednem dodajanju vseh dokumentov v korpusu izvedle približno štiri iteracije.

V procesu prerazporejanja dokumentov med grozdi se lahko zgodi, da ostane grozd brez dokumentov. V takem primeru mu takoj dodelim dokument, ki je najbolj oddaljen od svojega centroida.

Za zelo dobro izbiro se je izkazala tudi uporaba bisekcijskega k-meansa. Pri običajnem algoritmu k-means, kot je opisan v (Algoritem 3), je v vsaki iteraciji množica razbita na k segmentov. Bisekcijski k-meanspa spada pod algoritme hierarhičnega razvrščanja z razbijanjem. Prične z množico dokumentov in jo razbije na natanko dva grozda, potem pa rekurzivno vsakega od njiju razbije na dva nova in to ponavlja dokler ne ostanejo grozdi velikosti ena. S primernim rezanjem tako

nastalega drevesa si lahko naknadno izberemo razbitje za poljuben $k \in \{1 \dots n\}$ tako, da za grozde uporabimo le liste v porezanem drevesu, preostanek hierarhije pa ignoriramo. Takšno razbijanje se je izkazalo za zelo neobčutljivo na začetne pogoje.

Dodatna prednost tega pristopa je preprosto uveljavljanje zahteve po največji velikosti grozdov.

Potrebno je ločevati med hierarhijo, ki jo vzdržuje predlagani algoritem, in hierarhijo, ki jo med delovanjem proizvede bisekcijski k–means. Ko je sprejeta odločitev o številu in velikosti vnukov v koraku 4 algoritma 2, je rezultat nehierarhično razbitje, čeprav v samem postopku razbijanja v koraku 3 zgradi pomožno hierarhijo. Prav tako so lahko vozlišča v drevesu D ali dokumenti, ali pa grozdi, ki jih mora pomožni algoritem za razbijanje obravnavati, kot da so kompleksni dokumenti, saj jih ne sme razbiti naprej (število vnukov se v enem klicu algoritma 2 ne spremeni). Mera podobnosti, ki jo uporablja k-means, je tako neobičajna, saj mora primerjati tudi dokumente z grozdi in grozde med seboj. Preizkusil sem uporabo podobnosti med centroidi in pa UPGMA (obe opisani v 3.2) in se na podlagi primerjav odločil za mero med podobnosti centroidov. Posamezni dokumenti se obravnavajo, kot da so grozdi velikosti ena.

Končna verzija postopka za razvrščanje vnukov v algoritmu 2 je sledeča:

Algoritem 4 Bisekcijski k-means s prioritetnim razbijanjem

Vhod:
Množica točk D
Najmanjše število otrok k
Največa dovoljena velikost vnuka p

Lokalna spremenljivka:
Q = prioritetna vrsta množic točk:
urejena po velikosti, za množice nad velikostjo p
in po kompaktnosti, za množice manjše od p

```
1   dodaj D v Q
2   dokler
      Q ni prazna IN
      (listov v razbitju je manj kot k
      ALI prvi element v Q ima več kot p elementov
      ):
3     razbij vrhnji element E iz Q na dve podmnožici (in ga odstrani iz vrste):
4       za ta E največ 10× ponovi:
5         naključno izberi šest parov elementov iz E
6         par z najmanjšo podobnostjo naj predstavlja začetna centroida razbitja
7         dvakrat se sprehodi po množici E in za vsak element e ponovi:
8           e dodeli najbližjemu centroidu
9           temu centroidu prištej e
10          če je e že pripadal drugemu centroidu:
11            mu odštej e
12            če je centroid zaradi izgube e prazen:
13                dodeli mu točko, ki je najbolj oddaljena od svojega centroida
14      izberi razbitje z najvišjo povprečno kompaktnostjo
15    novonastali podmnožici dodaj v Q, če nista velikosti 1
16    zapomni si razbitje
```

Po vsakem koraku razbijanja ocenim trenutno povprečno kompaktnost listov v razbitju. Ker se vedno razbije le en list v drevesu, za to ni potreben prehod čez celotno drevo, ampak le odštevanje kompaktnosti razbitega vozlišča in prištevanje kompaktnosti novonastalih vozlišč.

Če je med izpolnitvijo posameznih disjunktivnih podpogojev iz točke 2 več korakov (torej da so nekaj časa vsi elementi Q že manjši od p, listov v razbitju pa še ni k), je načeloma primerno katerokoli razbitje s tega intervala, kot že opisano v prejšnem podpoglavju.

6

Eksperimenti

6.1 Splošno — eksperimentalni protokol

Obnašanje algoritma sem preveril na različnih podvzorcih iz zbirke *Reuters Corpus Volume 1*. Dokumenti v korpusu so že označeni s kategorijami, tako da sem za simulirani podatkovni tok lahko uporabil preprosto filtriranje datumsko urejenega zaporedja po znanih kategorijah.

Za hitro predstavo o smiselnosti predlaganega algoritma tu poleg numeričnih rezultatov prilagam tudi risbe nastalih dreves. Vsa drevesa so izrisana konsistentno: grozdi so ovalne oblike, dokumenti pa pravokotne. Barva dokumentov je funkcija njihove *dejanske* kategorije. Barva grozdov predstavlja razmerje med dejanskimi kategorijami dokumentov, ki so njihovi posredni ali pa neposredni potomci. Zaradi razumljivosti so izrisana samo drevesa, zgrajena na podlagi dveh ali treh kategorij; rdeča barva predstavlja eno kategorijo, modra drugo in zelena tretjo. Vijolična tako predstavlja grozd, ki vsebuje mešanico dokumentov, vzorčenih iz prvih dveh kategorij.

Na risbah je s K označena kompaktnost grozda, izračunana po enačbi 4.1. Pod kompaktnostjo je v oklepaju izpisana še distribucija dejanskih kategorij dokumentov v grozdu; v ta namen so kategorije označene z -1, 0 in 1. V korenu drevesa je z DL_p označena tudi povprečna dolžina opisa za drevo, glede na enačbo 4.10. Vsa vozlišča imajo še oznako, ki ustreza označbi v podatkovni množici.

Abscisa, označena s t, predstavlja na vseh grafih impliciten čas prihoda učnih primerov v model.

Kot sem omenil že v poglavju 4.2.2, mera, ki jo uporabljam v vseh poskusih v tem poglavju, ni idealna. Veliko izrednih vrednosti na grafih v resnici ne predstavlja napak v drevesih ampak so posledica težav z mero. Dolgoročno povprečje mere je veliko bolj primerno za ocenjevanje lastnosti algoritma, zato so pri eksperimentih navedena povprečja in priloženi grafi, ugotovitve pa sem preveril tudi na dejanskih drevesih, ki so nastala v postopku.

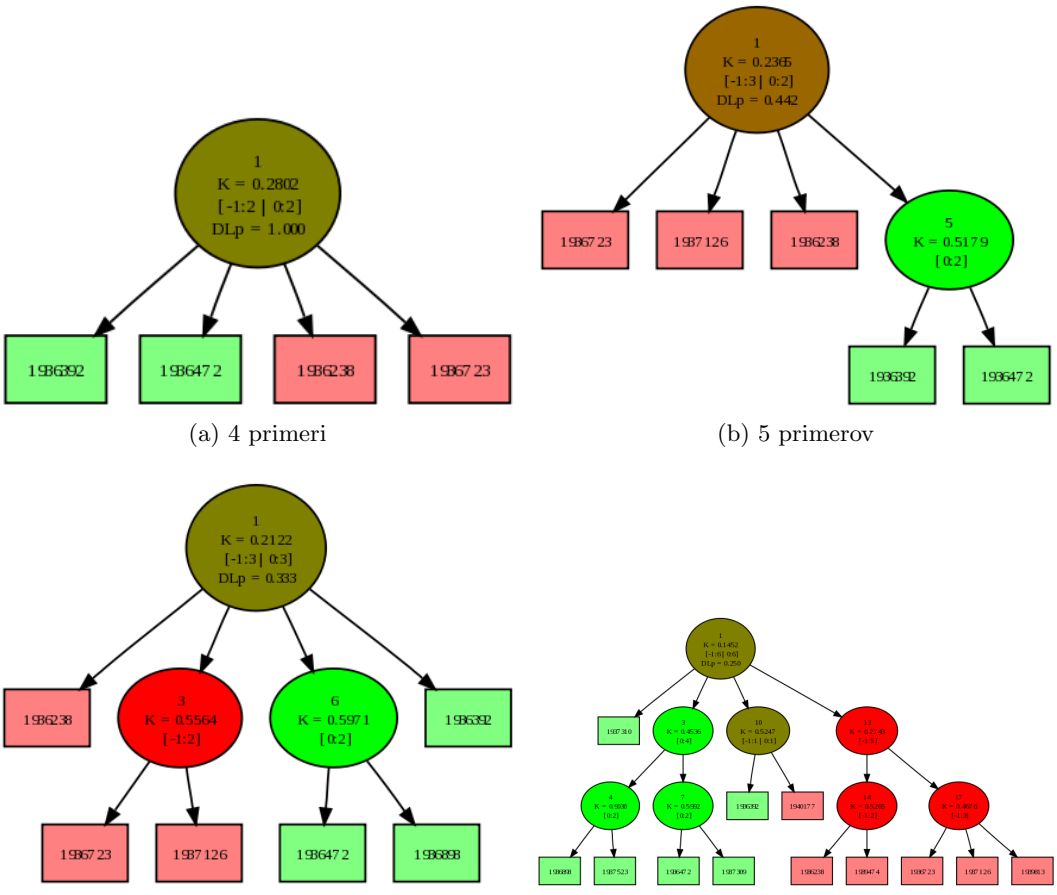
Pri vseh eksperimentih, razen kjer je eksplicitno navedeno, sem zaradi primerljivosti omejil razvejanost na tri otroke.

6.2 Osnovno obnašanje

Osnovno obnašanje algoritma pri dodajanju točk v model z omejitvijo stopnje razvejanosti 4 ilustrira slika 6.1. Ta, kot tudi vse ostale slike dreves prikazujejo dejanska izračunana drevesa z uporabo predlaganega algoritma.

Ko število učnih primerkov preseže največjo dovoljeno število v enem grozdu, se primeri iz kategorije 0 združijo v nov podgrozd.

Kljub majhnemu drevesu pa že ta slika prikazuje načeloma pravilno delovanje osnovnih komponent algoritma.



(a) 4 primeri

(b) 5 primerov

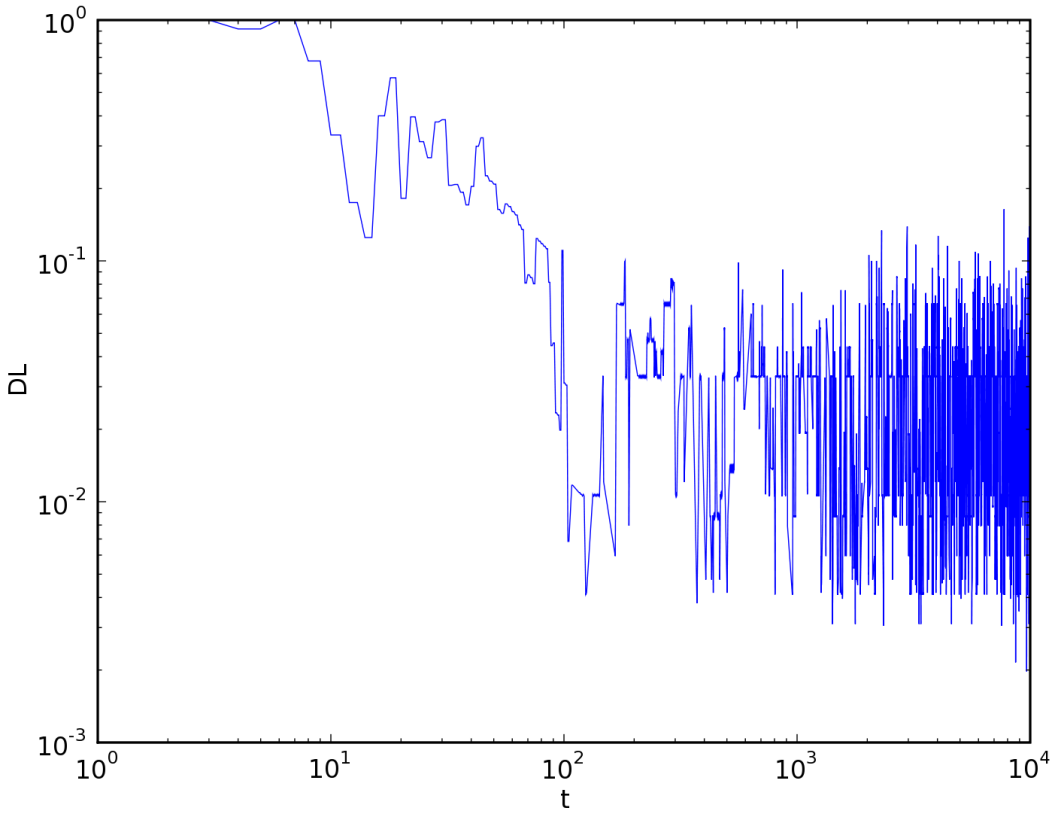
(c) 6 primerov

(d) 12 primerov

Slika 6.1: Drevo po nekaj dodanih primerih

6.3 Konvergenca na končni množici

Pričakujemo tudi, da bi lahko tak pristop uporabili za razvrščanje končne in statične množice dokumentov. Zaradi zelo lokalne optimizacije seveda ne pričakujemo, da bi že en sam prehod čez množico dokumentov zgradil pravilno drevo.



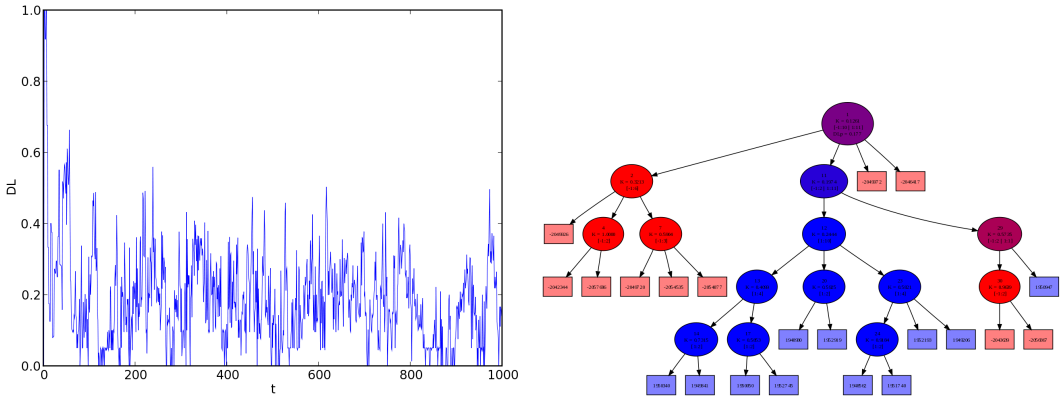
Slika 6.2: Stabilnost DL v odvisnosti od števila dodanih primerov

Graf 6.2 prikazuje nečistočo drevesa kot povprečno dolžino opisa primerov v poskusu, kjer sem vnaprej naključno izbral po trideset primerov iz dveh disjunktnih kategorij (kategoriji 'GFAS' (moda) in 'G15' (Evropska Unija)), potem pa jih zaporedno dodajal v model, katerega velikost je bila omejena na 60 dokumentov. Dodajanje sem ponavljal toliko časa, da je bilo v drevo dodanih 10.000 primerov. Dolgoročno povprečje je 0,017 bita na primer; z grafa pa se vidi, da se po približno dveh prehodih (pri $t\tilde{=}120$) čez učno množico model že stabilizira v okolici te vrednosti. Zaradi logaritemske skale na abscisi so razvidni tudi ekstremi obnašanja na dolgi rok. Povprečna nečistoča drevesa, zgrajenega z *običajnim* algoritmom za razvrščanje z razbijanjem grozdov na štiri podgrozde z uporabo k-means pa je za isto podatkovno množico 0,012 bita. V podobnem eksperimentu na kategorijah

'GCRIM' (kriminalna dejanja) in 'GDIS' (naravne nesreče) je dosegel predlagani algoritem nečistočo 0,038, hierarhični k-means pa 0,031.

Kvaliteta razbitja tako ne zaostaja veliko za razbitjem, ki je izračunano v enem koraku.

6.3.1 Stabilnost



(a) Nečistoča (b) Tipično drevo

Slika 6.3: Stabilnost nečistoče pri neomejeni zalogi primerov

Za razliko od prejšnjega poskusa prikazuje graf na sliki 6.3 nečistočo drevesa v poskusu, kjer sem iz neomejene zaloge dokumentov iz dveh kategorij ('GSCI' (znanost) in 'GENV' (okolje)) izmenično vzorčil naključne dokumente in jih dodajal v drevo, omejeno na dvajset dokumentov. Povprečje nečistoče je 0,173 bita, slika 6.3 (b) pa prikazuje tudi tipično drevo iz tega zaporedja.

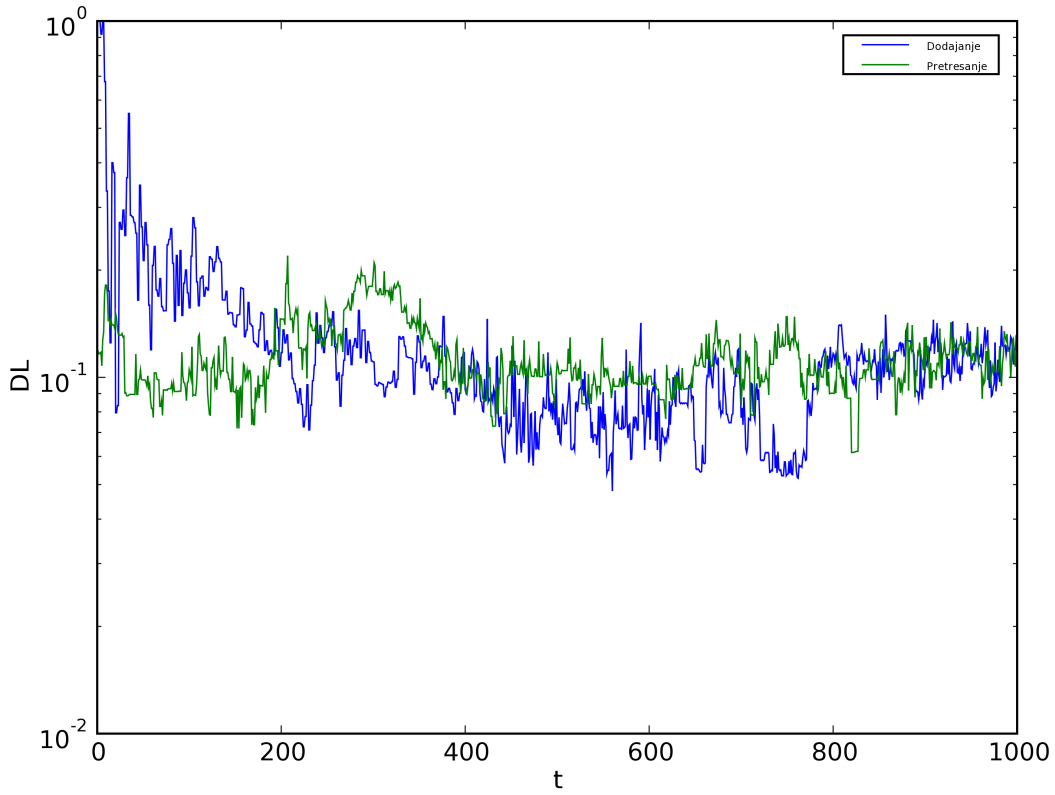
Vpliv posameznih neustrezno uvrščenih dokumentov na končno mero se zmanjša s povečanjem drevesa. Pri drevesu omejenem na sto dokumentov je povprečna nečistoča le še 0,113; za graf glej sliko 6.4.

Na podlagi tega eksperimenta in pa tudi na podlagi kasnejših eksperimentov s podobno zasnovo se zdi, da model uspešno ohranja dobršen del kontinuitete strukture med posameznimi koraki.

6.3.2 Konvergenca z inkrementalnim izboljševanjem

Ker je korak popravljanja drevesa neodvisen od dodajanja in odstranjevanja primerov, se lahko vprašamo, ali se da kvaliteto drevesa izboljšati tako, da si naključno izberemo neko vozlišče in od tega vozlišča navzgor izvedemo algoritem 2; torej pretresanje drevesa.

Graf 6.4 prikazuje nečistočo drevesa v tisoč korakih dodajanja primerov iz kategorij 'GSCI' in 'GENV' v drevo velikosti sto elementov, zelena črta pa prikazuje



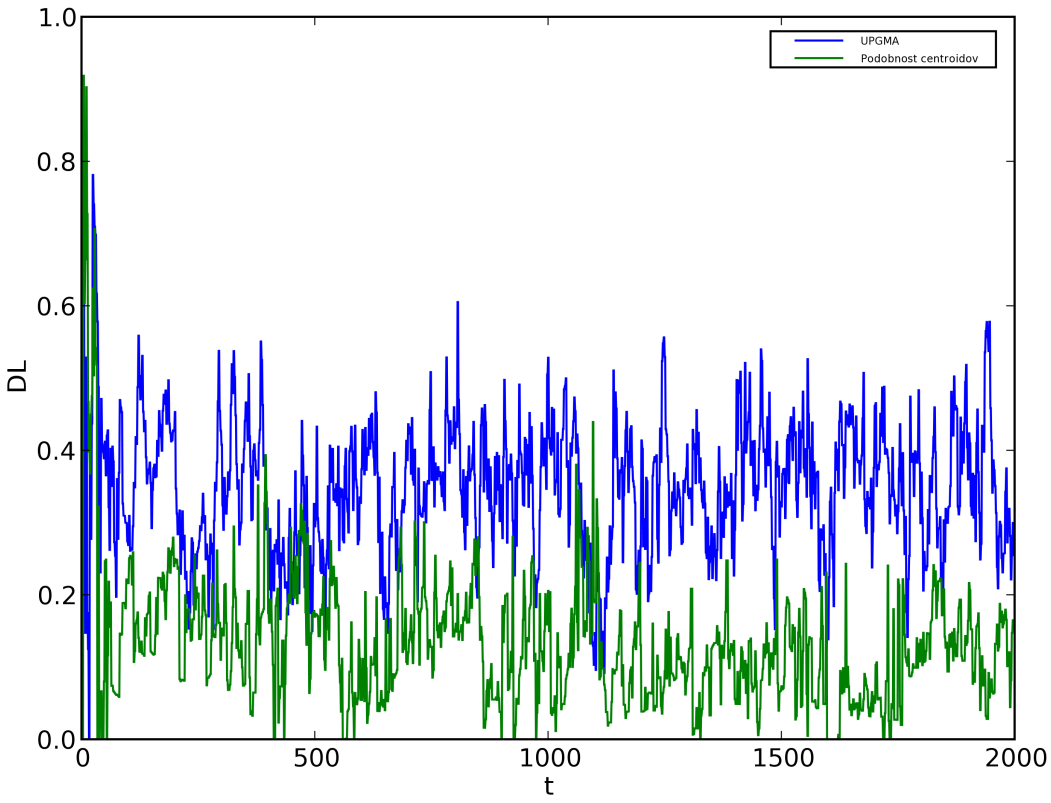
Slika 6.4: Izboljšava s pretresanjem
Modra črta prikazuje nečistočo drevesa v 1000 korakih dodajanja novih primerov, zelena črta pa *nadaljevanje* poskusa, 1000 korakov pretresanja tako nastalega drevesa

sledečih tisoč korakov naključnega izbiranja grozdnih listov drevesa in izvajanja algoritma 2 nad njimi in celotno potjo do korena drevesa. Povprečje v prvem primeru je 0,121, v drugem pa 0,113, vendar se razlika izniči, če pri izračunu zanemarimo prvih dvesto primerov, ko drevo še ni polno in stabilno.

Lahko ugotovimo, da prekomerno pretresanje drevesa vsaj v tem primeru nima občutnih koristi.

6.4 Uporaba različnih mer podobnosti

Če namesto uporabe podobnosti med centroidi grozdov pri razvrščanju v drevesu uporabimo mero UPGMA, se, kot že omenjeno v poglavju 3.1, nečistost drevesa močno poveča.



Slika 6.5: Razlike pri uporabi različnih mer podobnosti

Z grafa 6.5 je razvidna razlika v nečistoči drevesa, če za mero podobnosti v algoritmu razvrščanja namesto podobnosti med centroidi grozdov uporabimo mero UPGMA. Za eksperiment sem v drevo izmenično dodajal zaporedne primere iz kategorij 'ECAT' in 'CCAT', ki pokrivata ekonomijo ('ECAT') in novice iz sveta kor-

poracij ('CCAT'). Novice, ki pripadajo obema kategorijama (približno 8 odstotkov vseh) sem preskočil.

6.5 Vpliv raznih komponent

6.5.1 Uteževanje podatkov

Pri obdelavi tekstovnih podatkov je običajno, da namesto predstavitve TF uporabimo TFIDF ali pa sorodne načine za zmanjševanje vpliva neinformativnih besed (glej poglavje 3.1). Za izračun IDF vrednosti posameznih besed pa potrebujemo že v naprej znano distribucijo pogostosti besed. Ker ta pristop seveda ni primeren za obdelavo podatkovnih tokov, kjer ne poznamo prihodnosti, se uteževanju nisem veliko posvečal.

Poskusil sem dva para različnih adaptacij osnovnega TFIDF pristopa (opisanega v poglavju 3.1).

Para se razlikujeta po tipu uteževanja. V prvem paru sem uporabil uteževanje z razmerjem dokumentov, v katerih se beseda pojavi, glede na vse dokumente, v drugem paru pa z logaritmom tega razmerja.

Poskusa v paru se razlikujeta po tem, da v enem primeru besede, ki sestavljajo dokumente, ki jih zaradi zastaranja odstranjujem iz drevesa, odstranim tudi iz statistike pogostosti, v drugem primeru pa v statistiko podatke le dodajam.

Na grafu 6.6 so prikazani vse različne metode na istih podatkih (kategoriji 'GVIO' (nasilje) in 'GHEA' (zdravje), drevo velikosti dvesto primerov). Metodi z odstranjevanjem primerov iz statistike sta v legendi označeni z dodanim '-X', z logaritemskim uteževanjem pa z 'LOG'.

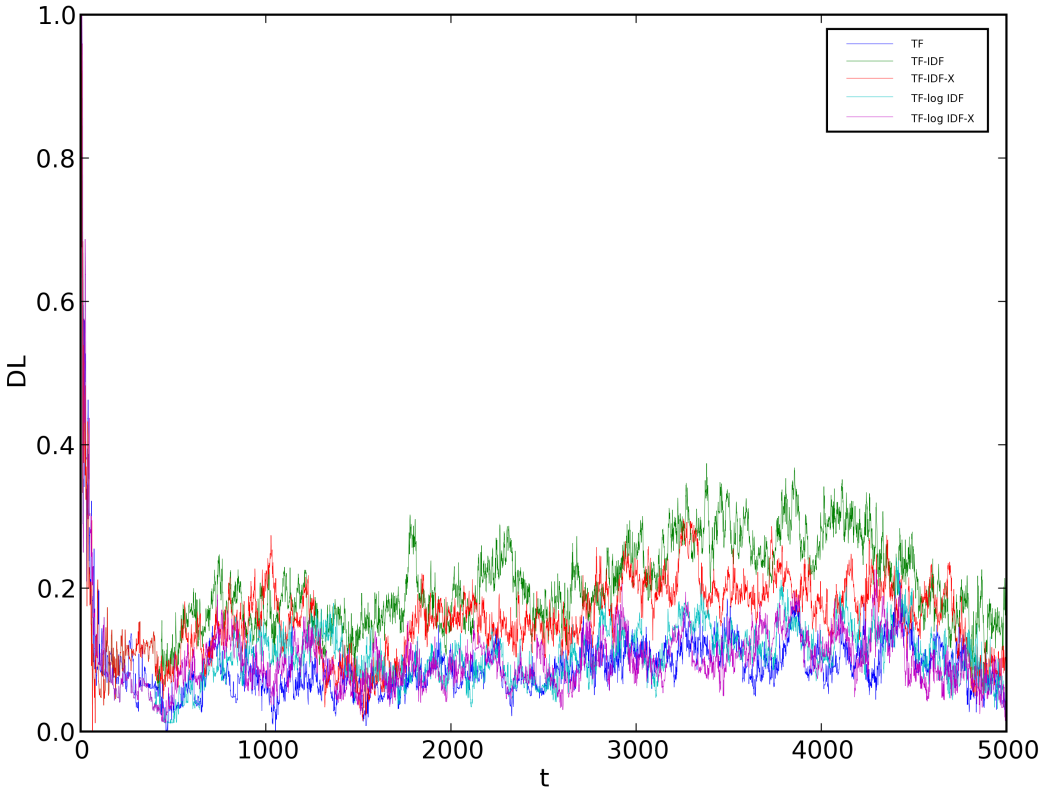
Ugotovitev na podlagi eksperimenta je, da vsaj za majhna drevesa in kratka zaporedja uteževanje s tako izbranimi faktorji IDF poslabša nastala drevesa; še najbolje pa se je obneslo obteževanje z logaritmom razmerja dokumentov.

6.5.2 Širina iskalnega snopa

En od parametrov algoritma je tudi širina snopa, ki se uporablja za iskanje grozda, v katerega se doda primer. Motivacija za uporabo iskalnega snopa s širino več kot ena je, da se izognemo potencialnim lokalnim maksimumom podobnosti.

Vpliv različnih širin iskalnega snopa je demonstriran na paru kategorij 'GHEA' (zdravstvo) in 'GENV' (okolje) na grafu 6.7 in pa v tabeli 6.1.

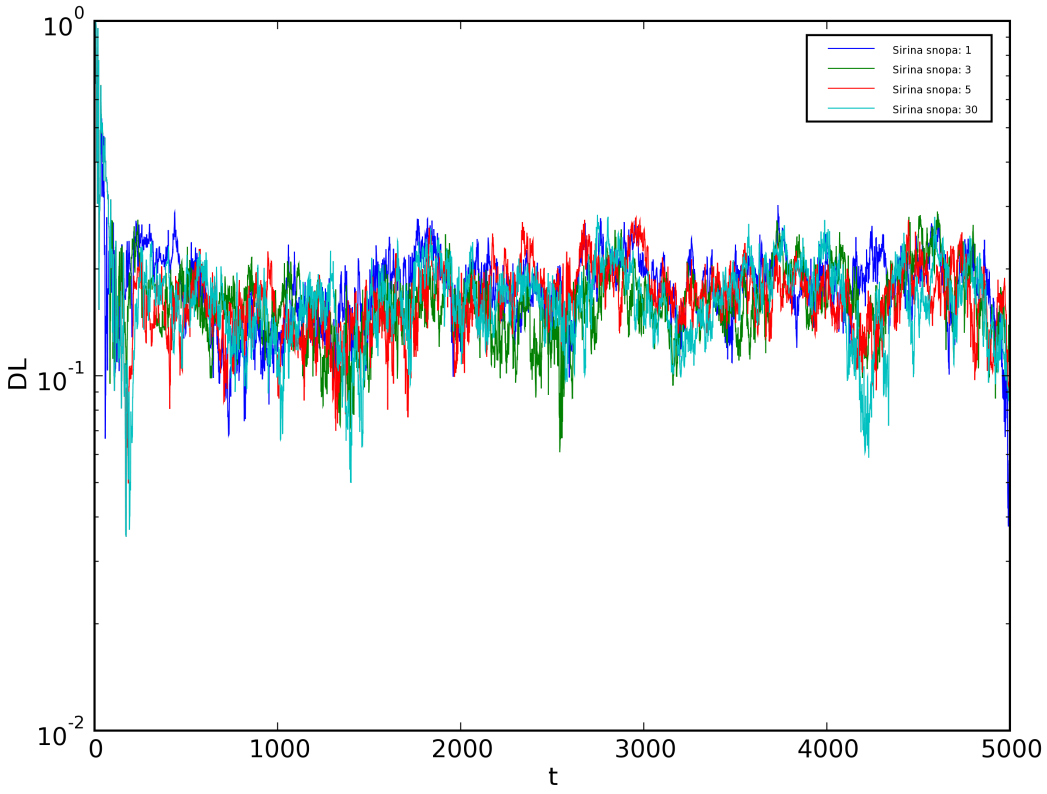
Kvaliteta dreves se s povečevanjem širine iskalnega snopa v splošnem nekoliko dvigne, vendar efekt ni velik, in mogoče tudi ne signifikanten. Zaradi praktično zanemarljivega vpliva širine snopa na časovno zahtevnost, pa pretirane škode zaradi uporabe verjetno ni.



Slika 6.6: Primerjava različnih načinov uteževanja besed

Kategoriji / širina snopa	1	3	5	30
GDIP:GSPO	0,020	0,021	0,020	0,022
GHEA:GENV	0,182	0,167	0,170	0,168
GCAT:CCAT	0,197	0,194	0,193	0,188

Tabela 6.1: Vpliv širine iskalnega snopa



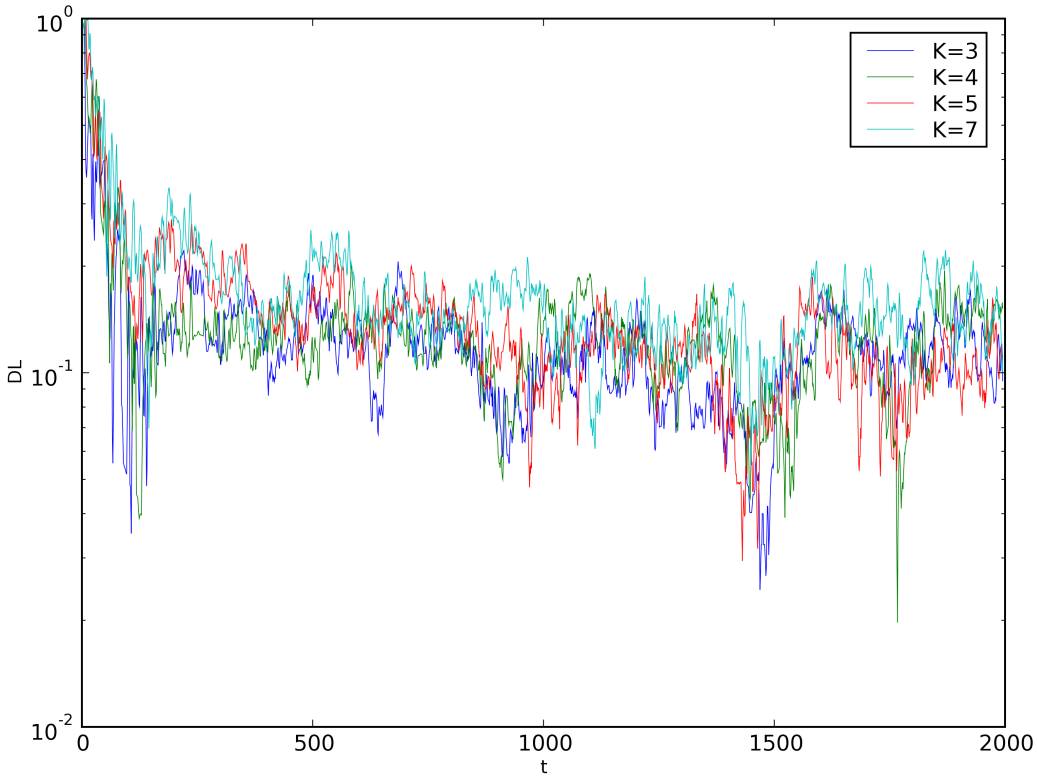
Slika 6.7: Vpliv različnih širin iskalnega snopa

Kategoriji / K	3	4	5	7
GSCI:GENV	0,123	0,134	0,145	0,172
GSPO:GDIP	0,032	0,032	0,034	0,039

Tabela 6.2: Vpliv omejitve razvejanosti

6.5.3 Stopnja razvejanosti

Vpliv različnih omejitev na najvišjo dovoljeno stopnjo razvejanosti v drevesu je prikazan na paru poskusov v tabeli 6.2 in na grafu 6.8. Kot pričakovano, imajo drevesa z nižjo stopnjo razvejanosti nižjo nečistočo. Na podlagi preiskovanja izrisanih dreves sklepam, da je to zato, ker je v drevesu z nižjo stopnjo razvejanosti za določeno število dokumentov več grozdov, ki se lahko bolje prilagodijo podatkom. Ugotavljanja možnosti prekomernega prilagajanja tu nisem raziskal.



Slika 6.8: Vpliv stopnje razvejanosti

6.6 Sledenje spremembam distribucije

Glavna lastnost predlaganega algoritma je sposobnost prilagajanja strukture glede na spremembe v vhodni porazdelitvi. Sledeča eksperimenta sta namenjena proučevanju te lastnosti.

6.6.1 Sledenje spremembam vhodne porazdelitve

Najprej nas zanima, kako uspešno se drevo prilagaja, če se v toku dokumentov začne pojavljati nova tema.

Za ta poskus si izberem tri različne kategorije, število korakov in pa razmerje med kategorijami v prvem in zadnjem koraku. V vsakem koraku dodam v drevo po en primer, ki ga z naključnim vzorčenjem iz navedenih kategorij (upoštevam le primere, ki se ne pojavljajo v dveh ali vseh treh kategorijah istočasno) izberem glede na razmerje v tem koraku. Razmerje med kategorijami v vmesnih korakih je dobljeno z linearno interpolacijo med začetnim in končnim.

Na grafu 6.9 je ilustrirana čistost drevesa največje velikosti 200 dokumentov v 4000 korakih eksperimenta, kjer je začetna porazdelitev enakomerna iz kategorij ‘GSCI’ (znanost) in ‘GREL’ (religija), končna pa enakomerna med ‘GSCI’ in ‘GENT’ (razvedrilo).

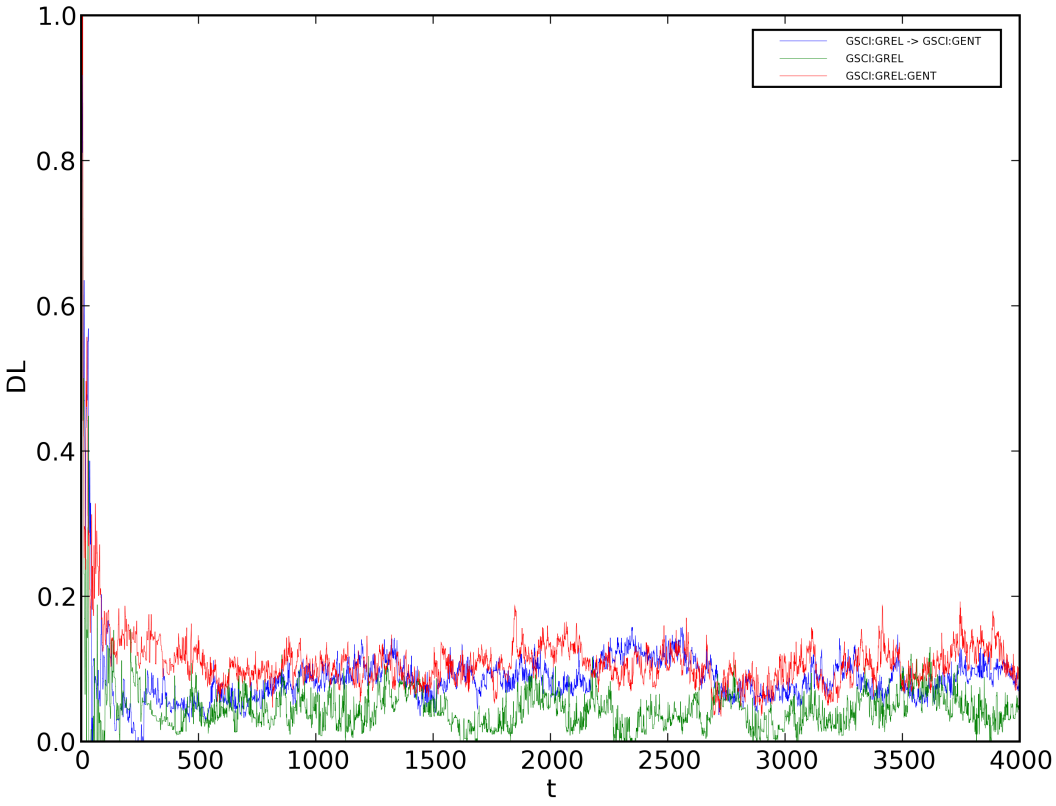
Če drevo ne bi ustrezno sledilo spremembam, bi bili primeri iz kategorije ‘GENT’ neustrezno uvrščeni med druge tematike, zaradi česar bi se morala močno povečati povprečna dolžina opisa, kar pa se ne zgodi. Za primerjavo sta narisana še primera, kjer sta začetna in končna distribucija enaki, v prvem enakomerna med ‘GSCI’ in ‘GREL’, v drugem pa enakomerna med vsemi tremi. Povprečna dolžina opisa, ko se distribucija spreminja, je za ta graf 0,087, za primer stacionarne porazdelitve med dvema kategorijama 0,045, za zadnji primer pa 0,110 — tudi iz grafa je razvidno, da je drevo, kjer se porazdelitev spreminja, redko slabše od drevesa, zgrajenega iz enakomerne mešanice vseh treh kategorij.

Za ilustracijo dogajanja v takšnem modelu je na sliki 6.10 ilustriran razvoj podobnega eksperimenta, le število korakov je zmanjšano na 400 in velikost drevesa na 30. Izbral sem takšna drevesa, pri katerih je povprečna dolžina opisa tipična, ne pa najboljša.

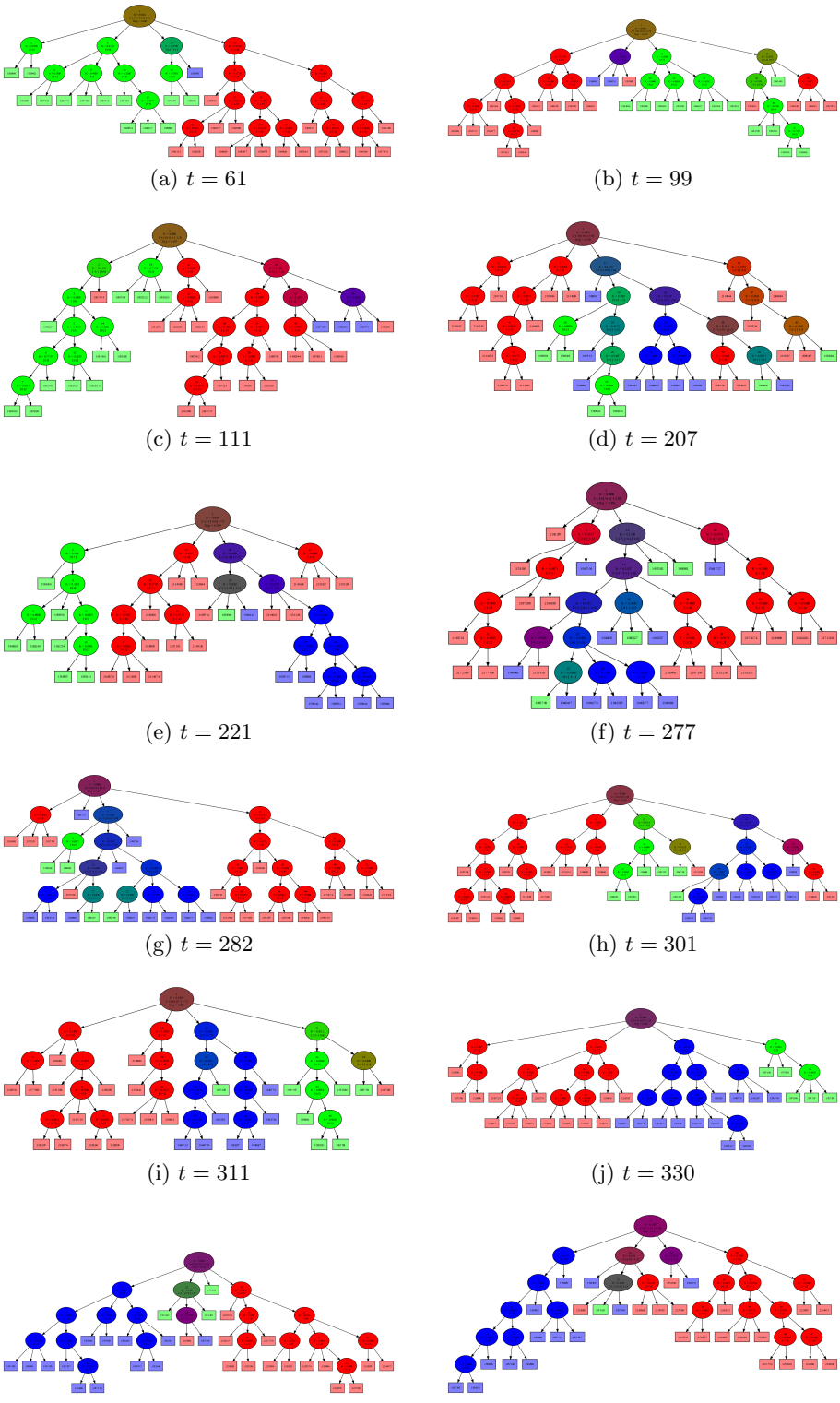
6.6.2 Zaznavanje redkih dogodkov

Vprašanje, ki se postavi ob nepregledni množici podatkov je tudi, ali znamo s pomočjo takšnega modela iz zaporedja izbrati najbolj *zanimive* primere. Zanimivost je seveda v očeh opazovalca; za ta eksperiment so zanimivi primeri takšni, ki verjetno ne prihajajo iz distribucije, ki je osnova za trenutni model, kot so npr. posebne novice, prvi opisi novih dogodkov, ...

Pristop, ki sem ga preizkusil, temelji na ideji, da naj bi se kompaktnost grozda, ki



Slika 6.9: Nečistoča drevesa med spreminjanjem vhodne porazdelitve



(a) $t = 61$ (b) $t = 99$

(c) $t = 111$ (d) $t = 207$

(e) $t = 221$ (f) $t = 277$

(g) $t = 282$ (h) $t = 301$

(i) $t = 311$ (j) $t = 330$

(k) $t = 371$ (l) $t = 396$

Slika 6.10: Razvoj drevesa nestacionarne distribucije

Kategorije A+B:C	razlika A	razlika B	razlika C
CCAT+MCAT:GCAT	0,1067 (0,0800)	0,1064 (0,0742)	0,1473 (0,0665)
CCAT+MCAT:M14	0,1081 (0,0781)	0,1023 (0,0768)	0,1245 (0,0691)
CCAT+MCAT:M143	0,1060 (0,0804)	0,1017 (0,0740)	0,1146 (0,0691)
MCAT:M14	0,1617 (0,1349)	-	0,2901 (0,1587)
MCAT:M143	0,1507 (0,1358)	-	0,2474 (0,1424)

Tabela 6.3: Razlike kompaktnosti grozdov pred in po dodajanju

Kategorije A+B:C	kvocient A	kvocient B	kvocient C
CCAT+MCAT:GCAT	1,2272 (0,1567)	1,2185 (0,1415)	1,3401 (0,1450)
CCAT+MCAT:M14	1,2357 (0,1578)	1,1969 (0,1401)	1,2946 (0,1385)
CCAT+MCAT:M143	1,2313 (0,1602)	1,2113 (0,1437)	1,2549 (0,1453)
MCAT:M14	1,2460 (0,2036)	-	1,4949 (0,2535)
MCAT:M143	1,2386 (0,2076)	-	1,4184 (0,2153)

Tabela 6.4: Kvocienti kompaktnosti grozdov pred in po dodajanju

sprejme takšen zanimiv učni primer, zmanjšala bolj, kot pa če bi bil primer podoben že obstoječim.

Podatke za eksperiment sem zgeneriral tako, da v tok podatkov, ki izvira iz stacionarne enakomerne porazdelitve para kategorij, občasno dodam primer iz povsem druge kategorije. Drevo je vedno omejeno na 200 primerov, redkih dokumentov pa je en odstotek.

V tabelah 6.3 in 6.4 so za vsak poskus, kjer je enakomerna mešanica dveh kategorij (A+B) predstavljala 'ozadje', kategorija C pa 'zanimive' dogodke, navedene razlike in razmerja kompaktnosti grozdov pred in po dodajanju primera izbrane kategorije. Kategorije 'MCAT' (trgovanje), 'M14' (trgovanje z dobrinami) in 'M143' (trgovanje z energijo) so vedno bolj specializirane kategorije ene same tematike; zanimiv dogodek tako npr. predstavlja 'trgovanje z energijo' v toku novic o trgovanju. V oklepaju je navedena tudi standardna deviacija.

Povprečna razlika in kvocient sta pri bolj specifičnih kategorijah sicer večji, vendar je standardna deviacija tako velika, da tak pristop verjetno ni primeren za izbiranje zanimivih primerov.

6.7 Učinkovitost

Ker naj bi se ta algoritem uporabljal za obdelavo velikih količin podatkov, je pomembna tudi časovna učinkovitost pristopa.

Ker je za vsa vozlišča v drevesu zagotovljeno, da imajo navzgor omejeno število otrok, pričakujemo, da bo zahtevnost dodajanja in odstranjevanja primerov v pov-

prečju odvisna od logaritma števila primerov, ki so že v drevesu — za izrojena drevesa pa seveda linearna. Resnično izrojenih dreves pa niti ne pričakujemo, saj daje algoritem prednost visoki stopnji razvejanosti. Ob sprehodu po drevesu navzgor se zaradi zbiranja vnukov več manjših otrok združi v enega, tako da se izrojena drevesa samodejno uravnotežijo.

Za test časovne zahtevnosti (in pa istočasno za eksperiment strategije določanja števila otrok) sem algoritem 'pokvaril' tako, da je dopuščal tudi večjo stopnjo razvejanosti in omejil le najvišje in najnižje še dovoljeno razmerje kompaktnosti otrok glede na starše (če so bili otroci za manj kot nek faktor bolj kompaktni kot starš, sem jih razbil — in obratno pri združevanju) — tako je lahko nastal graf, v katerem je imel posamezen grozd poljubno veliko dokumentov, še posebno če so si bili zelo podobni.

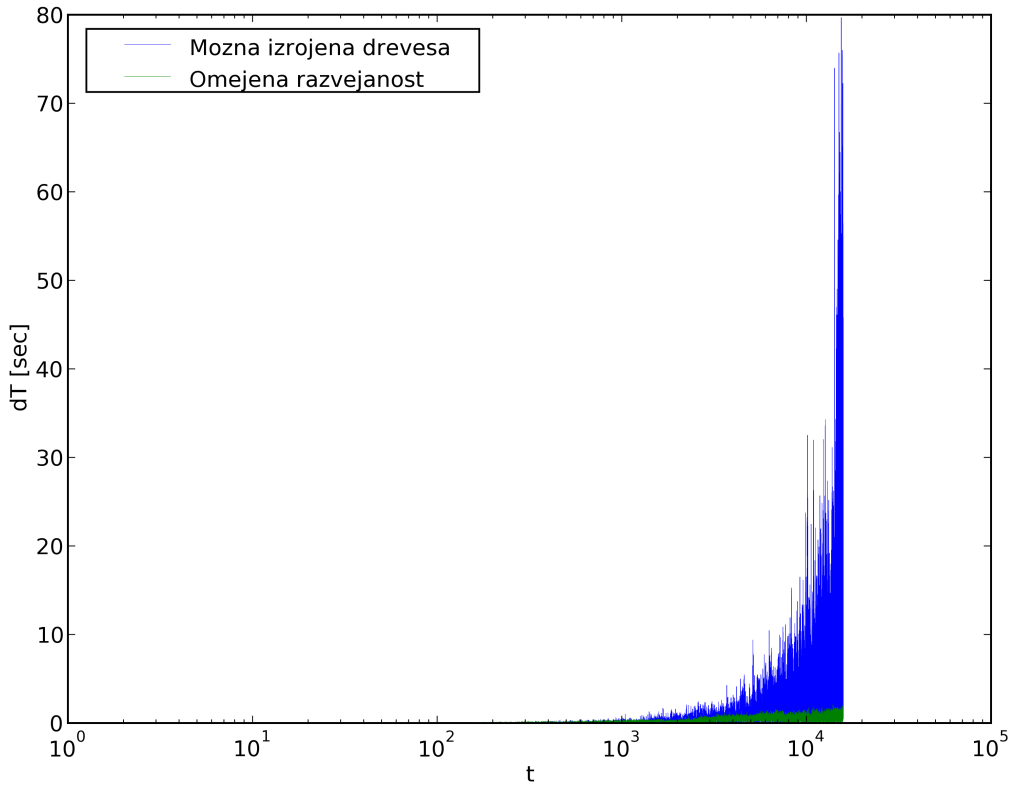
Graf 6.11 prikazuje časovno zahtevnost dodajanja v drevo. Za ta eksperiment je velikost drevesa neomejena, tako da drevo neprestano raste. Kot pričakovano, se v primeru, da število podgrozdov ni trdo omejeno, zahtevnost dodajanja povečuje s številom primerov, če pa razvejanost omejimo, raste zahtevnost približno z logaritmom števila primerov, kar se lepo vidi na grafu z logaritemsko absciso. Graf kumulativnega časa prikazuje tudi primer, kjer je razvejanost namesto na tri omejena na pet otrok.

Pogled v statistiko porabljenega časa razkrije sledeče:

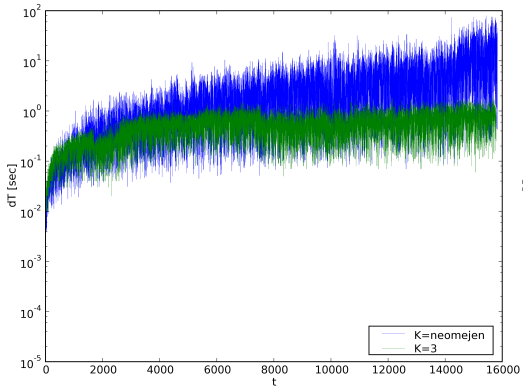
Odstotek časa	Funkcija
65,56%	podobnost med centroidi
31,91%	seštevanje in odštevanje vektorjev
0,94%	kompaktnost grozda
0,82%	evaluacija povprečne dolžine opisa razbitja
0,11%	k-means
...	...

Tabela 6.5: CPU čas po funkcijah

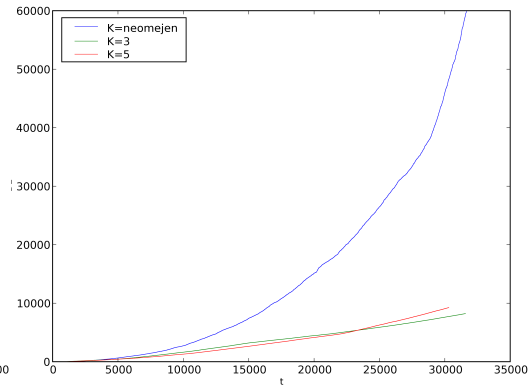
Ker je izračun podobnosti centroidov le množenje dveh vektorjev, ki predstavljata grozda, porabi algoritem okrog 99 odstotkov časa le za aritmetične operacije nad vektorji. Moja implementacija množenja in seštevanja redkih vektorjev je povsem naivna — s kazalcem se sprehodim čez oba vektorja in elemente, ki se ujemajo seštejem oziroma zmnožim. Analiza izvajanja pogojnih stavkov pokaže, da se pri zgoraj opisanem eksperimentu (z omejitvijo razvejanosti in $k = 3$) pri seštevanju ujema manj kot 4% elementov, pri množenju pa okrog 20%. Z drugačno implementacijo redkih vektorjev, na primer z uporabo dodatne razpršene tabele za neposreden dostop do posameznih elementov, ali pa z drugačnim opisom dokumentov se da torej algoritem še močno pospešiti.



(a) Časovna zahtevnost dodajanja enega primera



(b) Časovna zahtevnost dodajanja enega primera



(c) Čas od začetka izvajanja poskusa

Slika 6.11: Časovna zahtevnost dodajanja v drevo na različnih skalah

7

Zaključek

Predstavljeni algoritem po vseh eksperimentih in dosedanjih izkušnjah načeloma ustreza zastavljenemu problemu — samodejnemu odkrivanju strukture v neomejenih, spreminjajočih se virih besedil kot so novice, spletni dnevniki in podobnih.

Zaradi nizke asimptotske časovne odvisnosti je primeren za obdelavo velikih količin podatkov, veliko število možnih sprememb strukture drevesa tekom postopka dodajanja ali odstranjevanja le enega samega primera pa omogoča hitro prilagajanje spremembam porazdelitve vira, kar je tipično pri novicah, kjer lahko en sam dogodek povzroči več tisoč objav o isti temi. Vzrok za to je uporaba ploskega razvrščanja na vsakem nivoju v drevesu namesto samostojnih odločitev o združevanju ali razbijanju posameznih vozlišč. Ta lastnost ima tudi slabo stran, saj imajo vozlišča in povezave v drevesu relativno kratko življenjsko dobo, zaradi česar je smiseln neposreden prikaz več zaporednih dreves brez dodatnega dela skorajda nemogoč.

Zaradi velike mobilnosti grozdov tudi nisem uporabil staranja, kot je opisan v poglavju 2.2.1: nove teme lahko neovirano formirajo nove grozde, pa tudi z vidika prostorske učinkovitosti s staranjem vozlišč ne bi pridobil nič, če pričakujemo, da so podatki, ki jih drevo povzema, še vedno dostopni uporabniku. Ker vzdrževanje poddrevesa, ki vsebuje le stare tematike, ki se ne prekrivajo z novimi, praktično nič ne stane, je to še bolj privlačna izbira.

Od domene uporabe je odvisna tudi obravnava časa. V opisanih eksperimentih sem čas (tu le kot zaporedno številko dokumenta, lahko pa bi bil tudi izražen v sekundah) uporabil le za odločanje o tem, kdaj dokumenti zastarajo in so odstranjeni iz drevesa; lahko pa bi ga uporabili tudi v definiciji mere podobnosti. To bi bilo smiselno, če so besedila dobro lokalizirana v času, kot je to običajno pri novicah; pri blogih, ki pa se po večini ne ukvarjajo s takšnimi dnevnimi dogodki, ki bi bili skupni večji skupini ljudi, pa čas verjetno nima posebnega pomena. Ena možnost bi bila, da bi za čas, kot v CluStreamu, vodili povprečje in standardno deviacijo grozda, odstopanje pa upoštevali v izračunu podobnosti.

Ugotovil sem, da predstavlja uporaba iskanja najboljšega vozlišča s snopovnim

preiskovanjem zelo majhno upočasnitev algoritma, istočasno pa zagotavlja osnovno izogibanje lokalnim optimumom, zato je uporaba tega pristopa smiselna. Predvidevam, da je to še bolj res za zelo velika drevesa, kjer je verjetnost za lokalne optimume večja, cena preiskovanja pa se povečuje le z logaritmom števila dokumentov v drevesu.

Največja težava, s katero sem se ukvarjal tekom razvoja algoritma, je odločanje o razvejanosti na posameznih nivojih. Najprej sem poskusil minimizirati uteženo vsoto notranje podobnosti grozdov in njihove velikosti in različne tipe razmerij kompaktnosti med starši in otroci v drevesu, a so bila zgrajena drevesa precej naključna ali pa izrojena. Na koncu sem se odločil za fiksno omejitev zgornjega števila otrok, kar se zdi pretirano preprosto, a vsaj zagotavlja neizrojena drevesa. Iskanje boljše odločitvene funkcije bo osnova za nadaljnje raziskovanje.

Postopek porabi praktično ves čas za seštevanje in primerjanje vektorjev. Dokler se za opis dokumentov uporabljajo redki vektorji, je paralelizacija večjih delov algoritma nepraktična, nekajkratno pospešitev pa bi lahko dosegli z odstranjevanjem besed, ki imajo utež manjšo od določenega praga, iz posameznih vektorjev. Obstaja tudi možnost, da se dokumente najprej projicira v prostor nižjih dimenzij z vnaprej znano projekcijo, tako da postanejo vektorji gosti, s tem pa možne tudi vzporedne implementacije operacij nad njimi, kar je zagotovo možnost za prihodnje delo.

Čeprav izhod iz algoritma ni najbolj primeren za neposredno predstavitev uporabnikom, pa se mi zdi, da je rešitev predlagana v tej diplomski nalogi velik korak k zadanemu cilju.

Zahvala

Na tem mestu bi se rad zahvalil mentorju profesorju Ivanu Bratku; somentorici Dunji Mladenić za koristne nasvete pri pisanju; Janezu Branku in Marku Grobelniku za diskusije in obilico idej; vsem sodelavcem na IJS za zanimivo okolje; nenazadnje pa tudi staršema, ki sta me vedno podpirala in mi puščala obilico svobode.

Ljubljana, september 2008

Literatura

[1] DMOZ: open directory project. http://dmoz.org/.

[2] C. Aggarwal, J. Han, J. Wang, in P. Yu. A framework for clustering evolving data streams. V: *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, Berlin, Germany, 2003.

[3] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, in Philip S. Yu. A framework for projected clustering of high dimensional data streams. V: *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, Toronto, Canada, 2004.

[4] James Allan. *Topic Detection and Tracking: Event-Based Information Organization*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[5] Chris Buckley, James Allan, in Gerard Salton. Automatic routing and ad-hoc retrieval using SMART: TREC 2. V: *Proceedings of the Second Text REtrieval Conference (TREC-2). NIST Special Publication*, strani 45–56, 1994.

[6] Y. Dora Cai, David Clutter, Greg Pape, Jiawei Han, Michael Welge, in Loretta Auvil. Maids: mining alarming incidents from data streams. V: *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, strani 919–920, New York, NY, USA, 2004. ACM Press.

[7] Marko Grobelnik, Janez Brank, Dunja Mladenić, Blaž Novak, in Blaž Fortuna. Using DMoz for constructing ontology from data stream. V: Vesna Lužar Stiffler in Vesna Hljuz Dobrić, editors, *ITI 2006: Proceedings of the 28th International Conference on Information Technology Interfaces*, strani 439–444, Cavtat/Dubrovnik, Croatia, June 2006.

[8] Sudipto Guha, Nina Mishra, Rajeev Motwani, in Liadan O'Callaghan. Clustering data streams. V: *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, stran 359, Washington, DC, USA, 2000. IEEE Computer Society.

[9] Bjornar Larsen in Chinatsu Aone. Fast and effective text mining using linear-time document clustering. V: *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, strani 16–22, New York, NY, USA, 1999. ACM.

[10] David D. Lewis, Yiming Yang, Tony G. Rose, G. Dietterich, Fan Li, in Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[11] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. V: *Proceedings of 5–th Berkeley Symposium on Mathematical Statistics and Probability*, zvezek 1, strani 281–297. University of California Press, 1967.

[12] Christopher D. Manning, Prabhakar Raghavan, in Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, July 2008.

[13] Silvia Nittel in Kelvin T. Leung. Parallelizing clustering of geoscientific data sets using data streams. V: *SSDBM 2004*, stran 73. IEEE Computer Society, June 2004.

[14] M. F. Porter. An algorithm for suffix stripping. strani 313–316, 1997.

[15] G. Salton, A. Wong, in C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[16] M. Steinbach, G. Karypis, in V. Kumar. A comparison of document clustering techniques, 2000.

[17] Gerhard Widmer in Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Journal of Machine Learning*, 23(1):69–101, 1996.

[18] Tian Zhang, Raghu Ramakrishnan, in Miron Livny. BIRCH: an efficient data clustering method for very large databases. V: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, strani 103–114, 1996.

Izjava

Izjavljam, da sem diplomsko nalogo izdelal samostojno pod mentorstvom prof. dr. Ivana Bratka in doc. dr. Dunje Mladenić. Sodelavce, ki so mi pri delu pomagali, sem navedel v zahvali.

Blaž Novak