# Useaa tietolähdettä käyttävä klusterointi

Mikko Heinonen

Tiedon louhinnan seminaari, kevät 2008

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

| Tiedekunta/Osasto – Fakultet/Sektion – Faculty/Section | Laitos – Institution – Department | |
|---|---|---|
| Matemaattis-luonnontieteellinen tiedekunta | Tietojenkäsittelytieteen laitos | |
| Tekijä – Författare – Author | | |
| Mikko Heinonen | | |
| Työn nimi – Arbetets titel – Title | | |
| Useaa tietolähdettä käyttävä klusterointi | | |
| Oppiaine – Läroämne – Subject | | |
| Tietojenkäsittelytiede | | |
| Työn laji – Arbetets art – Level | Aika – Datum – Month and year | Sivumäärä – Sidoantal – Number of pages |
| Seminaarikirjoitelma | 17.3.2008 | 13 |

Tiivistelmä – Referat – Abstract

Tiedon klusteroinnille useasta tietolähteestä on kasvava tarve. Tässä seminnarissa esitellään CMS-algoritmi, joka klusteroi erillisistä tietolähteistä yhden osittelupuun. Algoritmi perustuu ohjaamattomaan oppimiseen. Siinä algoritmille ei tarvitse antaa mitään luokittelua, jonka perusteella tieto luokitellaan, vaan algoritmi löytää luokittelun itse annetuista tietolähteistä selvittämällä olennaiset jaottelun. Tietolähteissä pitää olla yhteinen nimittäjä, jonka avulla eri tietolähteissä ovat tietoalkiot voidaan tunnistaa samaksi tapaukseksi.
Ensimmäiseksi algoritmi luo osittelupuun jokaisesta tietolähteestä erikseen ja sen jälkeen osittelupuut sulautetaan yhdeksi puuksi. Algoritmissä on lisäksi kaksi vakio arvoa, joiden avulla voidaan parantaa algoritmin toimintaa epätäydellisen tiedon tapauksissa ja kohinaa sisältävän tiedon tapauksissa.
Aluksi on selvitettävä olennaiset attribuutit. Nämä voidaan selvittää algoritmista käyttämällä Bayesilaista luokittelua. Olennaisten attribuuttien selvittämisessä käytetään apuna vakio $\alpha$:a, jonka avulla estetään ettei epätäydellinen tietolähde mahdollista epäolennaisten attribuuttien vaikuttaa luokittelluun Seuraavaksi tietolähteet luokitellaan omiksi osittelupuiksi käyttäen olennaisimpia attribuutteja jaottelun alussa.
Osittelupuiden sulauttamisessa puut yhdistetään yhdeksi puuksi. Tässä vaiheessa on käytössä vakio $\beta$, jolla estetään kohinan vaikutusta lopputulokseen. Lopputuloksena on osittelupuu, jossa on tehty luokittelu kaikista tietolähteistä.

| |
|---|
| Avainsanat – Nyckelord – Keywords |
| Säilytyspaikka – Förvaringställe – Where deposited |
| Muita tietoja – Övriga uppgifter – Additional information |

# 1 Johdanto

Seminaari työ käsittelee tiedon klusterointia useista eri tietolähteistä tapauksissa, joissa luokkia ei ole annettu valmiina. Tieto on usein tallennettu moniin eri tietokantoihin tai tietolähteisiin, jotka voivat olla fyysisestikin eri paikoissa.

Useasti kun tietoa luokitellaan eri ryhmiin ja tutkitaan mikä ryhmä on ”suosittu”, niin luokittelut on tehty jonkun alan ammattilaisen toimesta perustuen esimerkiksi kokemukseen. Tällaista oppimista kutsutaan ohajtuksi oppimiseksi. Toisenlainen tilanne muodostuu, jos luokittelua ei tehdäkään etukäteen, vaan laitetaan kone tekemään luokitukset perustuen annetuun tietoon. Tällaista oppimista kutsutaan taas ohjaamattomaksi oppimiseksi.

Monesti yrityksissä on monenlaisia tietovarastoja talletettuna asiakkaista. Toisissa käsitellään esimerkiksi asiakkaan demografisia tietoja ja toisissa sitten taas asiakkaan ostokäyttäytymistä, palveluiden käyttöä tai lokeja internetin käytöstä. Monesti tietojen tutkiminen yhdistettynä on todella hankalaa. Toisaalta yhdistelmätiedosta saattaa löytää asioita, jotka voivat olla hyvinkin merkityksellisiä tulevaisuuden päätöksille.

Tilanteita joissa on tarvetta luokitella monista eri lähteistä tulevaa tietoa on lukuisia. Edellä mainittujen asiakastietojen lisäksi eri kohteita voivat olla esimerkiksi nettisivujen luokittelu, jossa toinen tiedonlähde on esimerkiksi sivun hakusanat ja toinen on sivun visuaaliset ominaisuudet. Myös erikoisemmat erilaisiin sensoreihin perustuvat tietojen yhdistämiset ovat tulleet esille. Esimerkiksi puheentunnistusta tietokoneella tehdään nykyään äänen kuuntelemisen lisäksi seuraamalla suun liikkeitä samanaikaisesti.

Tässä seminaarissa tarkastellaan useasta lähteestä haetun tiedon luokittelemista valvomattoman oppimisen puitteissa. Eri tietolähteissä oletetaan olevan yhteinen primääri avain, eli tietokenttä jonka avulla samat alkiot eri tietolähteissä voidaan yhdistää toisiinsa.

## 2 Algoritmin toiminta

Algoritmi CMS (Classification from Multiple Sources) toimii kahdessa osassa. Ensimmäisessä osassa jokaisen tietolähteen tieto ryhmitellään omiksi osituspuiksi olennaisten attribuuttien avulla. Luokkien tunnisteet ovat uniikkeja ja niiden määrä yritetään saada kohtuulliseksi. Toisessa vaiheessa osituspuut yhdistetään yhdeksi osituspuuksi selkeän lopputuloksen saamiseksi. Seuraavassa algoritmin läpikäynnissä on käytössä kaksi tietolähdettä A ja B, joilla on attribuutit

$A_1, A_2, ..., A_m$ ja $B_1, B_2, ...,B_n$.

Ohjatussa oppimisessa, jossa luokat annetaan etukäteen joihin tietoalkiot lajitellaan, on olemassa kaava, joka määrittelee mihin luokkaan kyseinen tietoalkio menee. A ja B tietolähteen tapauksessa on siis olemassa kaavat f ja g jotka määrittelevät tietoalkioiden luokitukset, niiden attribuuttien avulla.

$$luokka = f(A_1, A_2, ..., A_m) = g(B_1, B_2, \ldots, B_n)$$

Algoritmin tavoitteena on siis selvittää kaavat *f* ja *g*.

Luokitteluvaiheessa tietolähteistä tehdään osituspuut. Osituspuissa juureksi valitaan kaikista merkityksellisin attribuutti. Tieto jaetaan ensimmäiseksi yhden tietolähteen attribuuttien mukaan ja sen jälkeen jokainen alaryhmä jaetaan taas rekursiivisesti alipuihin.

Yhdistysvaiheessa tietolähteiden valintapuut yhdistetään yhdeksi puuksi. Tämä tehdään sulauttamalla luokkia yhteen tiettyjen periaatteiden mukaisesti. Lopputuloksena on yksi kokonainen valintapuu, jossa on kaikkien tietolähteiden attribuutit edustettuina ja tiedoissa eniten esiintyvät attribuutit ovat merkityksellisimmässä asemassa, eli lähellä osituuspuun juurta.

## 2.1 Olennaisten attribuuttien selvittäminen

Ensimmäisenä vaiheena on jaotella tieto ryhmiin siten, että jokaisella ryhmällä on uniikki tunnus. Riskinä tässä on, että tieto ryhmitellään liian hienoihin ryhmiin, eli pahimmassa tapauksessa jokainen ryhmä sisältäisi vain yhden tietoalkion. Tavoitteena olisi siis luoda isoja ryhmiä. Tämän saavuttamiseksi ryhmittely kannattaa tehdä käyttäen olennaisia attribuutteja ryhmien jakamiseen.

Oletetaan, että $p(X|Y)$ on posteriori todennäköisyys optimaalisesta Bayesilaisesta luokittelijasta, jossa $X$ on luokkamuuttuja ja $Y$ on attribuutit. Oletamme, että $P(X|Y)$ on keskiarvo todennäköisyys todennäköisimmän hypoteesin $X = x$ annettuna $Y = y$ käyttäen MAP (maximuma posteriori) periaatetta, jossa keskiarvo on otettu yli $Y$:n. Merkitään $A = \{A_1, A_2, ..., A_m\}$, $A'_i = A - A_i$ ja $C$ on luokan tunnus. Samanlainen merkintätapa on määritelty $B$:lle. Määrittelemme, että attribuutti $A_i$ on olennainen jos

$$p( C \mid A ) > p( C \mid A'_i ).$$

Luokan tunnus $C$ on kuitenkin tässä vaiheessa vielä tuntematon, joten yllä olevaa vertailua ei voida suoraan käyttää päättelemään attribuuttien olennaisuutta. $C$:n käyttö voidaan kuitenkin välttää käyttämällä $B$:n attribuutteja, koska tietoalkiot on esitettynä myös niiden avulla. Edellisestä vertailusta voidaan johtaa seuraava vertailu attribuutin $A_i$ olennaisuuden selvittämiseksi:

$$p( A_i \mid A'_i, B) > p( A_i ) \text{ tai } p( A_i \mid A'_i, B ) - p( A_i ) > 0$$

Jälkimmäistä vertailua kutsutaan attribuutin $A_i$ olennaisuuden kriteeriksi (Attribute Relevancy Criterion, ARC). Vertailun mukaan jos attribuutti $A_i$ on relevantti, niin sen ennustettava tarkkuus $A_i$:n todennäköisimmän arvon päättelemisessä käyttäen $A$:n muita attribuutteja ja $B$:n kaikkia attribuutteja pitäisi olla korkeampi kuin tarkkuus ennustettaessa $A_i$:n normaali arvoa (esimerkiksi useimmin esiintyvä $A_i$:n arvo).

Jos $A_i$:n arvo ei ole ennustettavissa $A´_i$:n ja $B$:n avulla se tarkoittaa, että ne ovat erilliset toisistaan ja niiden yhteiset tiedot ja ARC arvo ovat nolla. Tämä johtaisi siihen, että $A_i$:n käyttäminen kuvaamaan ryhmiä johtaisi isoon määrään ryhmiä, mikä ei ollut toivottava lopputulos. Joissain tapauksissa myös merkityksettömät attribuutit saattavat saada riitävän ison ARC arvon, jotta ne otetaisiin mukaan ryhmittelyä tehtäessä. Tämä saattaa tulla ilmi jos tietoalkioita on suhteellisen vähän kaikkiin mahdollisuuksiin verrattuna, eli kaikista mahdollisista attribuutti yhdistelmistä on vain osa edustettuna tietolähteessä. Jotta nämä epäolennaiset attribuutit voidaan eliminoida lopputuloksesta pois attribuutin olennaisuuden määrittämisen vertailu muutetaan muotoon

$$p\,(A_i \mid A´_i, B\,) - p\,(A_i\,) > \alpha$$

missä $\alpha$ on vakio, jonka arvo on välillä 0 ja 1. Tällä minimiarvolla voidaan estää, että sattumanvaraiset epäolennaiset attribuutit eivät pääse vaikuttamaan ryhmittelyyn lisäämällä ryhmien määrää tarpeettomasti.

## 2.2 Luokittelun tekeminen

Todennäköisyyden $p(A_i)$ arvioiminen on suhteellisen helppoa. Se on $A_i$:n yleisimmän arvon frekvenssi. $p(A_i \mid A´_i, B)$ selvittäminen onkin yleensä vähän hankalampaa. Siihen voidaan käyttää esimerkiksi C4.5 ohjelmaa 10-kertaista ristiinvalidointia (Qiunlan 1993).

Itse osittelupuu luodaan edellä mainittujen kahden arvon mukaan. Juureksi valitaan attribuutti $A_i$, joka antaa maksimiarvon kaavalla $p(A_i \mid A´_i, B) - p(A´_i)$. Tämän jälkeen loput alkiot jaetaan $A_i$:n arvojen mukaan alaryhmiin ja samanlainen osittelupuu luodaan jokaiselle alaryhmälle. Samaa algoritmi käytetään $B$:n alkioille. Alla pseudokoodi luokittelualgoritmille:

**luokittele(alkiot)**

For each $A_i$ Do

Calculate($A_i$)

Käytä C4.5-ohjelmaa todennäköisyyden p( $A_i$ | $A'_i$, B) arvioimiseksi

ARC($A_i$) = p( $A_i$ | $A'_i$, B) – p($A_i$)

If ei löydy $A_i$, jolla olisi ARC($A_i$) > a

Then

*Kaikki alkiot nimetään yhdeksi luokaksi*

Else

$A_j$ = argmax ARC($A_i$)

Jaa alkiot alaryhmiin $A_j$:n arvojen mukaisesti

Käytä luokittele algoritmiä jokaiseen alaryhmään

## 2.3 Luokkien yhteenliitäminen

Molemmista tietolähteistä on nyt muodostettu omat valintapuut. Seuraavaksi on tarkoitus yhdistää puut, jolloin lopputuloksena on yksi puu, jossa on kaikki merkitykselliset attribuutit edustettuna. Yhdistäminen lähtee siitä tilanteesta, että samat alkiot ovat edustettuina kaikissa valintapuissa, joten niiden yhdistäminen yhdeksi puuksi on mahdollista. Yhdistämisen perusajatuksena on, että jos kaksi eri alkiota on samassa ryhmässä yhdessä valintapuussa, niin niiden pitäisi olla samassa ryhmässä myös toisissa valintapuissa.

Esimerkiksi valintapuu A sisältää kaksi lehteä $L_{A1}$ ja $L_{A2}$ ja valintapuu B sisältää lehden $L_B$. Jos $a_1 \in L_{A1}$ ja $a_2 \in L_{A2}$ ja $a_1, a_2 \in L_B$, niin $L_{A1} \cap L_B \neq \emptyset$ ja $L_{A2} \cap L_B \neq \emptyset$. Joten lehdet $L_{A1}$ ja $L_B$ pitäisi nimetä samaksi välilehdeksi, kuten myös $L_{A2}$ ja $L_B$ pitäisi. Tästä seuraa, että $L_{A1}$ ja $L_{A2}$ voidaan sulautetaan yhteen. Tätä vertailua

tehdään molemmista valintapuista käsin ja sulautetaan yhteen kaikki tarvittavat lehdet.

Useissa isoissa tietolähteissä esiintyy yleensä kohinaa. Kohina voi helposti sotkea yhdistämisosion toimintoa. Vaikka $L_{A1}$ ja $L_B$ olisivatkin yhdistettävissä samaksi välilehdeksi, niin sitä ei kannata tehdä elleivät näytteet ole riittävän suuria, jotta yhdistäminen olisi järkevää. Kohinan eliminoimiseksi otetaan käyttöön vakio β, jonka arvo on 0:n ja 1:n välillä. Kohinaa poistetaan sulauttamalla $L_{Ai}$ ja $L_{Bj}$ vain jos

$$\frac{|L_{Ai} \cap L_{Bj}|}{\min(|L_{Ai}|, |L_{Bj}|)} > \beta$$

### 2.4 Algoritmin tehokkuus

Algoritmin aikavaativuus on $O(e^2)$. Algoritmin ensimmäisen vaiheen vaativuus on $O(ea)$, missä $e$ on tietoalkioiden määrä ja $a$ on attribuuttien kokonaismäärä kaikissa tietolähteissä. Yhdistämisvaiheessa jokainen yhdistäminen vähentää luokkien määrää jokaisessa tietolähteessä, joten läpimenoaika on maksimissaan $e$.

## 3 Testejä algoritmilla

CMS-algoritmilla tehdyissä kokeissa sekä keksityillä että oikeilla tietolähteillä algoritmi toimi kuten oli tarkoituskin. Algoritmia on ensimmäiseksi testattu keksityillä tietolähteillä, jolloin voidaan varmistaa, että algoritmi toimii oikein eri tilanteissa. Sen jälkeen algoritmia on testattu todellisilla tietolähteillä ja vertailtu toisiin lähes samaa asiaa tekeviin algoritmeihin.

### 3.1 Testaukset keksityillä tietolähteillä

Algoritmia testattiin käyttämällä kahta tehtyä tietolähdettä, joista tiedettiin, mikä on optimaalisin luokittelu tapa. Tietolähteet olivat taulun 3.1 mukaiset. Tietolähteiden oikeat kaavat f ja g ovat tässä tapauksessa:

$$f = T \text{ if } \begin{cases} (education = univ \vee high\ school) \\ (education = others) \wedge (age = young) \end{cases}$$

ja F muutoin

$$g = T \text{ if } assets = high$$

ja F muutoin

Testiaineistossa on yhteensä 2 304 tietoalkiota, eli 1 088 kumassakin tietolähteessä. Ehdon $f = g$ mukaan lopullisia eri vaihtoehtoja on siten 1 088 kappaletta. Niistä 448 kappaletta kuuluu T-ryhmään ja loput 640 F-ryhmään. Ideaali valintapuu tästä tapauksesta on esitetty kuvassa 3.2.

Nämä keksityt tietolähteet ovat täydellisiä ja ne eivät sisällä kohinaa, joten $\alpha$:n arvo asetetaan 4.0% ja $\beta$:n arvo asetetaan 0:ksi. Testiaineistoon käytettynä CMS algoritmi löytää juuri samanlaisen valintapuun, kuin ideaali tilanne on. Algoritmi siis toimi oikein kohinattomassa tilanteessa, jossa tieto on täydellisessä muodossa.

```
education = univ: T (192)
education = college: F (384)
education = high_school: T (192)
education = others
|          age = young: T(64)
|          age = middle: F (128)
|          age = old: F (128)
```

```
assets = low: F (320)
assets = medium: F (320)
assets = high: T (448)
```

Tilanteessa jossa tieto ei ole täydellistä, sattumanvaraiset poikkeukset saattavat muuttaa luokittelua huomattavastikin ja ryhmittelyn tekeminen tulee vaikeammaksi, koska merkitsevien attribuuttien löytäminen saattaa olla hankalampaa. Algoritmin toimintaa testattiin samalla aineistolla, mutta ottamalla mukaan vain 10-90% aineistosta sattumanvaraisesti. Testi tehtiin kolmeen kertaan ja tulokset ovat taulussa 3.3. Siitä huomataan, että tietoalkioiden vähentyessä ja siten tiedon ollessa epätäydellisempää lehtien määrä valintapuussa kasvaa huomattavasti, koska merkityksellisiä attribuutteja on vähemmän. Kuitenkin yhdistämisosuuden jälkeen valintapuu saadaan aina muotoon, jossa on kaksi lehteä (T ja F tässä tapauksessa). Jos tieto on hyvin epätäydellistä sitä voidaan korjata nostamalla $\alpha$:n arvoa ylöspäin, jolloin yksittäiset attribuutit eivät pysty vaikuttamaan jaotteluun niin paljoa, vaan ne attribuutit joita löytyy enemmän ovat vain niitä jotka vaikuttavat.

| Ajo | % tiedosta mukana | 90 % | 80 % | 70 % | 60 % | 50 % | 40 % | 30 % | 20 % | 10 % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | lehtiä *f*:ssä | 7 | 6 | 8 | 13 | 20 | 11 | 24 | 34 | 40 |
| | lehtiä *g*:ssä | 3 | 3 | 4 | 3 | 3 | 3 | 3 | 19 | 37 |
| | yhdistämisen jälkeen | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | lehtiä *f*:ssä | 6 | 7 | 8 | 6 | 9 | 16 | 19 | 34 | 35 |
| | lehtiä *g*:ssä | 3 | 4 | 3 | 4 | 3 | 4 | 17 | 24 | 42 |
| | yhdistämisen jälkeen | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | lehtiä *f*:ssä | 6 | 6 | 1 | 10 | 17 | 26 | 24 | 33 | 41 |
| | lehtiä *g*:ssä | 3 | 3 | 3 | 3 | 3 | 3 | 19 | 32 | 41 |
| | yhdistämisen jälkeen | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

*Taulu 3.3.* CMS algoritmin tulokset epätäydellisillä tietolähteillä.

Algoritmia testattiin myös tietolähteissä, joissa esiintyy kohinaa. Näissä tapauksissa ehto f = g ei ole voimassa. Tietolähteinä käytettiin samoja lähteitä kuin edellä, jossa tiedon täydellisyys oli 90%. Tietoja muokattiin siten, että niissä esiintyy

5%, 10%, 15% ja 20% kohinaa. Sitä kompensoitiin asettamalla b:n arvo 20% ja 40% välille. Testien tulokset ovat taulussa 3.4.

| | **Kohinan osuus** | **5 %** | **10 %** | **15 %** | **20 %** |
|---|---|---|---|---|---|
| β **= 20 %** | Lehtiä f:ssä | 8 | 4 | 4 | 4 |
| | Lehtiä g:ssä | 4 | 3 | 3 | 3 |
| | yhdistämisen jälkeen | 2 | 1 | 1 | 1 |
| β **= 30 %** | Lehtiä f:ssä | 8 | 4 | 4 | 4 |
| | Lehtiä g:ssä | 4 | 3 | 3 | 3 |
| | yhdistämisen jälkeen | 2 | 2 | 2 | 1 |
| β **= 40 %** | Lehtiä f:ssä | 8 | 4 | 4 | 4 |
| | Lehtiä g:ssä | 4 | 3 | 3 | 3 |
| | yhdistämisen jälkeen | 2 | 2 | 2 | 2 |

*Taulu 3.4.* Algoritmin testaukset kohinaa sisältävillä tietolähteillä.

## 3.2 Testaukset todellisilla tietolähteillä

Todellisilla tietolähteillä algoritmiä testattiin käyttämällä äänestys tuloksia. Tietolähde sisältää 16 attribuuttia, joilla jokaisella on 3 mahdollista arvoa. Äänestystuloksia on mukana vain 435 kappaletta (näistä 168 on republikaaneja ja 267 on demokraatteja). Tietolähde jaettiin kahteen osaan, jonka jälkeen algoritmia testattiin käyttämällä kahta kyseistä tietolähdettä. Algoritmin tuottama lopputulos on esitetty taulussa 3.5. Siinä on alkuperäinen jaottelu riveillä ja CMS-algoritmin tuottama jaottelu sarakkeissa.

| | $C_1$ | $C_2$ |
|---|---|---|
| | (180) | (255) |
| Republikaanit (168) | 163 | 5 |
| Demokraatit (267) | 17 | 255 |

*Taulu 3.5.* Algoritmin testaus vaalitulos tiedolla.

Algoritmi ei pääse ihan samaan tulokseen kuin alkuperäinen jaottelu. Algortimin jaottelussa 180 kappaletta $C_1$ luokkaan ja 255 kappaletta $C_2$ luokkaan. Tämä saattaa johtua kohinasta tai sitten jopa siitä, että muutama republikaani on valintojensa puolesta ollutkin demokraatti tai toiste päin.

# 4 Yhteenveto

CMS-algoritmi toimii hyvin usean tietolähteen klusteroinnissa. Keksityistä tietolähteistä tehdyt luokittelut menivät kohdalleen, niin kuin oli tarkoituskin. Epätäydelliset tietolähteet eivät aiheuttaneet pahempia virheitä tiedonluokitteluun, kunhan $\alpha$:n arvoa säädettiin tarpeen mukaan. Samoin kohinaa sisältävissä tietolähteissä luokittelu meni hyvin kohdalleen. Tietenkin jos tietolähteet ovat todella epätäydellisiä ja sisältävät paljon kohinaa, ei luokittelua pystytä välttämättä tekemään mitenkään järkevästi ohjaamatonta oppimista käyttämällä. Todellisten tietolähteiden testauksesta huomattiin, että luokittelu kyllä onnistuu, mutta todellisten tietolähteiden kohina saattaa yleensä olla todella suurtakin, joka puolestaan sekoittaa luokittelua jonkin verran.