# Aikasarjojen segmentointi
## Seminaariraportti

**Jaana Heino**

**Data Mining-seminaari**
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Helsinki 6. helmikuuta 2002

# Sisältö

# 1 Johdanto

Aikasarja on sarja mittausarvoja eri ajanhetkillä. Esimerkiksi tietyn yhtiön pörssikurssi, lämpötila tietyssä paikassa tai tietyn taudin esiintyvyys väestössä muodostavat aikasarjan. Aikasarjojen analysoinnissa pyritään esimerkiksi kuvailemaan tietyn aikasarajan käyttäytymistä tai etsimään riippuvuuksia eri aikasarjojen väliltä.

Aikasarjan tarkastelemiseksi on se ensin esitettävä sopivassa muodossa. Mahdollisia esitysmuotoja aikasarjalle ovat esimerkiksi Fourier-muunnokset, aallokkeet (wavelets) ja paloittain lineaarinen regressio.

Tässä raportissa käsitellään aikasarjan muuntamista paloittain lineaariseksi esitykseksi eli segmentoimista. Raportti perustuu artikkeliin

Keogh, E., Chu., S, Hart, D. & Pazzani, M.,

An Online Algorithm for Segmenting Time Series,

*Proceedings of IEEE International Conference on Data Mining* s. 289-296.

Raportissa kerrotaan ensin lyhyesti mitä paloittain lineaarinen regressio on. Sen jälkeen esitellään kolme peruslähestymistapaa ja mainitussa artikkelissa esitetty uusi algoritmi. Lopuksi esitetään lyhyesti Keogh et al:n tulokset näiden algoritmien vertailusta.

# 2 Paloittain lineaarinen esitys

Kaksiulotteisessa lineaariregressiossa pyritään etsimään suoraa, joka kuvaa datapistejoukon x- ja y-koordinaattien riippuvuutta toisistaan.

Yleisesti käytössä oleva regressiosuoran etsimismenetelmä on pienimmän neliösumman menetelmä. Regressiosuoran neliösumma lasketaan mittaamalla kunkin datapisteen y-akselin suuntainen etäisyys suorasta ja laskemalla näiden etäisyyksien neliöt yhteen. Paras regressiosuora on se suora, joka minimoi tämän neliösumman. Tunnetusti pienimmän neliösumman regressiosuora on laskettavissa datasta lineaarisessa ajassa.

Paloittain lineaarisessa regressiossa on esitettävä $n$:n kokoinen datajoukko (esim. $n$:n arvon pituinen aikasarja) korkeintaan $k$:n peräkkäisen suoran joukkona, missä $k$ tyypillisesti on huomattavasti pienempi kuin $n$. Yhtä tällaista suoraa sanotaan segmentiksi, ja aikasarjan tapauksessa kaikkien segmenttien joukkoa ko. aikasarjan segmentoinniksi.

Paloittain lineaarisen regression ongelma voidaan muotoilla useilla tavoilla, esimerkiksi:

- annettuna aikasarja $T$ on muodostettava korkeintaan $k$ segmettiä sisältävä segmentointi

- muodostettava segmentointi jonka segmenttien yhteenlaskettu virhe on korkeintaan annettu $total\_max\_error$
- muodostettava segmentointi jossa yhdenkään segmentin virhe ei ole suurempi kuin tietty $max\_error$.

On syytä huomata, että kaikki seuraavassa esitettävät algoritmit eivät tue kaikkia näitä tavoitteita.

# 3 Kolme perusalgoritmia

## 3.1 Liukuva ikkuna ("sliding window")

Liukuva ikkuna ("sliding window") on kolmesta peruslähestymistavasta ainoa, joka pystyy ns. online-segmentointiin. Online-toiminta tarkoittaa sitä, että algoritmi käsittelee dataa ja muodostaa segmenttejä sitä mukaa kun aikasarja etenee sen sijaan, että odottaisi että koko aikasarja on kerätty ennen segmentoinnin aloittamista.

Liukuva ikkuna -algoritmi saa syötteenä käyttäjän spesifioiman suurimman yksittäiselle segmentille sallitun virheen $max\_error$ sekä lähteen, josta se voi lukea seuraavan aikasarjan arvon. Aloittaen aikapisteestä 0 algoritmi lukee sarjasta seuraavia datapisteitä kunnes jollain hetkellä $i$ se ei enää onnistu luomaan regressiosuoraa jonka virhe olisi pienempi kuin $max\_error$. Kun näin käy, muodostaa algoritmi parhaasta regressiosuorasta välillä $[0, i-1]$ aikasarjan segmentin ja aloittaa sovittamisen uudestaan arvosta $i$.

Liukuvan ikkunan menetelmä on jo sinällään yksinkertaisuudessaan melko nopea, mutta sen toimintaa voidaan monissa tapauksissa edelleen nopeuttaa esimerkiksi lukemalla kerralla yhden seuraavan arvon sijaan useita. Koska parhaan regressiosuoran neliösumma ei koskaan pienene kun aproksimoitavaan sarjaan lisätään arvoja, voidaan täten edetä sarjassa pitkiäkin hyppäyksiä kerrallaan ja olla aina varmoja siitä, että jos $max\_error$ ei ylity hyppäyksen jälkeen, se ei ole voinut ylittyä missään aikaisemmassakaan vaiheessa. Löydettäessä hyppäyksen jälkeen raja-arvon ylittävä neliösumma voidaan se joko hyväksyä sellaisenaan tai palata sarjassa taaksepäin kunnes taitekohta löytyy.

Liukuvan ikkunan menetelmän etuna on sen yksinkertaisuus ja online-luonne. Se ei kuitenkaan pysty ottamaan huomioon sarjan historiaa, ja niinpä saattaa pahimmissa tapauksissa tuottaa segmentointeja, joiden kokonaisvirhe on huomattavan suuri vaikkei minkään yksittäisen segmentin virhe ylitäkään raja-arvoa. Liukuva ikkuna toimii parhaiten kohinaisella datalla, jossa kohina synnyttää virhettä jo ennen ”oikeaa” käännekohtaa sarjassa. Jos taas sarja jo valmiiksi muodostuu selkeistä suorista, kestää taitekohdan jälkeen jonkin aikaa ennen kuin algoritmi ”huomaa” taitteen tapahtuneen.

Yksinkertainen liukuvan ikkunan menetelmä pseudokoodina ilmaistuna on seuraava:

```
algorithm SlidingWindow(T, max_error, SegT) {
  anchor = 0;
  while (data left in T) {
    i = 1;
      while (GetError(T[anchor, anchor+i]) < max_error) {
      i = i+1;
    }
    SegT = concat(SegT, CreateSegment(T[anchor, anchor+(i-1)]);
    anchor = anchor + i;
  }
}
```

Tässä siis GetError on funktio, joka palauttaa syötteenään saamastaan arvojoukosta muodostetun segmentin virheen ja CreateSegment funktio, joka palauttaa syötteenään saamastaan arvojoukosta muodostetun segmentin. $T[a, b]$:llä on merkitty aikasarjan jaksoa $T$ jaksoa $[a, b]$. $SegT$ on muodostettava segmentointi.

## 3.2 Kokoava menetelmä (”bottom-up”)

Kokoava (engl. ”bottom-up”) algoritmi aloittaa muodostamalla hienojakoisimman mahdollisen paloittaisen lineaariesityksen segmentoitavasta aikasarjasta. Tällaisessa segmentoinnissa on $n/2$ segmettiä, jos aikasarjan pituus on $n$; jokainen alkusegmentti on muodostettu kahden vierekkäisen pisteen perusteella ja niinpä kaikkien segmenttien virhe on tasan nolla.

Alkusegmenteistä etsitään se vierekkäisten segmenttien pari, jonka yhdistäminen yhdeksi segmentiksi lisäisi kokonaisvirhettä vähiten, ja tämä pari yhdistetään. Näin jatketaan, kunnes valittu lopetusehto täyttyy (segmenttejä on määrätty $k$ kappaletta, ei voida löytää uutta segmenttiä jonka virhe olisi alle $max\_error$, tai segmenttiä jonka muodostamisen jälkeen segmentoinnin kokonaisvirhe olisi alle $total\_max\_error$).

Algoritmi pitää kirjaa kaikkien olemassaolevien vierekkäisten segmenttien yhdistämisen aiheuttamasta lisävirheestä ja päivittää sitä vain yhdistettyjen segmenttien osalta.

Seuraavassa on esitetty pseudokoodina yksinkertainen kokoava algoritmi, käyttäen lopettamiskriteerinä yksittäisten segmenttien virhettä.

```
algorithm BottomUp(T, max_error) returns a segmentation of T {

  // luodaan ensimmäinen segmentointi
  for(i = 0; i < lenght(T); i = i + 2) {
    SegT = concat(SegT, CreateSegment(T[i, i+1])
  }

  // lasketaan yhdistämiskustannukset
  for(i = 0; i < length(SegT)-1; i = i +1) {
    mergeCost(i) = GetError(MergeSegments([SegT(i), SegT(i+1)));
  }
```

```
  while(min(mergeCost) < max_error) {
    // etsitään oikea yhdistämiskohta ja yhdistetään segmentit
    i = find_min(mergeCost);
    SegT(i) = MergeSegments(SegT(i), SegT(i+1));

    // poistetaan vanhat tiedot listoista
    delete(SegT(i)); delete(mergeCost(i));
    // ja lisätään uudet
    mergeCost(i) = GetError(MergeSegments(SegT(i), SegT(i+1));
    mergeCost(i-1) = GetError(MergeSegments(SegT(i-1), SegT(i));
  }

  return SegT;
}
```

## 3.3 Osittava menetelmä (”top-down”)

Osittava (engl. ”top-down”) mentelmä toimii päinvastaisesti kuin kokoava menetelmä. Se etsii ensin pisteen, josta aikasarja katkaisemalla saadaan aikaan kahden segmentin segmentointi, jonka virhe on pienin. (Mittana voidaan käyttää joko yhteenlaskettua virhettä tai etsiä segmentointia, jossa suurin yksittäisen segmentin virhe on mahdollisimman pieni).

Tämän jälkeen segmentointia toistetaan rekursiivisesti muodostuneille segmenteille, kunnes lopetusehto täyttyy (segmenttejä on haluttu määrä, kokonaisvirhe on painunut alle raja-arvon, tai kaikkien yksittäisten segmenttien virhe on hyväksyttävän rajan alapuolella).

Seuraavassa on esitetty pseudokoodina yksinkertainen osittava algoritmi, jossa lopettamiskriteerinä on yksittäisten segmenttien virhe.

```
algorithm TopDown(T, max_error) {
  // etsitään oikea sopiva katkaisukohta

  best_so_far = inf;
  for(i = 1; i < length(SegT)-1; i = i + 1) {
    error_here = GetError(T[0,i]) + GetError(T[i+1,length(SegT)];
    if(error_here < best_so_far) {
      breakpoint = i;
      best_so_far = error_here;
    }
  }

  // jos tarpeen, pilkotaan rekursiivisesti vasen puoli
  if(GetError(T[0, breakpoint]) > max_error) {
    SegTLeft = TopDown(T[0, breakpoint], max_error);
  } else {
    SegTLeft = CreateSegment(T[0, breakpoint]);
  }

  // samoin oikea
  if(GetError(T[breakpoint+1, length(T)]) > max_error) {
    SegTRight = TopDown(T[breakpoint+1, length(T)], max_error);
  } else {
    SegTRight = CreateSegment(T[breakpoint+1, length(T), max_error);
  }

  // yhdistetään tulokset ja palautetaan segmentointi
  return concat(SegTLeft, SegTRight);
}
```

# 4 Sliding Window and Bottom-up

Liukuvan ikkunan menetelmän ongelma on sen huono suorituskyky tietynlaisilla sarjoilla. Toisaalta mikään muu perusalgoritmi ei tue online-segmentointia. Keoghin et al:n artikkelissaan esittämä uusi SWAB-algoritmi yhdistää nimensä mukaisesti liukuvan ikkunan ja kokoavan ("sliding window and bottom up") menetelmän toiminnan tehokkaammaksi online-algoritmiksi.

SWAB-algoritmi pitää yllä bufferia, jonka koko aluksi on sellainen, että siihen mahtuu noin 5-6 segmentin verran dataa. Tämä datajoukko segmentoidaan kokoavalla algoritmilla. Ajallisesti vanhin suora raportoidaan aikasarjan segmentiksi ja siihen kuuluvat datapisteet poistetaan bufferista.

Sen jälkeen luetaan sarjasta uusia datapisteitä bufferiin niin kauan kuin luetuista uusista pisteistä muodostetun segmentin virhe ei ylitä tiettyä etukäteen määriteltyä virhettä. Kun pisteiden luku loppuu, sovelletaan taas bufferissa olevaan dataan kokoavaa algoritmia ja vanhin segmentti raportoidaan. Näin jatketaan niin kauan kuin dataa riittää (periaatteessa ikuisesti).

Algoritmissa on asetettava bufferin koolle tietyt rajat. Näin siksi, että jos bufferin koko kasvaa suureksi, on algoritmi käytännössä puhdas kokoava algoritmi eikä siis enää online-käyttöön sopiva. Jos taas bufferin koko pienenee lähestyy algoritmin toiminta liukuvan ikkunan menetelmää kaikkine sen ongelmineen. Keogh et al asettavat bufferin koolle rajoiksi puolikkaan ja kaksinkertaisen alkuperäisestä bufferista. Bufferin rajoissa pysyminen tarkistetaan uusien datapisteiden luvun yhteydessä; tarvittaessa luetaan lisää pisteitä tai keskeytetään lukeminen kesken.

Hieman yllättäen Keoghin ja al:n empiirisissä testeissä SWAB saavuttaa saman tuloksen kuin varsinainen bottom-up, vaikka se pitää muistissaan vain 5-6 aikaisempaa segmenttiä koko aikasarjan historian sijaan.

# 5 Algoritmien vertailu

## 5.1 Koejärjestely

Vertaillakseen segmentointialgoritmeja Keogh et al tutkivat niiden suorituskykyä kymmenellä eri aineistolla. Aineistot oli pyritty valitsemaan siten, että ne edustaisivat ääripäitä kohinan, syklisyyden, stationäärisyyden ja symmetrisyyden tai näiden puutteen suhteen. Lisäksi testiaineistot pyrittiin valitsemaan todellisten aineistojen joukosta; mukana on taloustieteellisiä, lääketieteellisiä ja luonnontieteellisiä aikasarjoja.

Kaikista algoritmeista testattiin versiota, jossa tavoitteena on muodostaa segmentointi, jossa kunkin yksittäisen segmentin virhe on korkeintaan $max\_error$. Sekä suurilla että pienillä $max_error$:n arvoilla kaikkien algoritmien luoma segmentointi on sama - suurilla arvoilla yksittäinen regressiosuora, pienillä arvoilla hienoin mahdollinen jako eli $n/2$ suoraa sisältävä segmentointi.

Tämän vuoksi oli tarpeen valita $max\_error$:n arvo siten, että se on ”järkevä”. Keogh et al määrittelivät ensin ”järkevän” kullekin datasetille erikseen. Tämän arvon ympärille he asettivat kuusi arvoa jotka erosivat toisistaan kahden potenssin kertoimilla; toisin sanoen, kutakin algoritmia testattiin paitsi subjektiivisesti valituilla ”järkevillä” arvoilla myös arvoilla, joiden arveltiin parhaassa tilanteessa tuottavan liian suuren tai liian pienen virheen.

Kukin algoritmi ajettiin jokaista kymmentä aineistoa kohden jokaisella kuudella $max\_error$:n arvolla ja saadun segmentoinnin kokonaisvirhe laskettiin.

## 5.2 Koetulokset

Kokeissa havaittiin, että liukuva ikkuna -algoritmin suoriutuminen oli ennakoidusti heikoin. Osittava algoritmi suoriutui joissain tapauksissa marginaalisesti paremmin kuin kokoava, mutta yleensä nämä algoritmit olivat tasavertaiset, ja kokoavan ollessa parempi ero oli selvempi. Yllättäen kokoavan menetelmän ja SWAB-algoritmin suoriutumisessa ei ollut merkittävää eroa.

# 6 Yhteenveto

Tässä raportissa esiteltiin neljä algoritmia aikasarjan segmentoimiseksi. Aikasarjan segmentoinnilla tarkoitetaan sen kuvaamista perättäisillä regressiosuorilla. Segmentointialgoritmien kolme peruslähestymistapaa ovat sliding window, bottom-up ja top-down. Lisäksi esitettiin kaksi ensimmäistä peruslähestymistapaa yhdistävä SWAB-algoritmi ja ko. algoritmin kehittäjien (Keogh et al) suorittama empiirinen algoritmien vertailu.

Vertailutulosten mukaan parhaiten suorituivat algoritmit bottom-up ja SWAB, eikä niiden välillä ollut merkittävää eroa. Tämä on tärkeä havainto, koska SWAB on online-algoritmi, mutta pystyy siitä huolimatta yhtä hyviin tuloksiin kuin bottom-up, joka ei pysty online-suoritukseen.