

582216 Johdatus tekoälyyn

Syksy 2012
T. Roos

KURSSIKOODI: 582216
OPINTOPISTEET: 4.0
ERIKOISTUMISLINJA:
Algoritmit ja koneoppiminen
TASO: Aineopinnot
KUVAUS:
Kurssilla käydään läpi
tekoälyn ongelma-alueita ja
niihin liittyviä
ratkaisumenetelmiä koostuen
luennoista,
ohjelmointitehtävistä,
harjoitustehtävistä, sekä
kurssikokeesta.
KURSSIKOE: 19.10.2012
klo 9-12, A111

YLEISTÄ

Kurssin päätavoitteena on saada käsitys tekoälyn perusongelmista, -sovelluksista ja -menetelmistä, sekä tekoälyn tärkeimmistä kehitysaskeleista sen historian kuluessa. Painotus on nykytutkimuksella ja käytännön sovelluksilla. Erityinen tavoite on luoda yhteyksiä teorian ja käytännön välillä toteuttamalla luennoilla opettettujen periaatteiden mukaisesti oikeasti hyödyllisiä työkaluja.

1. Mitä on tekoäly?	2
2. Etsintä ongelmanratkaisuna	3
3. Pelit	6
4. Logiikka tekoälyssä	9
5. Päättely epävarmuuden vallitessa: Todennäköisyydestä	11
6. Päättely epävarmuuden vallitessa: Bayes-verkot	15
7. Päättely epävarmuuden vallitessa: Roskapostisuodatin	21
8. Koneoppiminen	25
9. Neuroverkot	28
10. Digitaalinen signaalinkäsittely	33
11. Robotiikka	35

I. Mitä on tekoäly?

Luennon alussa käytiin läpi kurssin logistiikkaa ja suorittamiseen liittyviä asioita. Lisäksi kuvailtiin tekoälyä tutkimusalueena. Jaottelu **GOFAI** (“Good Old-Fashioned AI”) vs “**moderni AI**” esitettiin jo tässä vaiheessa. Siihen palataan vielä kurssin kuluessa.

Tekoäly esiintyy usein **kulttuurissa** (elokuvat, kirjat, pelit, ...), jossa on tapana painottaa kauhuskenaarioita.

Tekoälyn tavoitteet voidaan jaotella yhtäältä akselilla **älykäs–ihmismäinen** ja toisaalta akselilla **ajattelee–toimii**. “**Vahva tekoäly**” viittaa älykkäästi ajattelevaan ja itsestään tietoiseen tekoälyyn. “**Heikolla tekoälyllä**” tarkoitetaan älykkäästi toimivaa ohjelmaa tai agenttia. Ihmismäisesti toimiva kone läpäisee **Turingin kokeen**.



Kuva 1. Tekoälyn tavoitteisiin liittyvät akselit.

Kiinalaisen huoneen argumentti liittyy kysymykseen, edellyttääkö älykäs toiminta ajattelua. Onnistuneet tekoälysovellukset näyttäisivät viittaavan siihen, että ei edellytä.

Kulttuuristen viitteiden ja filosofisen pohdinnan jälkeen siirryttiin ripeästi eteenpäin ja kysyttiin “Mitä tekoäly oikeasti on?” Sovellusten (itsestään ajava auto, Jeopardy-pelin voittava Watson-tietokone, Google, Amazonin suositelualgoritmi) ja konferenssien ohjelmiston perusteella **tekoälytutkimus** ja sen sovellukset keskittyvät tyypillisesti pienten yksittäisten ongelmien, kuten konenäön, reitinoiminnin, konekäännöksen, suosittelun, tiedonhaun, jne. ratkaisuun. Suurten kysymysten (“Mitä on tietoisuus?”) pohdinnasta on enimmäkseen luovuttu hedelmättömänä.

Tähän aiheeseen liittyviä oppimistavoitteita (Huom: aihetta käsitellään myös myöhemmin kurssilla muiden aiheiden rinnalla):

Esitiedot

- tuntee tekoälyyn liittyvää problematiikkaa kulttuurissa (elokuvat, kirjat, pelit, ...)

Lähestyy oppimistavoitetta

- osaa kuvailla Turingin kokeen
- hahmottaa tekoälyn nykytilan ja scifin välisen eron

Saavuttaa oppimistavoitteet

- osaa vertailla GOFAI- ja modernin tekoälyn menetelmiä
- osaa mainita tärkeimpiä tekoälytutkimuksen suuntauksia (sekä historiallisia että nykyisiä)

Syventää oppimistavoitteita

- tuntee nykyisen tekoälytutkimuksen kenttää (lehdet ja konferenssit) sekä sen sisäisiä jaotteluja

2. Etsintä ongelmanratkaisuna

Aluksi on syytä palauttaa mieleen TiRa-kurssilta tutut [leveys- ja syvyysuuntainen haku](#), joiden erona on pelkästään solmujen tallettamiseen käytettävä tietorakenne ([jono vs pino](#)).

Alla olevasta algoritmirungosta saadaan aikaan joko leveys- tai syvyysuuntainen etsintä riippuen LISÄÄ-funktion toteutuksesta.

ETSINTÄ(Alkusolmu)	% yleinen etsintäalgoritmi
<pre>Solmulista = [Alkusolmu] Käsitellyt = [] while Solmulista not empty Solmu = EKA(Solmulista) Solmulista = LOPUT(Solmulista) if Solmu not in Käsitellyt Käsitellyt = Käsitellyt + [Solmu] if MAALI(Solmu) return("ratkaisu", Solmu) Solmulista = LISÄÄ(NAAPURIT(Solmu), Solmulista) end if end while return("ei ratkaisua")</pre>	

Algoritmi 1. Yleinen etsintäalgoritmin runko. Muokattu 14.9.2012



Huom: Hakualgoritmeista on olemassa monta hieman toisistaan eroavaa versiota, joten älä hämäänny, jos tässä esitettävät versiot eivät ole samanlaisia kuin ne joihin olet aiemmin törmännyt. Esimerkiksi syvyysuuntainen haku esitetään

usein näppärästi rekursiivisena algoritmina. Solmujen läpikäyntijärjestys on joka tapauksessa aina samanlainen (poislukien saman solmun naapurien järjestyksen vaihtelusta johtuvat erot).

LISÄÄ(Naapurilista, Solmulista) % jono

Uudet = Naapurilista – Käsitellyt – Solmulista

return(PERÄKKÄIN(Solmulista, Uudet))

Algoritmi 2. LISÄÄ-funktion leveyssuuntaisen etsinnän tuottava versio. Muokattu 14.9.2012

LISÄÄ(Naapurilista, Solmulista) % pino

Uudet = Naapurilista – Käsitellyt

return(PERÄKKÄIN(Uudet, Solmulista))

Algoritmi 3. LISÄÄ-funktion syvyysuuntaisen etsinnän tuottava versio. Muokattu 14.9.2012

Esimerkki (ks. luentokalvot): Jos solmujonossa on alkio D ja sinne lisätään naapurit A ja F, joista F on uusi ja A on vanha, on tulos jonon (Algoritmi 2) tapauksessa [D,F] ja pinon tapauksessa [F,D]. Solmujonosta poimitaan seuraavaksi läpikäytäväksi solmuksi aina ensimmäinen alkio – siis jonon tapauksessa sinne ensin lisätty alkio ja pinon tapauksessa sinne viimeksi lisätty alkio.

Kummassakin LISÄÄ-funktion versiossa ensimmäisellä rivillä poistetaan listasta sellaiset solmut, jotka on jo käsitelty (Käsitellyt -lista). Näin vältetään [syklit](#) eli etsinnän palaaminen takaisin jo aiemmin käsiteltyihin solmuihin. Muistin säästämiseksi leveyssuuntaisessa haussa lisäämättä jätetään myös solmut, jotka jo ovat solmulistassa. (Käytännössä solmujen tilaa (käsitelty/solmulistassa/uusi) on kätevin pitää yllä kussakin solmussa, jotta tarkistus voidaan tehdä nopeasti.)

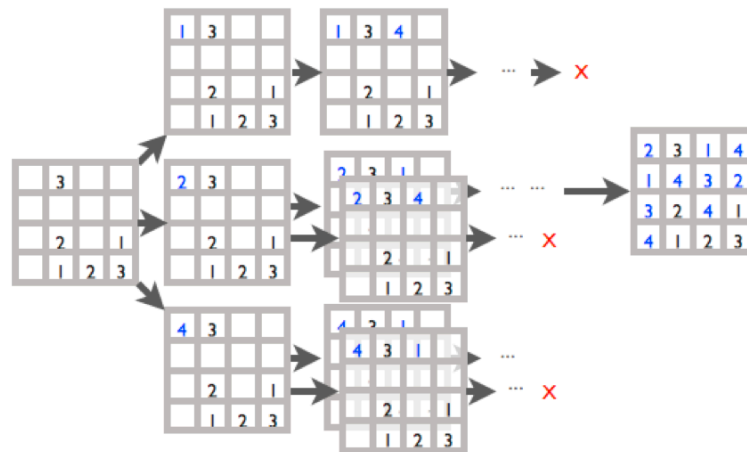


Huomaa, että jos etsintä tapahtuu [puussa](#), ei sykleistä tarvitse välittää.

NAAPURIT-funktio voidaan nimittäin määritellä siten, että se palauttaa argumenttina annetun solmun lapset. Käsitellyt -listaa ei tällöin tarvita, koska etsintä etenee puussa aina kohti lapsia, pois päin juuresta.

Hakualgoritmeja voi käyttää [ongelmanratkaisuun](#), kunhan ongelman saa muotoiltua sopivaan muotoon. Lähetysaarnajien ja kannibaalien joenylitystä voi kuvata tilakaaviona, jossa sallitut [tilat](#) ja niiden väliset [siirtymät](#) määräävät hakuavaruuden. Ongelma voidaan ratkaista etsimällä (miellellään lyhyt) reitti alkutilan ja lopputilan välillä.

Toisena esimerkkinä etsintäalgoritmien käytöstä ongelmanratkaisussa on sudokualgoritmi, joka vastaa syvyysuuntaista hakua, kun tiloja ovat kelvolliset osittaiset ratkaisut ja siirtymät vastaavat yhden numeron lisäämistä.



Kuva 2. Sudoku-etsintäavaruus puurakenteena.

Kun hakuavaruus on suuri, tulee ongelmaksi helposti se, että etsintä kestää liian kauan. Tällöin on hyödyllistä, jos voidaan käyttää [heuristista hakua](#), jossa jonon tai pinon asemesta solmut talletetaan ns. [prioriteettijonoon](#), josta ne poimitaan [paras-ensin-järjestyksessä](#): kustannusarvioltaan paras talletettu solmu poimitaan aina ensimmäisenä.

LISÄÄ(Naapurilista, Solmulista) % prioriteettijono

return(JÄRJESTÄ(Naapurilista, Solmulista))

Algoritmi 4. LISÄÄ-funktion paras-ensin-etsinnän tuottava versio.

Paras-ensin-versiossa solmulistassa olevat ja sinne lisättävät alkiot järjestetään lisäysvaiheessa niiden [kustannusarvion](#) mukaisesti. Huomaa, että toisin kuin leveys- ja syvyysuuntaisessa haussa tällä kertaa ei lisätä vain käsittelemättömiä solmuja, koska joku solmulistassa oleva solmu saatetaan lisätä listaan uudelleen paremmalla kustannusarviolla. (Tässä tapauksessa ko. solmun kustannusarvio päivitetään ja se siirtyy mahdollisesti jonossa lähemmäksi kärkeä.)

Jos kustannusarviona käytetään [polkukustannuksen](#) (nykyiseen solmuun päättyvän polun kustannus) ja ns. [heuristiikan](#) summaa

$$(I) \quad f(N) = g(N) + h(N),$$

saadaan erittäin käyttökelpoinen menetelmä eli [A*-haku](#). Jos heuristiikka ei koskaan yliarvioi nykyisestä solmusta maaliin johtavan polun pituutta, löytää A* aina [optimaalisen](#) reitin. Yleensä A* löytää optimaalisen reitin vieläpä melko tehokkaasti eli käymättä läpi valtavaa määrää eri reittejä.

Lisämateriaalia:

- Tietorakenteet ja algoritmit -kurssin materiaali.
<http://www.cs.helsinki.fi/u/floreen/tira2012/tira.pdf>
- W. Ertel, *Introduction to Algorithms*, Springer 2011, kappale 6: "Search, Games and Problem Solving" (kurssikansiossa)

MAX-ARVO(Solmu)	% Max-pelaajan funktio
-----------------	------------------------

```
if LOPPUTILA(Solmu) return(ARVO(Solmu))
v =  $-\infty$ 
for each Lapsi in LAPSET(Solmu)
    v = MAX(v, MIN-ARVO(Lapsi))
return(v)
```

Algoritmi 4. Minimax-algoritmi: Max-pelaajaa vastaava funktio.

MIN-ARVO(Solmu)	% Min-pelaajan funktio
-----------------	------------------------

```
if LOPPUTILA(Solmu) return(ARVO(Solmu))
v =  $+\infty$ 
for each Lapsi in LAPSET(Solmu)
    v = MIN(v, MAX-ARVO(Lapsi))
return(v)
```

Algoritmi 5. Minimax-algoritmi: Min-pelaajaa vastaava funktio.

Minimax-algoritmi ratkaisee periaatteessa optimaalisen tuloksen ja siihen johtavat pelisiirrot kaikissa kahden pelaajan peleissä, joihin ei sisälly satunnaisuutta. Siten esimerkiksi shakkipelissä on teoriassa olemassa optimaalinen pelistrategia kummallekin pelaajalle, jolla musta tai valkea voittaa tai päädytään tasapeliin (kukaan ei ainakaan toistaiseksi tiedä mikä vaihtoehtoista on oikea). Käytännössä ongelmaksi muodostuu monessa pelissä pelipuun valtava koko. Näin on esimerkiksi shakin kohdalla.

Jotta pelissä, jossa peliä ei voida ratkaista minimax-algoritmin avulla pelipuun koon vuoksi, voidaan saada aikaan hyviä, mutta ei välttämättä optimaalisia, pelistrategioita, voidaan pelitilanteiden hyvyttä arvioida erilaisilla [heuristisilla arviointikriteereillä](#). Silloin minimax-algoritmin rekursio pysäytetään esim. kun pelipuussa on saavuttu tietylle syvyydelle vaikei olisikaan päädytty tilanteeseen, jossa peli on ratkennut ja palautetaan pelin lopputuloksen sijaan jonkinlainen arvio siitä, miten peli päättyy, jos se pelataan ko. tilanteesta loppuun asti. Arvio on lisäksi oltava helppo laskea. Shakkipelissä arvio perustuu tyypillisesti kunkin pelaajan jäljellä olevien pelinappuloiden määrään ja laatuun sekä niiden sijoittumiseen laudalla.

Toinen keskeinen pelitekoälyalgoritmien tehostamiskeino on [alpha-beta-karsinta](#). Alpha-beta-karsinnassa minimax-algoritmin kumpaakin funktiota muokataan lisäämällä niihin α - ja β -arvot, joiden perusteella voidaan osa alipuista jättää kokonaan laskematta turhina. Saatavat pelisiirrot ovat silti tarkalleen samat kuin minimax-algoritmillä.

Alpha-beta-karsintaa selventävä video löytyy kurssin sivulta 2. luennon kohdalta. Siinä esimerkkinä käytettävä ristinollapeli alkaa kuvan 4 juurisolmun esittämästä tilanteesta.

MAX-ARVO(Solmu)	% Max-pelaajan funktio
-----------------	------------------------

```

if LOPPUTILA(Solmu) return(ARVO(Solmu))
v =  $-\infty$ 
for each Lapsi in LAPSET(Solmu)
  v = MAX(v, MIN-ARVO(Lapsi,  $\alpha$ ,  $\beta$ ))
  if v  $\geq$   $\beta$  return v
   $\alpha$  = MAX( $\alpha$ , v)
return(v)

```

Algoritmi 5. Alpha-beta-karsinta: Max-pelaajaa vastaava funktio.

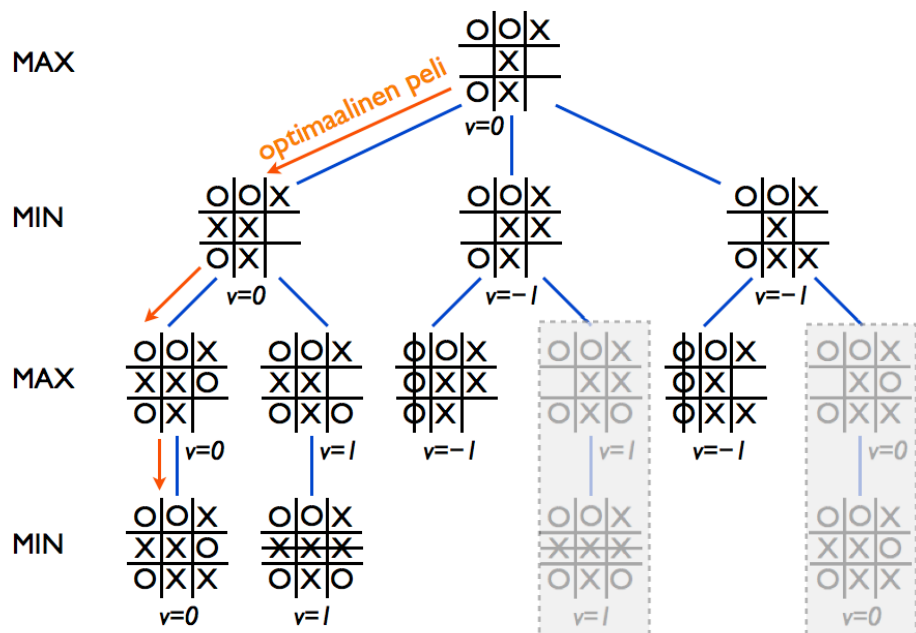
MIN-ARVO(Solmu)	% Min-pelaajan funktio
-----------------	------------------------

```

if LOPPUTILA(Solmu) return(ARVO(Solmu))
v =  $+\infty$ 
for each Lapsi in LAPSET(Solmu)
  v = MIN(v, MAX-ARVO(Lapsi,  $\alpha$ ,  $\beta$ ))
  if v  $\leq$   $\alpha$  return v
   $\beta$  = MIN( $\beta$ , v)
return(v)

```

Algoritmi 6. Alpha-beta-karsinta: Min-pelaajaa vastaava funktio.



Kuva 4. Ristinollapeliä vastaava pelipuu (osittainen). Optimaalinen peli johtaa tasapeliin. Alpha-beta-karsinnassa harmaat alipuut voidaan jättää käymättä läpi. Tulos on silti sama kuin minimax-algoritmillä.

Tähän aiheeseen liittyviä oppimistavoitteita:

Esitiedot

- perustietorakenteet (pino, jono)
- leveys- ja syvyysuuntainen haku (*TiRa*)
- ohjelmointitaito

Lähestyy oppimistavoitetta

- osaa selittää A*-haun perusidean (heuristiikka, kustannusfunktio)
- osaa piirtää annettua peliä vastaavan pelipuun

Saavuttaa oppimistavoitteet

- osaa esittää annettua ongelmaa vastaavan etsintäavaruuden ja ratkaista ongelman etsintäalgoritmia käyttäen
- osaa toteuttaa A*-haun
- osaa toteuttaa minimax-algoritmin ja alpha-beta-karsinnan
- osaa suunnitella heuristisen pelitilanteen arviointikriteerin (esim. shakkiin)

Syventää oppimistavoitteita

- osaa toteuttaa kompleksisia sovelluksia, jotka perustuvat etsintäalgoritmeihin
- osaa toteuttaa shakkia tai muuta epätriviaalia peliä pelaavan algoritmin tehokkaasti

Lisämateriaalia:

- W. Ertel, *Introduction to Algorithms*, Springer 2011, kappale 6: "Search, Games and Problem Solving" (kurssikansiossa)
- Alpha-beta-karsintaa havainnollistava video kuvan 4 ristinollapelissä: <http://www.cs.helsinki.fi/video/alphabeta>

4. Logiikka tekoälyssä

Logiikka oli 1980-luvulle saakka tekoälyn keskeisin lähestymistapa.

Predikaattilogiikan lauseet voivat olla esimerkiksi muotoa

$$(2) \quad \text{isä}(X,Y) \wedge \text{isä}(Y,Z) \Rightarrow \text{isoisä}(X,Z)$$

missä \wedge -merkki tarkoittaa konjunktiota (luetaan "ja"; muistisääntö kuvassa 5).

Päätely voidaan osin automatisoida esim. Prolog-kielessä. Käytännössä on kuitenkin vaikea kirjoittaa ohjelmaa siten, että kaikki relevantti tieto on koodattu oikein: tietokoneelle voidaan unohtaa mainita, että isä ja äiti ovat ainoat selkäjänteisten (?) eliöiden biologiset vanhemmat. Lisäksi joidenkin lauseiden todistaminen voi olla käytännössä niin hankalaa, että vaikka ne pitävätkin paikkansa



Kuva 5. Muistisääntö konjunktioille:
Con hueso, kivillä.

(esim. P vs NP -ongelma tai mikä tahansa muu toistaiseksi ratkaisematon matemaattinen pulma), ei niitä voida koneellisesti todistaa.

Gödelin epätäydellisyyslause sanoo vieläpä, että on olemassa ratkeamattomia lauseita, eli tosia tai epätosia lauseita, joiden totuusarvoa ei voida edes periaatteessa todistaa! (Mikä on siis eri asia kuin se, että joidenkin lauseiden todistaminen kestää liian kauan.)

Prologia käytettiin esimerkkinä **logiikkaohjelmoinnista**. Siinä aihealueen kannalta relevantti tietämys kuvataan joukkona predikaattilogiikan lauseita (esim. "kissa(felix)") ja "ohjelman" suoritus aloitetaan tekemällä kysely "voidaanko muotoa predikaatti(argumentit) oleva väite todistaa?": esim. "voidaanko väite eläin(felix) todistaa?"

CYC on esimerkki 1980-luvulla alkunsa saaneesta hankkeesta, jonka toivottiin (toivotaan?) johtavan ihmistasoiseen älykkyyteen kasaamalla yhteen valtava määrä logiikan avulla esitettyä tietämystä. Suuret lupaukset ovat toistaiseksi jääneet joka kerta täyttämättä.



Huom: Prolog ja CYC eivät kuulu koealueeseen muutoin kuin yleisellä tasolla. Esimerkiksi syntaksia ei tarvitse opetella. Niitä käsiteltiin esimerkkeinä, joiden kautta voi nähdä minkälaisiin ongelmiin on törmätty logiikkaan perustuvassa tekoälyssä. Tällaisia ongelmia ovat etenkin

- a) kaiken relevantin tiedon ja arkijärjen ("**common sense**") esittäminen formaalissa muodossa,
- b) **skaalautuminen** ongelmien koon kasvaessa siten, että laskenta-aika säilyy siedettävissä rajoissa,
- c) **epävarman** ja ristiriitaisen tiedon käsittely
- d) sekä jossain (pienehkössä) määrin loogiset paradoksit ja **ratkeamattomuus**.

Kun havaittiin, että näitä ongelmia ei pystytäkään ratkaisemaan toivotussa aikataulussa, tutkimusrahoitus logiikkaan perustuvalle Good Old Fashioned AI:lle eli **GOFAI**:lle romahti ja sitä suunnattiin ns. "**modernin tekoälyn**" aiheisiin, kuten neuroverkkoihin ja hiukan myöhemmin etenkin todennäköisyyksille pohjaaviin menetelmiin.

Logiikkaa käytetään toki edelleen monella tekoälyn osa-alueella, kuten automaattisessa teoreemantodistuksessa, ohjelmien oikeellisuuden todentamisessa, koneoppimisessa, jne.

Tähän aiheeseen liittyvistä oppimistavoitteista, ks. sivu 3.

Tässä vaiheessa kurssia jätettiin taakse GOFAI ja siirryttiin modernin tekoälyn puolelle.

5. Päättely epävarmuuden vallitessa: Todennäköisyydestä

Oikeassa maailmassa pitää pystyä ottamaan huomioon monenlaisia epävarmuustekijöitä. Esimerkkinä auto, joka ei starttaa: ongelma voidaan yrittää paikallistaa laskemalla eri vikojen todennäköisyys. Todennäköisyys muuttuu, kun saadaan uutta tietoa (esim. "radio soi").

Todennäköisyyden ei aina tarvitse liittyä fysikaalisesti satunnaiseen tai stokastiseen ilmiöön, vaan todennäköisyys voi kuvata myös tiedon puutteesta johtuvaa epävarmuutta: esim. $P(\text{"Herra Azzip söi eilen pizzaa"})$. Todennäköisyydet riippuvat myös taustatiedosta ja voivat olla subjektiivisia eli riippua henkilöstä.

Tekoälysovelluksissa vaikein osa on muodostaa sopiva **todennäköisyysmalli**, joka kuvastaa riittävässä määrin kyseessä olevaa ilmiötä. **Todennäköisyyspäättely** on periaatteessa suoraviivaista, koska se seuraa tunnettuja todennäköisyyslaskennan sääntöjä, mutta se voi silti olla matemaattisesti tai laskennallisesti hyvinkin vaikeaa – samaan tapaan kuin looginen päättely.

Todennäköisyyslaskennan peruskäsitteitä ovat

- a) **alkeistapahtumat** $\omega \in \Omega$, jotka vastaavat mahdollisia maailmoja
- b) **tapahtumat** $A, B, C, \dots \subseteq \Omega$, jotka ovat osajoukkoja mahdollisista maailmoista
- c) **todennäköisyysmalli**, liittää jokaiseen alkeistapahtumaan $\omega \in \Omega$ **todennäköisyyden** $P(\omega) \in [0, 1]$; näiden tulee summautua yhteen eli $\sum_{\omega \in \Omega} P(\omega) = 1$

Edellisten perusteella saadaan tapahtuman $A \subseteq \Omega$ todennäköisyys laskemalla yhteen tapahtumaan A kuuluvien alkeistapahtumien todennäköisyydet:

$$(3) \quad P(A) = \sum_{\omega \in A} P(\omega) \in [0, 1].$$

Tapahtumista A, B, C, \dots voidaan edelleen johtaa uusia tapahtumia yhdistelemällä:

- a) **negaatio**: $\neg A = \{\omega : \omega \notin A\}$ eli ne mahdolliset maailmat, joissa A ei päde.
- b) **konjunktio**: $A \wedge B = \{\omega : \omega \in A \cap B\}$ eli ne mahdolliset maailmat, joissa pätee sekä A että B
- c) **disjunktio**: $A \vee B = \{\omega : \omega \in A \cup B\}$ eli ne mahdolliset maailmat, joissa pätee joko A tai B

Kohdan b tyyppisen tapahtuman eli kahden tai useamman tapahtuman konjunktion todennäköisyyttä sanotaan **yhteistodennäköisyydeksi** (joint probability) ja siitä puhuttaessa merkitään usein $P(A, B)$, mikä siis merkitsee samaa kuin $P(A \wedge B)$. Sekä konjunktion että disjunktion kohdalla pätee symmetrisyys: $A \wedge B$ tarkoittaa samaa

kuin $B \wedge A$ ja vastaavasti $A \vee B$ tarkoittaa samaa kuin $B \vee A$.

Järjestysriippumattomuus koskee myös useamman tapahtuman konjunktioita ja disjunktioita, kuten $A \vee B \vee C$, jne.

Ehdollinen todennäköisyys $P(A | B)$ tarkoittaa tapahtuman A todennäköisyyttä silloin, kun tiedetään, että tapahtuma B tapahtuu. Ehdollinen todennäköisyys voidaan laskea kaavalla

$$(4) \quad P(A | B) = P(A, B) / P(B),$$

kunhan $P(B) > 0$. Ehdollistajana voi olla yksittäisen tapahtuman B lisäksi muitakin tapahtumia, jolloin merkitään $P(A | B, C, \dots)$. Ehdollistavien tapahtumien järjestyksellä ei ole merkitystä, eli $P(A | B, C) = P(A | C, B)$.



Vaikka ehdollistavien tapahtumien järjestyksellä ei olekaan merkitystä, on erittäin tärkeää muistaa, että ehdollisessa todennäköisyydessä sillä, kummalla puolella pystyviivaa tapahtumat ovat on merkitystä: $P(A | B)$ ei siis yleensä ole yhtä kuin $P(B | A)$, jne.

Ehdolliseen todennäköisyyteen liittyy tärkeä käsite: **riippumattomuus**. Jos tapahtuman A todennäköisyys ei riipu siitä, tapahtuuko B vai ei, sanotaan, että A on riippumaton B:stä ja merkitään

$$(5) \quad A \perp B \quad \Leftrightarrow \quad P(A | B) = P(A).$$

Voidaan osoittaa, että riippumattomuus on **symmetrinen** ominaisuus, eli jos A on riippumaton B:stä on B riippumaton A:sta. (Jos et usko, voit kokeilla soveltaa ketjusääntöä ensin yhteen suuntaan $P(A, B) = P(A) P(B | A)$ ja sitten toiseen suuntaan $P(A, B) = P(B) P(A | B)$. Sovella nyt riippumattomuuden määritelmää jompaan kumpaan kaavaan ja toinen seuraa automaattisesti.)

Riippumattomuus voidaan määritellä myös **ehdollistettuna** yhdellä tai useammalla muulla tapahtumalla:

$$(6) \quad A \perp B | C \quad \Leftrightarrow \quad P(A | B, C) = P(A | C),$$

jolloin sanotaan, että A on riippumaton B:stä annettuna C. Voidaan esimerkiksi ajatella, että sisarusten silmien väri ei ole riippumaton, kun vanhempien silmien väriä ei tunneta (sinisilmäisen henkilön sisaruksella on keskimääräistä useammin siniset silmät), mutta annettuna vanhempien silmien väri, sisarusten silmien värit ovat riippumattomat.

Seuraavat laskusäännöt tulevat jatkossa tarpeeseen:

$$(7) \quad P(\neg A) = 1 - P(A) \quad \text{"negaatio"}$$

$$(8) \quad P(A \vee B) = P(A) + P(B) - P(A, B) \quad \text{"inkluisio-eksluusio"}$$

$$(9) \quad P(A) = P(A, B) + P(A, \neg B) \quad \text{"marginalisointi"}$$

$$(10) \quad P(A_1, \dots, A_k) = P(A_1) P(A_2 | A_1) P(A_3 | A_1, A_2) \dots P(A_k | A_1, \dots, A_{k-1})$$

"ketjusääntö"

$$(11) \quad P(A | B) = P(A) P(B | A) / P(B) \quad \text{"Bayesin kaava"}$$

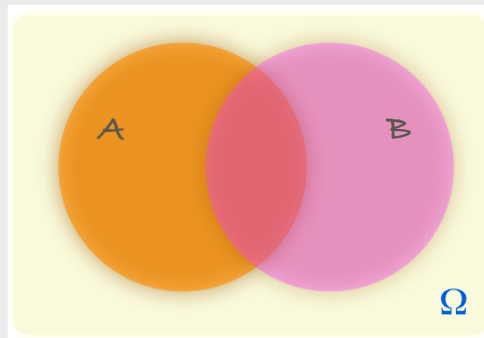
Ketjusäännön kohdalla tapahtumia A_1, \dots, A_k voi olla mielivaltainen määrä.

Yksinkertaisimmillaan tapahtumia on kaksi, jolloin ketjusääntö sanoo $P(A, B) = P(A) P(B | A)$. Tapahtumien järjestyksellä ole ole tässä yhteydessä väliä.

Huomaa, että kaikki edellä olevat kaavat pätevät myös silloin, kun niiden ehdollistajaksi lisätään muita tapahtumia, esim. $P(\neg A | Z) = 1 - P(A | Z)$ tai $P(A | B, Z) = P(A | Z) P(B | A, Z) / P(B | Z)$, missä Z :n paikalla voi olla yksi tai useampia muita tapahtumia.

NIKSI-PIRKKA: VENN-DIAGRAMMI

Todennäköisyyslakennan kaavojen muistamisessa voi olla apua [Venn-diagrammien](#) piirtämisestä.



Kuva 6. Venn-diagrammi kahdella tapahtumalla A ja B.

Venn-diagrammissa kahta tai useampaa tapahtumaan kuvaa kutakin oma kappaleensa (yllä ympyrät). Esimerkiksi kaava (9) $P(A) = P(A, B) + P(A, \neg B)$ vastaa havaintoa, että diagrammissa joukko A saadaan laskemalla yhteen A:n ja B:n leikkaus (ympyröiden päällekkäin osuvat osat) ja A:n se osuus, joka ei kuulu B:hen (oranssi alue).

Tapahtumien lisäksi todennäköisyysmalliin saattaa liittyä [satunnaismuuttujia](#).

Satunnaismuuttuja X on formaalisti funktio, jonka arvo määräytyy alkeistapahtuman perusteella: $X : \Omega \rightarrow X(\Omega)$, missä $X(\Omega)$ on X :n mahdollisten arvojen joukko (arvojoukko), eli $x(\omega) \in X(\Omega)$ kaikilla $\omega \in \Omega$.

Esimerkki: Kahden nopan heitossa alkeistapahtumat ovat muotoa $\omega \in \{(m, n) \mid m, n \in \{1, 2, 3, 4, 5, 6\}\}$. Silmälukujen summa X on nyt satunnaismuuttuja, jonka arvo määräytyy alkeistapahtumasta kaavan $\omega = (m, n) \Rightarrow X(\omega) = m + n$ mukaisesti. Kyseisen satunnaismuuttujan arvojoukko on $X(\Omega) = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$.



Tällä kurssilla puhutaan vain ja ainoastaan ns. diskreeteistä

satunnaismuuttujista eli satunnaismuuttujista, joiden arvojoukko on äärellinen (kuten nopanheiton tulos) tai numeroituvasti ääretön (kuten kokonaislukujen joukko). On olemassa myös muunlaisia satunnaismuuttujia. Esimerkiksi jatkuva satunnaismuuttuja saa arvoja jossakin reaalilukujen joukon ylinumeroituvassa osajoukossa: tällainen muuttuja voisi olla vaikkapa henkilön pituus senttimetreinä.

Jokaiseen satunnaismuuttujaan liittyy [jakauma](#), joka on yksinkertaisesti luettelo todennäköisyyksistä, joilla muuttuja saa kunkin arvon. Edellisen esimerkin nopan silmälukujen summan jakauma on $P_x = (P(X=2), P(X=3), P(X=4), \dots, P(X=12)) = (1/36, 2/36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/36, 2/36, 1/36)$, mikä voidaan osoittaa laskemalla kutakin muuttujan arvoa vastaavien alkeistapahtumien todennäköisyydet yhteen, kun oletetaan, että kunkin alkeistapahtuman todennäköisyys on $1/36$. (Painotetulla nopalla jakauma olisi erilainen.)

Satunnaismuuttujien kautta määritellyille tapahtumille, kuten $A: "X=5"$, pätevät samat laskusäännöt kuin "tavallisille" tapahtumille (ks. yllä). Ainoa merkitsevä pantava huomio koskee kaavan (9) sääntöä eli [marginalisointisääntöä](#) $P(A) = P(A, B) + P(A, \neg B)$. Se pätee toki edelleen, vaikka tapahtumat A ja B olisi määritelty satunnaismuuttujien avulla, mutta usein edellistä kätevämpi on muoto:

$$(12) \quad P(X=x) = \sum_{y \in \text{dom}(Y)} P(X=x, Y=y),$$

missä siis summassa lasketaan yhteen termit kaikilla satunnaismuuttuja Y :n mahdollisilla arvoilla. Säännön tulkinta on seuraava: X :n todennäköisyys saada arvo x voidaan laskea ynnäämällä kaikkien kaikkien niiden tapahtumien todennäköisyydet, joissa X saa arvon x ja Y saa arvon y , missä y voi olla mitä tahansa.

Satunnaismuuttujien yhteydessä hieman turhan pitkältä tuntuva merkintä $P(X=x)$, missä X on satunnaismuuttuja ja x sen mielivaltainen arvo, lyhennetään usein $P(x)$. Näin esimerkiksi kahden muuttujan yhteistodennäköisyyden ketjusääntö voidaan kirjoittaa joko pitkästi:

$$(13) \quad P(X=x, Y=y) = P(X=x) P(Y=y | X=x)$$

tai [lyhennysmerkinnällä](#):

$$(14) \quad P(x, y) = P(x) P(y | x).$$

Bayesin kaavalla on merkittävä rooli [tilastollisessa päättelyssä](#) ja koneoppimisessa, mikä johtuu siitä, että sitä voidaan soveltaa tilanteissa, joissa yksi muuttujista vastaa jonkinlaista "tilaa", jota ei tunneta, mutta josta ollaan kiinnostuneita ja toinen muuttuja (tai toiset muuttujat) vastaavat havaintoa (tai havaintoja), jonka avulla tilasta voidaan päätellä jotakin:

$$(15) \quad P(\text{tila} | \text{havainto}) = P(\text{tila}) P(\text{havainto} | \text{tila}) / P(\text{havainto}).$$

Vasemman puolen termiä nimitetään **posterioritodennäköisyydeksi** (havainnon jälkeinen todennäköisyys). Oikean puolen ensimmäinen termi on **prioritodennäköisyys** (havaintoa edeltävä todennäköisyys), seuraavaa termiä kutsutaan **uskottavuudeksi** (havainnon todennäköisyys, kun tila tunnetaan) ja viimeistä termiä monella termillä, joista "ärsyttävä **nimittäjä**" lienee osuvin, muttei yleisin. (Evidenssi ja marginaaliuskottavuus (*marginal likelihood*) ovat yleisempiä.)

NIKSI-PIRKKA: NIMITTÄJÄN LASKEMISESTA

Nimittäjän laskeminen voi joskus olla työlästä, kuten luennon lääketieteelliseen diagnoosiin liittyvässä esimerkissä ja laskuharjoitustehtävässä huomattiin. Joskus on helpointa jättää nimittäjä laskematta suoraan ja laskea sen sijaan eri tilojen **posterioritodennäköisyyksien suhde**, esimerkiksi:

$$(16) \quad \frac{P(\text{Tila}=1 \mid h)}{P(\text{Tila}=2 \mid h)},$$

missä h on lyhenne havainnosta. Tätä laskettaessa nimittäjä $P(h)$ nimittäin **supistuu** pois:

$$(17) \quad \frac{P(\text{Tila}=1) P(h \mid \text{Tila}=1) / P(h)}{P(\text{Tila}=2) P(h \mid \text{Tila}=2) / P(h)} = \frac{P(\text{Tila}=1) P(h \mid \text{Tila}=1)}{P(\text{Tila}=2) P(h \mid \text{Tila}=2)}.$$

Tarvittaessa nimittäjä voidaan laskea summaamalla yhteen edellisessä osamäärässä esiintyvät tekijät kaikilla Tila-muuttujan arvoilla:

$$(18) \quad P(h) = \sum_{t \in \text{dom}(\text{Tila})} P(\text{Tila}=t) P(h \mid \text{Tila}=t).$$

Voit vakuuttaa itsesi edellisen kaavan oikeellisuudesta soveltamalla oikealla puolella ketjusääntöä sekä yllä esitetyn kahden tapahtuman marginalisointisäännön yleistystä useamman tapahtuman tapaukseen.

Materiaalia:

- P. Tuominen, *Todennäköisyyslaskenta I*, Limes, 1994
- W. Ertel: Introduction to Artificial Intelligence, Springer, 2011. Kappale 7 "Reasoning with Uncertainty" (s. 122 asti)

6. Päätely epävarmuuden vallitessa: Bayes-verkot

Edellisessä kappaleessa esitettyjä todennäköisyyslaskennan menetelmiä voidaan soveltaa monessa tekoälyyn liittyvässä sovelluksessa. Kuten sanottua, vaikein osa on usein todennäköisyysmallin rakentaminen, eli eri alkeistapahtumien todennäköisyyksien määrittäminen. Useimmiten mallinnettava tilanne on niin laaja,

että kaikkien alkeistapahtumien luettelointi ja todennäköisyyden liittäminen niihin yksitellen ei ole mielekästä. Yksi tapa helpottaa todennäköisyyksien määrittämistä on [Bayes-verkkojen](#) käyttö.

Bayes-verkon rakennuspalikat

Bayes-verkossa ongelmakenttä mallinnetaan joukkona satunnaismuuttujia ja niiden välisiä riippuvuuksia (ja riippumattomuuksia). Riippuen Bayes-verkon rakenteesta (verkon kaarien määrästä ja paikoista), voi joskus riittää määrätä huomattavasti pienempi määrä todennäköisyysarvoja verrattuna alkeistapahtumien joukon kokoon.

Bayes-verkko koostuu joukosta [solmuja](#) (muuttujat), niiden välisistä [kaarista](#) (suorat riippuvuudet), ja [parametreista](#), jotka määrittävät kunkin solmun ehdollisen todennäköisyysjakauman annettuna ko. solmun [vanhempien](#) arvot. Esimerkki parametriarvosta ja sen määrittämisestä todennäköisyydestä on

$$(19) \quad P(\text{"KÄYNNISTYY"} \mid \text{"SYTYTYS TOIMII"} \wedge \text{"BENSAA"}) = 0.99 ,$$

tai lyhennysmerkinnöin

$$(20) \quad P(K \mid S, B) = 0.99 .$$

TAPAHTUMAT VS MUUTTUJAT: ERÄ 2

Edellä todennäköisyyslaskennan sääntöjä on käsitelty välillä tapahtumien ja välillä satunnaismuuttujien kautta. Näiden välillä on selkeä yhteys: kukin satunnaismuuttuja ja sitä vastaava arvo määrittelee tapahtuman. Jos X on satunnaismuuttuja, voidaan esimerkiksi määritellä tapahtuma A : " $X = 3$ ". Lisäksi jos X ja Y ovat satunnaismuuttujia, voidaan määritellä tapahtuma B : " $X < Y$ ", jne.

Sen lisäksi, että satunnaismuuttujien avulla voidaan määritellä tapahtumia eli alkeistapahtumien osajoukkoja, voidaan koko todennäköisyysmalli alkeistapahtumineen määritellä suoraan muuttujien avulla. Jos esimerkiksi heitetään kahta noppaa, voidaan sanoa, että kummankin nopan silmäluku on oma muuttujansa, M ja N . Jos tilanteessa ei ole tarpeen mallintaa muita ilmiöitä (pizzaa, tms.), voidaan alkeistapahtumat johtaa suoraan muuttujien M ja N arvoista m ja n määrittelemällä $\omega = (m, n)$. (Huomaa, että edellisen kappaleen nopanheittoesimerkissä päädyttiin samaan malliin aloittamalla alkeistapahtumien määrittelystä. Nyt lähdettiin liikkeelle muuttujista ja määriteltiin niiden avulla alkeistapahtumat. Ero on ehkä hienoinen, mutta se saattaa olla tärkeä tiedostaa.)

Autoesimerkissä jokaista muuttujaa A, R, S, B, K, L kohti on olemassa vastaava tapahtuma " A kussa on virtaa" $\Leftrightarrow A=1$, " R adio soi" $\Leftrightarrow B=1$, jne., joten voimme samaistaa ko. tapahtumat ja muuttujat. Yhdessä satunnaismuuttujat A, R, S, B, K, L määrittävät kaiken olennaisen, joten voimme sopia, että alkeistapahtumat ovat muotoa $\omega = (a, r, s, b, k, l)$, missä $a, r, s, b, k, l \in \{0, 1\}$. Nyt alkeistapahtumia on $2^6 = 64$ ja niiden todennäköisyyksien vetäminen hatusta olisi työlästä. Kuten jatkossa

huomaamme, Bayes-verkko helpottaa tätä tehtävää.

Autoesimerkissä Bayes-verkon parametrien määräämät todennäköisyydet ovat kokonaisuudessaan:

$$(21) \quad P(A) = 0.9$$

$$(22) \quad \begin{aligned} P(R | A) &= 0.9 \\ P(R | \neg A) &= 0 \end{aligned}$$

$$(23) \quad \begin{aligned} P(S | A) &= 0.95 \\ P(S | \neg A) &= 0 \end{aligned}$$

$$(24) \quad P(B) = 0.95$$

$$(25) \quad \begin{aligned} P(K | S, B) &= 0.99 & P(K | S, \neg B) &= 0 \\ P(K | \neg S, B) &= 0 & P(K | \neg S, \neg B) &= 0 \end{aligned}$$

$$(26) \quad \begin{aligned} P(L | K) &= 0.99 \\ P(L | \neg K) &= 0 \end{aligned}$$

Muotoa $P(\neg X | Z)$ olevat todennäköisyydet saadaan luonnollisesti kaavan (7) avulla $P(\neg X | Z) = 1 - P(X | Z)$. Nämä yhdessä Bayes-verkon rakenteen kanssa määräävät koko todennäköisyysmallin.

Kun Bayes-verkko ja sen parametrit on määrätty, voidaan laskea minkä tahansa alkeistapahtuman todennäköisyys [kertomalla yhteen](#) kunkin muuttujan todennäköisyys:

$$(27) \quad \begin{aligned} P(A, R, S, B, \neg K, \neg L) &= P(A) P(R|A) P(S|A) P(B) P(\neg K|S, B) P(\neg L|\neg K) \\ &= 0.9 \times 0.9 \times 0.95 \times 0.95 \times (1 - 0.99) \times (1 - 0) \approx 0.00731 . \end{aligned}$$

Kunkin muuttujan kohdalla ehdollistajana, eli pystyviivan oikealla puolella ovat ko. muuttujan vanhemmat.

Edellistä kaavaa voidaan verrata aiemmin esitettyyn ketjusääntöön:

$$(28) \quad \begin{aligned} P(A, R, S, B, \neg K, L) &= P(A) P(R|A) P(S|A, R) P(B|A, R, S) \\ &\quad \times P(\neg K|A, R, S, B) P(L|A, R, S, B, \neg K) , \end{aligned}$$

missä jokaisen tapahtuman (muuttujan) kohdalla ehdollistajana ovat kaikki listassa edellä olevat muuttujat.

Huomaa, että ketjusääntö pätee joka tilanteessa ja joka järjestyksessä. Jos olisimme soveltaneet sitä Bayes-verkon sijaan, olisimme joutuneet luettelemaan paljon enemmän todennäköisyysarvoja. Esimerkiksi viimeisen muuttujan, L, kohdalla tarvittaisiin yksi todennäköisyysarvo jokaista muuttujien AR,S,B,K arvojen yhdistelmää kohti, eli yhteensä $2^5 = 32$ lukuarvoa. Bayes-verkon implikoima kaava on kätevämpi, koska kuten yllä todettiin, sen avulla tarvitsee luetella yhteensä vain

12 lukuarvoa koko jakauman määrittämiseksi. Olemme siis saaneet määriteltä 64 alkeistapahtuman todennäköisyydet vain 12 lukuarvon avulla. Kätevää!

Eksakti päättely Bayes-verkossa

Kun nyt osaamme laskea alkeistapahtumien todennäköisyydet kaavan (27) tapaan, voidaan muiden tapahtumien todennäköisyydet laskea vanhan kunnon yhteenlaskukaavan (3) avulla tähän tapaan:

$$\begin{aligned}(29) \quad P(A, R, B, \neg K) &= \sum_{\omega \in (A, R, B, \neg K)} P(\omega) \\ &= P(A, R, S, B, \neg K, L) \\ &\quad + P(A, R, S, B, \neg K, \neg L) \\ &\quad + P(A, R, \neg S, B, \neg K, L) \\ &\quad + P(A, R, \neg S, B, \neg K, \neg L)\end{aligned}$$

Todennäköisyyden laskeminen jätetään harjoitustehtäväksi – huomaa, että summan toinen termi $P(A, R, S, B, \neg K, \neg L)$ on laskettu valmiiksi kaavassa (27).

Jos ollaan kiinnostuneita ehdollisista todennäköisyyksistä, kuten

$$(30) \quad P(A \mid R, B, \neg K)$$

voidaan soveltaa ehdollisen todennäköisyyden kaavaa (4):

$$(31) \quad \frac{P(A, R, B, \neg K)}{P(R, B, \neg K)},$$

missä kumpikin osamäärän osa voidaan laskea erikseen kaavan (29) tapaan.

Alkeistapahtumien todennäköisyyksien summaaminen voi joskus olla liian työlästä, jos mallissa on hyvin monta muuttujaa. Siksi käytännössä edellä esitettyä päättelymenetelmää sovelletaan vain yksinkertaisimmissa tilanteissa.

Monimutkaisemmissa malleissa voidaan usein laskea **eksakti** (tarkalleen oikea) vastaus soveltaen tehokkaampia **päättelyalgoritmeja**, kuten ns. *junction tree* -algoritmia, mutta se ei sisälly tämän kurssin aihepiireihin. Toinen vaihtoehto on soveltaa **approksimatiivista** päättelyä, joka on periaatteessa hyvin suoraviivaista, mutta tulos ei ole eksakti. Tutustumme tähän seuraavaksi, mitä varten on tarpeen selvittää, miten Bayes-verkosta voidaan generoida dataa.

Datan generoiminen Bayes-verkosta

Bayes-verkosta on helppo **generoida dataa** eli havaintoja, jotka noudattavat Bayes-verkon määrittelemää todennäköisyysmallia. Data koostuu joukosta **monikkoja**, joista jokainen on lista, joka sisältää kaikkien mallissa olevien muuttujien (solmujen) arvot. Monikko vastaa siis yhtä alkeistapahtumaa ω .

Datan generoiminen tapahtuu tuottamalla tietokoneella **satunnaislukuja** (tai tarkemmin sanottuna pseudosatunnaislukuja, koska satunnaislukugeneraattorit

perustuvat deterministisiin laskukaavoihin, joiden tulos vain näyttää satunnaiselta, siksi "pseudo-"). Jokaisen monikon jokainen muuttuja poimitaan ko. muuttujan ehdollisesta jakaumasta annettuna sen vanhempien arvot kyseisessä monikossa, ks. algoritmi 6 alla.

GENEROI-MONIKKOJA(N, Malli):

```
for i = 1 to N:
    for X in Malli.Muuttujat:
        X = SATUNNAISLUKU(X.Jakauma(VANHEMMAT(X)))
    print X
```

Algoritmi 6. Datan generoiminen Bayes-verkosta.

SATUNNAISLUKU(Jakauma):

```
r = TASAJAKAUMA(0,1)
if r > Jakauma[0] then return 1
else return 0
```

Algoritmi 7. Tiettyä jakaumaa noudattavan binäärisen (kaksiarvoisen) satunnaisluvun poimiminen. Argumentti Jakauma on kahden luvun lista, joista ensimmäinen on arvon 0 todennäköisyys.

SATUNNAISLUKU(Jakauma, k):

```
r = TASAJAKAUMA(0,1)
i = 0
while r > Jakauma[i]:
    r = r - Jakauma[i]
    i = i + 1
return i
```

Algoritmi 8. Tiettyä jakaumaa noudattavan k-arvoisen satunnaisluvun poimiminen. Argumentti Jakauma on k luvun lista, jossa on lueteltuna kaikkien muuttujan arvojen todennäköisyydet.

Jos ollaan esimerkiksi poimimassa yhtä autoesimerkin monikkoa, voidaan ensin poimia muuttujan $A = \text{"Akussa on virtaa"}$ arvo, joka olkoon 1, jos vastaava tapahtuma tapahtuu ja 0 muuten. Tällä muuttujalla ei ole verkossa vanhempia, joten sen jakauma on yksinkertaisesti kaavassa (21) esiintyvä (ei-ehdollinen) jakauma $P(A = 1) = 0.9$ (ja kääntäen $P(A = 0) = 1 - P(A = 1) = 0.1$). Muuttujan A -arvo poimitaan siis jakaumasta (0.1, 0.9), missä päätimme merkitä nollan todennäköisyyden ennen ykkösen todennäköisyyttä. Toisin sanoen, A :n arvon tulee olla 0 todennäköisyydellä 0.1 ja 1 todennäköisyydellä 0.9.

Muuttujat poimitaan sellaisessa järjestyksessä, että kunkin muuttujan vanhempien arvot on poimittu ennen kyseistä muuttujaa. Tämä on tarpeen, jotta voidaan valita oikea ehdollinen jakauma. Jos esimerkiksi muuttujalle A on poimittu arvo 0, on autoesimerkissä muuttujan R jakauma (0.0, 1.0), eli R saa varmuudella arvon 0. Jos

taas A:n arvoksi poimittiin 1, on R:n jakauma (0,1,0.9), ks. kaava (22), eli R saa 90% todennäköisyydellä arvon 1.

Jos muuttujalla on useampi vanhempi, kuten muuttujalla K (käynnistyy), riippuu ehdollinen jakauma niistä kaikista, ks. kaava (25).

Algoritmeissa 7 ja 8 esiintyvä TASAJAKAUMA(0,1)-kutsu palauttaa reaalityyppistä lukua (esim. double), joka on tasaisesti jakautunut välillä [0,1]. Javassa `java.util.Random.random()`. Pythonissa `random.random()`.

Approksimatiivinen päättely Bayes-verkossa

Edellä esitettyä datan generointitekniikkaa voidaan soveltaa todennäköisyyspäättelyyn. Ideana on arvioida todennäköisyyksiä ns. [Monte Carlo -tekniikalla](#). Arviot eivät ole täsmälleen oikeita, joten todennäköisyyspäättely tällä menetelmällä on [approksimatiivista](#) (eli likimääräistä).

Arvio perustuu siihen, että minkä tahansa tapahtuman frekvenssi (tapahtumien suhteellinen osuus) suppenee kohti ko. tapahtuman todennäköisyyttä:

$$(32) \quad P(A) \rightarrow \frac{\text{Kuinka monessa tapauksessa A pätee}}{\text{Tapausten lukumäärä}},$$

kun tapausten lukumäärä kasvaa rajatta.

Voit esimerkiksi kokeilla nostaa pakasta kaksi korttia (sitä, että panet ensimmäisen kortin välillä takaisin pakkaan) ja laskea kuinka monta kertaa N yrityksestä saat molemmilla nostoilla saman kortin. Näin saadun arvion pitäisi lopulta antaa hyvä arvio laskuharjoitustehtävän 2.1.b oikeasta vastauksesta. (Itse jaksoin kokeilla 20 kertaa, joista 4:ssä kortit olivat samaa maata. Kuinka lähellä näin saatu arvio $4/20 = 0.2$ on oikeaa vastausta?)

Vastaavasti ehdollisia todennäköisyyksiä $P(A | B)$ voi arvioida soveltamalla edellistä arviota todennäköisyyksiin $P(A, B)$ ja $P(B)$ ja käyttämälle sen jälkeen kaavaa (4). Tästä saadaan arvio

$$(33) \quad P(A | B) \approx \frac{\text{Kuinka monessa tapauksessa pätee sekä A että B}}{\text{Kuinka monessa tapauksessa pätee B}},$$

missä voidaan tuki joutua ongelmiin, jos poimitujen tapausten joukossa ei ole yhtään sellaista, jossa B pätee, koska silloin arvioksi saataisiin 0/0.

Materiaalia:

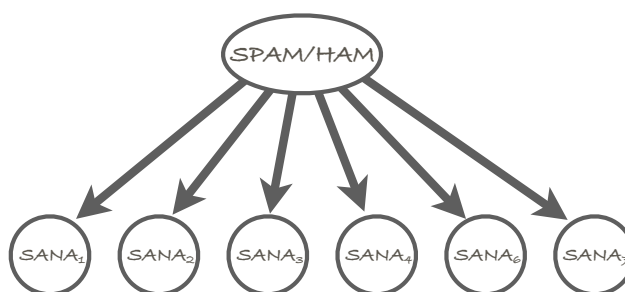
- W. Ertel, *Introduction to Artificial Intelligence*, Springer 2011, kappale 7.4 "Reasoning with Bayesian Networks".

7. Päättely epävarmuuden vallitessa: Roskapostisuodatin

Yksi käyttökelpoisimmista Bayes-verkkoihin perustuvista menetelmistä on niin sanottu **naivi-Bayes -luokitin**. Se on **koneoppimismenetelmä**, jonka avulla voi **luokitella** esimerkiksi tekstidokumentteja kahteen tai useampaan luokkaan. Luokitin opetetaan esittämällä sille joukko **opetusesimerkkejä**, joiden oikea luokka tunnetaan. Koneoppimisen käsitteisiin, kuten luokitteluun ja esimerkkeihin palataan vielä myöhemminkin kurssilla.

Naivi-Bayes-mallissa on yksi **luokkamuuttuja** sekä joukko **piirremuuttujia**. Piirremuuttujien oletetaan olevan **riippumattomia** toisistaan annettuna luokka.

Silloin kun luokitinta käytetään **roskapostin** suodattamiseen, eli sähköpostiviestien luokitteluun roskapostiin ("spam") tai asialliseen postiin ("ham"), luokkamuuttujana on viestin tyyppi (spam/ham) ja piirremuuttujina viestin sisältämät sanat, kuva 7.



Kuva 7. Naivi-Bayes -luokitin roskapostinsuodattamiseen.

Jotta naivi-Bayes -mallin avulla saadaan määriteltyä todennäköisyysmalli, on jokaisen muuttujan kohdalla määriteltävä (ehdollinen) jakauma annettuna sen vanhemmat. Luokkamuuttujan kohdalla vanhempia ei ole, joten riittää määritellä todennäköisyys, jolla saapuva viesti on roskapostia. Tämä voidaan arvioida esimerkiksi arvioimalla, kuinka suuri osa saapuvasta postista on roskaa. Lisäksi on määriteltävä kunkin yksittäisen sanan jakauma, eli todennäköisyys, jolla viestin ensimmäinen, toinen, kolmas, jne., sana on mikä tahansa sana – on siis määriteltävä todennäköisyys, jolla viestin ensimmäinen sana on "Aadolf", "aakkonen", "aallokko", ..., "Öölanti" ja samat todennäköisyydet toisesta viimeiseen sanaan saakka.

Jotta ei olisi tarvetta mallintaa jokaisen sanan jakaumaa erikseen, oletetaan että viestin kaikkien sanojen jakauma annettuna luokka on sama. Sanojen jakaumat voidaan arvioida opetusesimerkeistä laskemalla erilaisten sanojen esiintymistiheydet roskapostissa ja asiallisessa postissa, jota kumpaakin on oltava riittävän paljon, jotta arvio osuisi lähelle oikeaa. Jos siis esimerkiksi sana "cheap" esiintyy roskapostiesimerkeissä yhteensä 147 kertaa ja näiden viestin yhteenlaskettu sanamäärä on 100 000, voidaan arvioida

$$(34) \quad P(\text{Sana}_i = \text{'cheap'} \mid \text{Luokka} = \text{spam}) \approx \frac{147}{100\,000} = 0.00147$$

kaikille sanoille $i \in \{1,2,3,\dots\}$, eli noin 0.15 %. Jos sama sana esiintyy asiallisissa viesteissä 45 kertaa ja jos asiallisten viestin yhteenlaskettu sanamäärä on 250 000, voidaan vastaavasti arvioida

$$(35) \quad P(\text{Sana}_i = \text{'cheap'} \mid \text{Luokka} = \text{ham}) \approx \frac{45}{250\,000} = 0.00018,$$

eli noin 0.02 %. Vaikka kumpikin luku on melko pieni, on tärkeää huomata, että jälkimmäinen on selkeästi pienempi kuin edellinen. Tämä on tärkeää, sillä osoittautuu, että olennainen tekijä on näiden todennäköisyyksien suhde $0.00147 / 0.00018 \approx 8.2$. Sana 'cheap' on siis roskaposteissa noin kahdeksan kertaa niin yleinen kuin asiallisissa posteissa.

Olkoon nyt luokkamuuttujan jakauma sekä sanojen jakaumat annettuna luokka, eli ns. [luokkaehdolliset](#) (class-conditional) jakaumat arvioitu datasta. Saamme uuden viestin, joka alkaa sanoilla "buy cheap algorithm". (Esimerkki on kuvitteellinen.) Miten luokitin arvioi, onko viesti roskapostia vai ei?

Itse asiassa luokitin ei pelkästään arvioi onko viesti roskaa vai ei, vaan se tuottaa arvion siitä, kuinka todennäköisesti viesti on roskaa, eli

$$(36) \quad P(\text{Luokka} = \text{spam} \mid \text{Sana}_1 = \text{'buy'}, \text{Sana}_2 = \text{'cheap'}, \text{Sana}_3 = \text{'algorithm'}),$$

jonka lyhennämme ilman suurta sekaannuksen vaaraa

$$(37) \quad P(\text{spam} \mid \text{'buy'}, \text{'cheap'}, \text{'algorithm'}).$$

Ehdollinen todennäköisyys voidaan nyt kirjoittaa

$$(38) \quad P(\text{spam} \mid \text{'buy'}, \text{'cheap'}, \text{'algorithm'}) = \frac{P(\text{spam}, \text{'buy'}, \text{'cheap'}, \text{'algorithm'})}{P(\text{'buy'}, \text{'cheap'}, \text{'algorithm'})},$$

Joka on olennaisesti Bayesin kaava – emme vain toistaiseksi halunneet jakaa osoittajan yhteistodennäköisyyttä osiin $P(\text{spam}) P(\text{'buy'}, \text{'cheap'}, \text{'algorithm'} \mid \text{spam})$.

Jotta saamme yhtälön oikealla puolella olevan osamäärän osoittajan laskettua, voimme hyödyntää Bayes-verkon rakenteen (naivi-Bayes, kuva 7) mukaista kertolaskusääntöä

$$(39) \quad \begin{aligned} &P(\text{spam}, \text{'buy'}, \text{'cheap'}, \text{'algorithm'}) \\ &= P(\text{spam}) P(\text{'buy'} \mid \text{spam}) P(\text{'cheap'} \mid \text{spam}) P(\text{'algorithm'} \mid \text{spam}), \end{aligned}$$

missä on huomattavaa, että kunkin sanan todennäköisyys riippuu ainoastaan viestin luokasta (spam/ham), eikä viestin muista sanoista. Kaikki kaavassa (39) yhtäsuuruusmerkin oikealla puolella esiintyvät todennäköisyydet on oletettu tunnetuksi, joten olemme kaavan (38) nimittäjän osalta valmiit.

Seuraavaksi olisi edessä kaavan (38) ("ärsyttävä") nimittäjä. Nyt kuitenkin sovellamme sivun 15 niksiä, jolla vältämme ainakin eksplisiittisesti nimittäjän laskemisen. Niksi on siis laskea kaavan (38) asemesta roskapostin ja asiallisen postin suhteellinen todennäköisyys eli

$$(40) \quad \frac{P(\text{spam} \mid \text{'buy'}, \text{'cheap'}, \text{'algorithm'})}{P(\text{ham} \mid \text{'buy'}, \text{'cheap'}, \text{'algorithm'})}$$

$$= \frac{P(\text{spam}, \text{'buy'}, \text{'cheap'}, \text{'algorithm'}) / P(\text{'buy'}, \text{'cheap'}, \text{'algorithm'})}{P(\text{ham}, \text{'buy'}, \text{'cheap'}, \text{'algorithm'}) / P(\text{'buy'}, \text{'cheap'}, \text{'algorithm'})},$$

missä huomaamme, että yhteinen nimittäjä $P(\text{'buy'}, \text{'cheap'}, \text{'algorithm'})$ supistuu pois. Jäljelle jäävässä lausekkeessa esiintyy osamäärän osoittajana kaavassa (39) laskettu tekijä ja nimittäjä on samaa muotoa oleva

$$(41) \quad P(\text{ham}, \text{'buy'}, \text{'cheap'}, \text{'algorithm'})$$

$$= P(\text{ham}) P(\text{'buy'} \mid \text{ham}) P(\text{'cheap'} \mid \text{ham}) P(\text{'algorithm'} \mid \text{ham}),$$

jonka osaamme myös laskea, koska oikealla puolella olevat neljä termiä ovat jokainen tunnettuja, tai pikemminkin meillä on niille esimerkkiainestosta lasketut arviot.

Summa summarum, voimme laskea kaavan (40) osamäärän yhdistämällä kaavojen (39) ja (41) oikeat puolet:

$$(42) \quad \frac{P(\text{spam} \mid \text{'buy'}, \text{'cheap'}, \text{'algorithm'})}{P(\text{ham} \mid \text{'buy'}, \text{'cheap'}, \text{'algorithm'})}$$

$$= \frac{P(\text{spam}) P(\text{'buy'} \mid \text{spam}) P(\text{'cheap'} \mid \text{spam}) P(\text{'algorithm'} \mid \text{spam})}{P(\text{ham}) P(\text{'buy'} \mid \text{ham}) P(\text{'cheap'} \mid \text{ham}) P(\text{'algorithm'} \mid \text{ham})}.$$

Koska tulojen osamäärä on sama kuin osamäärien tulo (siis esimerkiksi $(A \times B \times C) / (D \times E \times F) = (A/D) \times (B/E) \times (C/F)$), voidaan edellinen kirjoittaa yleisessä tapauksessa muodossa

$$(43) \quad \frac{P(\text{spam} \mid \text{viesti})}{P(\text{ham} \mid \text{viesti})} = \frac{P(\text{spam})}{P(\text{ham})} \prod_{i=1}^N \frac{P(\text{sana}_i \mid \text{spam})}{P(\text{sana}_i \mid \text{ham})},$$

missä siis lasketaan tulo priorisuhteesta $P(\text{spam})/P(\text{ham})$ ja kaikista viestin sanojen, $\text{sana}_1, \dots, \text{sana}_N$, suhteellisista todennäköisyyksistä roskaposteissa ja asiallisissa posteissa.

Huomaa että esimerkiksi toisen sanan kohdalla, $i=2$, tulon tekijänä on esimerkkitestin 'buy cheap algorithm' tapauksessa

$$(44) \quad \frac{P(\text{'cheap'} \mid \text{spam})}{P(\text{'cheap'} \mid \text{ham})} \approx 8.2,$$

kuten kappaleen alussa (kaavat (34) ja (35)) arvioimme. Intuitiivisesti kukin sana, joka on yleisempi roskapostissa kuin asiallisessa postissa, kasvattaa kaavan (43) arvoa, ja käänteisesti kukin sana, joka on yleisempi asiallisessa postissa kuin roskapostissa (kuten luultavasti sana 'algoritmi'), pienentää kaavan (43) arvoa.

Jos kaavan (43) arvo on suurempi kuin yksi, on oltava $P(\text{spam} \mid \text{viesti}) > P(\text{ham} \mid \text{viesti})$, eli voidaan arvata, että posti on roskaa ja päinvastoin.

Kaava (43) kääntyy mukavasti algoritmiksi.

ROSKAPOSTIMAISUUS(Viesti, P):

```

Osama = P.spam / P.ham      // P.spam + P.ham = 1
for each Sana in Viesti
    Osama = Osama * P.sana_spam(Sana) / P.sana_ham(Sana)
return(Osama/(1+Osama))

```

Algoritmi 9. Naivi-Bayes -malliin perustuva roskapostisuodatin, joka saa argumenttina sähköpostiviestin ja todennäköisyysmallin P ja palauttaa todennäköisyysarvon, joku kuvaa sitä, kuinka todennäköisesti viesti on roskapostia. (Lyhenne Osama = osamäärä.)

Algoritmi 9 palauttaa arvon $x/(1+x)$, missä x on yhtä kuin kaava (43). Tämä voi aluksi vaikuttaa hämärältä, mutta selkenee, kun sijoitetaan kaava (43) x :n paikalle:

$$(45) \quad \frac{\frac{P(\text{spam} \mid \text{viesti})}{P(\text{ham} \mid \text{viesti})}}{1 + \frac{P(\text{spam} \mid \text{viesti})}{P(\text{ham} \mid \text{viesti})}} = \frac{P(\text{spam} \mid \text{viesti})}{P(\text{ham} \mid \text{viesti}) + P(\text{spam} \mid \text{viesti})},$$

missä huomataan oikean puolen nimittäjän olevan muotoa $P(A \mid Z) + P(\neg A \mid Z) = 1$ (yhtäsuuruus pätee kaavan (7) ja sitä seuraavan huomautuksen perusteella). Nimittäjä häviää siis kaavasta ja jäljelle jää haluttu termi $P(\text{spam} \mid \text{viesti})$.

EPÄSYMMETRINEN KUSTANNUSFUNKTIO

Roskapostisuodattimeen liittyy vielä tärkeä seikka, joka liittyy suodattimen toimintaan riippuen todennäköisyydestä $P(\text{spam} \mid \text{Viesti})$. On nimittäin huomioitava, että roskapostisuodattimen tekemiin virheisiin liittyvä "kustannus" ei ole symmetrinen: on haitallisempaa luokitella asiallinen viesti roskapostiksi kuin roskapostiviesti asialliseksi viestiksi.

Siksi luokittelupäätöstä ei kannatakaan tehdä suoraan kriteerillä $P(\text{spam} \mid \text{viesti}) > 0.5$, vaan jollakin ykköstä lähempänä olevalla kynnyksarvolla $P(\text{spam} \mid \text{viesti}) > \alpha$, eli viestin täytyy olla roskapostia vähintään todennäköisyydellä α , jotta se ohjattaisiin roskapostikansioon. Näin asiallisiksi luokiteltujen roskapostien määrä kasvaa, mutta roskapostiksi luokiteltujen asiallisten viestin määrä vähenee.

Todennäköisyysmallinnukseen aiheeseen liittyviä oppimistavoitteita (Huom: vaalennettuja kohtia ei ole vielä tässä vaiheessa käsitelty, eikä tietenkään kohdassa syventävää oppimistavoitteita –mainittavia aiheita):

Esitiedot

- hallitsee todennäköisyyslaskennan peruskäsitteet: todennäköisyys, muuttuja, jne (lukiomatematiikka)
- tuntee luonnollisten neuroverkkojen peruskäsitteitä: neuroni, verkko, aktivaatio (lukion biologia)

Lähestyy oppimistavoitetta

- osaa soveltaa todennäköisyyslaskennan peruskaavoja yksinkertaisissa tilanteissa (esim. Bayesin kaava)
- osaa arvioida yksinkertaisia todennäköisyysarvoja satunnaisotoksesta
- osaa selittää koneoppimisen eri lajien erot (ohjattu vs ohjaamaton oppiminen) sekä peruskäsitteitä (opetusjoukko, testijoukko)
- osaa kuvailla joitakin neuroverkkotyyppisiä (eteenpäin syöttävä, takaisinkytketty, jne.)

Saavuttaa oppimistavoitteet

- osaa esittää ongelmanratkaisutilanteen todennäköisyysmallina (Bayes-verkkona)
- osaa generoida dataa Bayes-verkosta
- osaa tehdä pienimuotoista todennäköisyyspäättelyä joko eksaktisti tai stokastista approksimaatiota soveltaen
- osaa toteuttaa yksinkertaisia luokittelualgoritmeja kuten naivi Bayes- ja lähimmän naapurin luokitin
- tuntee vähintään kolmen eri neuroverkkotyyppiä edustavan neuroverkon toimintaperiaatteet

Syventää oppimistavoitteita

- osaa päätellä annetun Bayes-verkon implikoimat riippumattomuudet (*Todennäköisyysmallinnus*)
- osaa toteuttaa tehokkaan eksaktin päättelyalgoritmin Bayes-verkoille
- osaa oppia Bayes-verkon rakenteen datasta (*Todennäköisyysmallinnus*)
- osaa toteuttaa monia eri koneoppimis- ja neuroverkkomenetelmiä ja soveltaa niitä luontevasti eri tilanteissa (*Introduction to Machine Learning, Unsupervised Machine Learning, Supervised Machine Learning*)

8. Koneoppiminen

Koneoppimisen avulla on saavutettu viime aikoina läpimurtoja monella alalla (itsestään ajavat autot, roskapostin suodatus, tiedonhaku, tekstintunnistus, suosittelu, konekäännös, jne.).

Koneoppimisen tavoitteena on automatisoida oppiminen, jolloin jonkin ongelman ratkaisua ei tarvitse syöttää valmiina tietokoneeseen vaan sen voi tuottaa

automaattisesti opetusaineistosta. Tämä säästää käsityötä (vaikkapa roskapostisuodattimen parametrien säätämisessä) ja etenkin jos saatavilla on suuri määrä opetusaineistoa, yleensä johtaa parempaa tulokseen kuin käsin koodattu ratkaisu.

Koneoppimisessa keskeisiä käsitteitä ovat **tehtävä** (ongelma ja tavoitteen kuvaus, sallitut ratkaisut), **hyvyysmitta** (ratkaisun hyvyys), **esimerkit/data** (aineisto, josta opitaan). Usein datan esittäminen sopivassa muodossa on haasteellista, vrt. kuvantunnistuksessa käytettävät piirteet.

Koneoppimisen lajeja ovat:

1. **Ohjattu (supervised) oppiminen**: Esimerkit ovat muotoa (x,y) , missä x on syöte ja y haluttu tuloste (esim **luokittelu** tai ennuste jatkuvasta arvosta).
2. **Ohjaamaton (unsupervised) oppiminen**: Esimerkit ovat muotoa x , missä tavoitteena on luoda datasta esitys, joka on helpommin ymmärrettävissä tai muutoin hyödynnettävissä kuin raakadata. Tyypillinen esimerkki: klusterointi (eli ryvästys).
3. **Vahvistusoppiminen (tai palauteoppiminen; reinforcement learning)**: Palaute on epäsuorempaa kuin ohjatussa oppimisessa.

Tällä kurssilla käsiteltiin eniten ohjattua oppimista, erityisesti **luokittelua**. Luokitteluongelmassa syötteistä x on yritettävä päätellä niitä vastaava luokka y . Mahdollisia luokkia voi olla kaksi tai useampia. Esimerkkeinä eli **opetusjoukkona** on (x,y) -pareja. Niiden avulla muodostettava **luokitinta** voi soveltaa uusiin x -syötteisiin. **Luokitteluvirhe** mitataan soveltamalla luokitinta joukkoon testiesimerkkejä, eli **testijoukkoon**, ja laskemalla virheiden suhteellinen osuus, joka usein ilmoitetaan prosentteina.

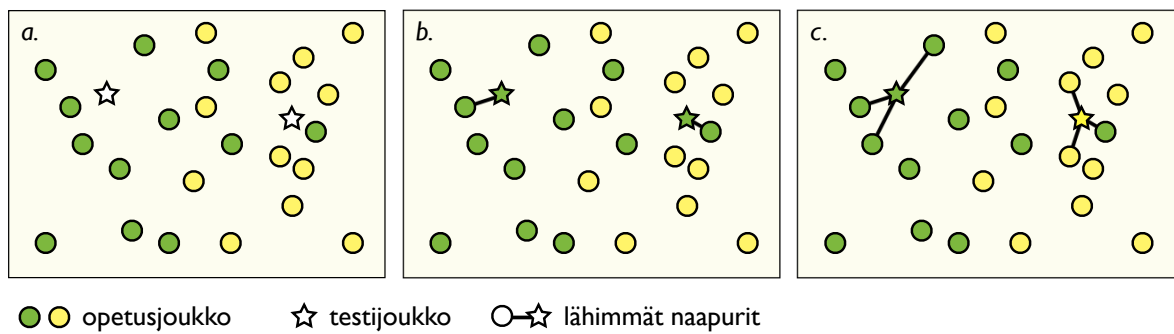
Yksi esimerkki luokittelutehtävästä on käsin piirrettyjen numeroiden tunnistaminen. Tällöin syötteenä on kuva ja oikea luokka on numero, jota kuva esittää, kuva 8. Hyvyysmitta on tällöin luokitteluvirhe, eli väärin luokiteltujen numeroiden osuus testijoukossa. Luokittimen muodostamiseen eli oppimiseen käytetään suurta määrää eri numeroita esittäviä esimerkkejä, joihin on liitetty oikea luokka.



Kuva 8. Esimerkkejä 28×28 -kokoisista bittikarttakuvista. Kuvien yllä oikeat luokat (numerot). Kuvia voi käyttää 784 -elementtiä pitkinä syötevektoreina, jolloin tavoite voi olla luokitella ne eri numeroiksi.

Seuraavaksi tutustumme kahteen eri luokittimeen: lähimmän naapurin luokittimeen ja naivi-Bayes -luokittimeen.

Lähimmän naapurin luokitin etsii opetusaineistosta sen syötevektorin x^{train} , joka on lähinnä uutta syötevektoria x^{test} ja palauttaa edellistä vastaavan luokan y^{train} . **k-**



Kuva 9. k -lähimmän naapurin luokitin. a) opetusjoukko, jonka alkiot kuuluvat kahteen luokkaan, vihreä ja keltainen, sekä testijoukko, jonka alkoita ei vielä ole luokiteltu. b) Lähimmän naapurin luokitin: kukin testialkio on luokiteltu sitä lähimmän opetusjoukon alkion mukaan. c) k -lähimmän naapurin luokitin, $k=3$: kukin alkio on luokiteltu k lähimmän naapurin mukaan. Huomaa, että toisen testiesimerkin kohdalla on luokka valittu äänestämällä eli valitsemalla useimmin esiintyvä luokka.

lähimmän naapurin luokitin hyödyntää useampaa lähintä naapuria, joiden kesken suoritetaan äänestys: eniten ääniä saanut luokka-arvo valitaan luokittimen tulosteeksi eli arvaukseksi luokasta, kuva 9.

Yksi kiinnostava ongelma k -lähimmän naapurin luokitinta (ja joitain muita etäisyyksiin perustuvaa menetelmää) soveltaessa on, miten määritellä alkoiden väliset **etäisyydet**. Kuvan 8 esimerkissä on sovellettu euklidista etäisyyttä, mutta tämä ei aina on viisainta tai edes mahdollista, koska joskus esimerkkejä kuvaava x voi olla vaikkapa merkkijono tai muu esitys, jonka kohdalla euklidinen etäisyys ei ole mielekäs käsite. Etäisyys onkin syytä määritellä tapauskohtaisesti. Luennolla käsitellyssä numerontunnistustehtävässä etäisyytenä käytettiin mustavalkoisten bittikarttakuvien pikselien eroavaisuuksien laskemista, minkä todettiin olevan herkkä esim. kuvan siirtymiselle paikaltaan.

Erona roskapostisuodattimessa käytettyyn naivi-Bayes -luokittimeen, nyt käsitellään versiota, joka voi luokitella esimerkkejä useampaan kuin kahteen luokkaan.

Algoritmi 9 toteuttaa naivi-Bayes -luokittimen tapauksessa, jossa luokkia on yhdeksän, 0, ..., 9, ja piirremuuttujia 784. Syötteenä x on 784 elementtiä pitkä vektori, jossa on lueteltuna rivi kerrallaan kaikki 28×28 -kokoisen kuvan pikselit ja kuvaa vastaava luokka kertoo, mitä numeroa se esittää, kuten kuvassa 8.

Tässäkin tapauksessa eri piirremuuttujien arvojen oletetaan olevan riippumattomia toisistaan annettuna luokkamuuttujan arvo. Jokaisen luokan kohdalla lasketaan todennäköisyys

$$(46) \quad P(\text{Luokka} = i, X = x) = P(\text{Luokka} = i) \prod_{j=1}^p P(X_j = x_j \mid \text{Luokka} = i),$$

missä X on p elementtiä pitkä syötevektori (edellä $p = 784$) ja X_j ovat sen elementit, kun $j \in \{1, \dots, 784\}$. Tulossa esiintyvät todennäköisyydet opitaan opetusjoukosta laskemalla kuinka usein kukin piirre saa eri arvonsa tiettyä luokkaa edustavissa esimerkeissä, samaan tapaan kuin roskapostisuodattimen parametrit.

Priorijakaumana algoritmissa 9 on käytetty tasajakaumaa, jolla $P(\text{Luokka} = i) = 0.1$, kaikilla $i \in \{0, \dots, 9\}$.

NAIVI-BAYES-LUOKITIN(x,P):

```
y = 0
for each luokka i = 0,...,9
  P(i) = 0.1
  for each piirre j = 1,...,784
    P(i) = P(i) * P(x[j] | i)
  if P(i) > P(y) then y = i
return y
```

Algoritmi 9. Naivi-Bayes -luokitin, joka tunnistaa numeroita 0,...,9 syötteenään 28x28 -kokoinen bittikarttakuva $x = x[1], \dots, x[784]$ ja todennäköisyysmalli P.

Luokitin valitsee tulostekseen sen luokan, jolla todennäköisyys (46) maksimoituu. Tämä on sama kuin luokka, jonka posterioritodennäköisyys on suurin, koska laskettaessa posterioritodennäköisyyttä

$$(47) \quad P(\text{Luokka} = i \mid X = x) = \frac{P(\text{Luokka} = i, X = x)}{P(X = x)}$$

nimittäjä $P(X = x)$ on sama kaikilla luokilla, joten sillä ei ole vaikutusta siihen, millä i :n arvolla lauseke saa suurimman arvonsa. (Näin säästyttiin taas laskemasta ärsyttävää nimittäjää; ks. sivun 15 laatikko.)

9. Neuroverkot

Tekoälyn historiassa logiikan ja todennäköisyysmallinnuksen ohella yksi merkittävimmistä suuntauksista on ollut [neuroverkot](#). Kun logiikan kohdalla ajateltiin, että ihmistasoisen älykkyyden saavuttamiseksi on toteutettava ihmiselle ominainen looginen päättelykyky, neuroverkkojen kohdalla ideana on kopioida aivojen toimintamekanismeja.

Neuroverkkojen on ajateltu olevan jopa tavallisesta Turingin koneeseen perustuvan laskennan mallin vaihtoehto, jonka erityisominaisuuksia ovat luonnollisten neuroverkkojen tapaan:

- ▶ rinnakkaisuus: [neuronit](#) toimivat samanaikaisesti (yleensä asynkronisesti eli ei-tahdistetusti),
- ▶ stokastisuus: neuronien toiminta on usein osin satunnaista eli samoista syötteistä ei välttämättä seuraa sama tulos,
- ▶ massiivinen skaala: neuroneita voi olla yhteenliitettynä kymmeniä miljardeja, kuten ihmisaivoissa,

- adaptiivisuus (mukautuvuus tai oppivuus): verkon yhteydet muokkautuvat ajan mittaan.

Neuroverkkoja voidaan silti yleensä simuloida tavallisella tietokoneella.

Keinotekoisii neuroverkkoihin on kopioitu luonnollisista neuroverkoista erityisesti monen yksinkertaisen prosessorin (hermosolun tai neuronin) välinen yhteistoiminta, joka perustuu verkossa välitettäviin signaaleihin. Neuronien sisäinen toimintamekanismi ja signaalit on sen sijaan yleensä toteutettu tavoin, joka vain etäisesti muistuttaa luonnollista vastinettaan.

Useimmat keinotekoiset neuroverkot eroavat luonnollisista verkoista hyvinkin paljon, esimerkiksi seuraavilla tavoilla:

- laskenta on yleensä synkronista: kaikki neuronit suorittavat laskenta-askeleen samanaikaisesti,
- viestit ovat usein jatkuva-arvoisia erotuksena luonnollisten verkkojen binäärisiin päälle/pois-viesteihin,
- luonnollisissa neuroverkoissa paljon esiintyvää takaisinkytkentää (ks. alla) esiintyy harvemmin,
- keinotekoiset neuroverkot ovat usein paljon pienempiä kuin luonnolliset esikuvansa.

Perusneuroniin liittyy joukko syötteitä, x_1, \dots, x_n , jotka voivat olla joko binäärisiä (0,1) tai jatkuva-arvoisia. Neuronin i laskee niiden painotetun summan

$$(48) \quad z_i = \sum_{j=1}^n w_{ij} x_j,$$

missä w_{ij} ovat **painokertoimet** (tai painot), jotka ovat reaali-lukuja. Neuronin tuloste saadaan soveltamalla painotettuun summaan aktivaatiofunktioita f , joka voi olla esimerkiksi kynnysfunktio:

$$(49) \quad f(z) = \begin{cases} 0, & \text{jos } z < 0, \\ 1, & \text{muuten.} \end{cases}$$

Muitakin aktivaatiofunktioita käytetään.

Neuronin tuloste voi olla syötteenä yhdelle tai useammalle muulle neuronille, joiden tulosteet voivat edelleen olla syötteenä seuraavallille neuroneille, ja niin edelleen, riippuen siitä, minkälainen verkon rakenne on.

Neuroverkko oppii siten, että sen painokertoimet mukautuvat tuottamaan halutunlaisia syötteitä. Painojen optimointi voi olla hyvinkin vaikeaa, mikä on yksi neuroverkkojen suurimmista haittapuolista verrattuna moniin muihin koneoppimismenetelmiin.

Käsitlemme kolmenlaisia neuroverkkoja:

1. Eteenpäin syöttävä verkko, jossa laskenta etenee yhteen suuntaan eikä verkossa ole silmuikoita (eli syklejä).
2. Takaisinkytketty verkko, jossa on syklejä.
3. Itseorganisoiva kartta.

Yksi klassinen neuroverkkoalgoritmi on perseptronialgoritmi (algoritmi 10). Perseptroni on eteenpäin syöttävä neuro"verkko", joka koostuu vain yhdestä edellä kuvatuslaisesta perusneuronista. Sitä voi käyttää yksinkertaisena luokittimena, joka voi luokitella syötteitä kahteen luokkaan (0/1).

PERCEPTRON-LUOKITTELIJA(Data):

```
w = [0,...,0]    // painovektori. dimensio=n; sama kuin datan
while Luokitteluvirhe(Data, w) > 0
    x = PoimiSatunnainenEsimerkki(Data)
    z = 0
    for i = 1,...,n
        z = z + w[i] * x[i]    // kynnysfunktion argumentti
    if z ≥ 0 and y = 0:        // luokiteltiin miinus plussaksi
        w = w - x
    if z < 0 and y = 1:        // luokiteltiin plus miinukseksi
        w = w + x
    end-while
return w
```

Algoritmi 10. Perseptronialgoritmi. Algoritmi saa syötteenä (x,y) -esimerkkejä ja yrittää etsiä painokertoimet w, joilla neuronin tuloste on sama kuin oikea luokka-arvo y jokaisella esimerkisyötteellä.

Perseptronialgoritmissa jokainen virheluokitus aiheuttaa korjauksen painokertoimiin. Jos neuronin tuloste on 1 ja oikea luokka y=0, vähennetään vektori x painokerroinvektorista w. Jos taas neuronin tuloste on 0 ja oikea luokka y=1, lisätään vektori x painokerroinvektoriin w. (Vektorien yhteen- ja vähennyslaskussa lisätään tai vähennetään yksinkertaisesti jokainen yksittäinen elementti erikseen, eli vektorin w + x elementti i saadaan laskemalla yhteen luvut w_i ja x_i.)

Perseptronialgoritmin ongelmana on se, että jos kaikkia esimerkisyötteitä ei voi luokitella oikein yksinkertaisen neuronin mahdollistamilla säännöillä, jää algoritmi ikuisesti muuttelamaan painoja. Esimerkkisyötteet voi luokitella perseptronilla 100% oikein silloin, kun ne ovat **lineaarisesti eroteltavissa**. Tämä tarkoittaa sitä, että jos syötteet esitetään n-ulotteisessa avaruudessa, voidaan muodostaa niin sanottu hypertaso, joka kulkee origon kautta ja jonka toisella puolella ovat kaikki esimerkit, joilla y=0, ja toisella puolella ne, joilla y=1. Kaksiulotteisessa tapauksessa esimerkit tulee voida erottaa origon kautta kulkevalla suoralla.

Käytännössä algoritmi voidaan pysäyttää, vaikkei kaikkia esimerkkejä olisikaan saatu luokiteltua oikein, kun tietty määrä opetusaskelaita (eli algoritmin 10 silmukan läpikäyntejä) on suoritettu. Tällöin (toivottavasti) saadaan luokitin, joka luokittelee suurimman osan esimerkeistä oikein ja joka toimii hyvin myös testijoukossa.

Perseptronia vahvempi luokitin saadaan aikaan, kun monta perusneuronit kytetään yhteen [monikerroperseptroniksi](#), jossa neuronit on jaoteltu eri kerroksiin: syötekerroksen neuronit saavat tulostearvonsa suoraan syötteistä, jokainen piilokerros saa syötteensä edellisen kerroksen tulosteista, ja viimeinen kerros antaa lopulta koko verkon tulosteen. Piilokerrosten neuronien aktivaatiofunktio on usein jokin muu epälineaarinen funktio kuin kynnysfunktio. Monikerroperseptronin oppiminen on vaikeampaa kuin yhden neuronin, mutta niin sanottu takaisinvirtausalgoritmi (backpropagation) on nopea tapa löytää hyvät, muttei välttämättä parhaat mahdolliset, painokertoimet. Takaisinvirtausalgoritmi ja monikerroperseptroni eivät kuitenkaan kuulu tämän kurssin oppimistavoitteisiin, eikä niitä siten tarvitse opetella.

[Takaisinkytkettyissä neuroverkoissa](#) neuronien väliset kytkennät voivat muodostaa silmukoita (eli syklejä), joissa yhden neuronin tulosteet vaikuttavat muiden neuronien välityksellä saman neuronin syötteisiin. Tällaisten verkkojen toiminta on vaikeampi ennustaa ja niiden tuottaman monimutkaiset dynaamiset ilmiöt ovat niiden kiinnostavin ominaisuus.

Perusesimerkki takaisinkytketystä neuroverkosta on [Hopfieldin verkko](#). Siinä neuronit on kytketty toisiinsa symmetrisillä yhteyksillä, eli neuronit i ja j vaikuttavat toisiinsa samalla painokertoimella w_{ij} . Oppimisen aikana jokainen verkon neuronit saa arvon suoraan syötteen perusteella; syötevektoreissa on yhtä monta elementtiä kuin verkossa on neuroneita. Verkon painokertoimet lasketaan kaavalla

$$(4) \quad w_{ij} = \sum_{k=1}^N q_{ik} q_{jk} / N ,$$

missä $q_{ik} = +1$, jos neuronit i on päällä syötteessä k ja $q_{ik} = -1$ muuten. Tämä sääntö tarkoittaa käytännössä sitä, että lasketaan kuinka usein neuronit i ja j ovat päällä tai pois päältä samaan aikaan.

Kun Hopfieldin verkon painokertoimet on opittu kaavan (50) mukaisesti, sitä voidaan käyttää alustamalla verkon neuronit haluttuun alkutilaan ja antamalla neuronien valita uusi tila painokertoimien ja muiden neuronien perusteella. Neuronin uusi tila valitaan aktivaatiosäännöllä, joka on täsmälleen sama kuin edellä kuvatussa perusneuronissa, kaavat (48) ja (49). Kun jokainen neuronit on valinnut uuden tilansa tällä säännöllä, toistetaan sama uudelleen, jolloin uudet tilat toimivat uusina syöteinä. Samaa jatketaan, kunnes tila vakiintuu (jos vakiintuu).

Hopfieldin verkon sovelluksena voi olla vikasietoinen muisti, joka pyrkii palaamaan kohti opetusjoukossa esiintyviä tiloja, vaikka osa neuroneista saisi alkutilassa virheellisen arvon.

Toinen samantyyppinen takaisinkytketty neuroverkko on [Boltzmannin kone](#). Siinä aktivaatiosääntö on stokastinen, eli neuronit valitsevat tilansa satunnaisesti. Aktivoitumistodennäköisyydet perustuvat muiden neuronien tiloihin ja painokertoimiin.

Kolmas neuroverkkotyyppi on [itseorganisoiva kartta](#) (self-organizing map, SOM), jonka on kehittänyt suomalainen neuroverkkotutkija Teuvo Kohonen. Sen neuronit [kilpailevat](#) siitä, kuka saa aktivoitua syötteestä ja muokkaavat omaa tilaansa muistuttamaan enemmän syötettä. Myös aktivoituneen voittajaneuronin [naapureiden](#) tilaa muokataan saman suuntaisesti. Tuloksena on kartta, jossa samankaltaiset syötteet aktivoivat lähekkäin olevia neuroneita. Itseorganisoivia karttoja voidaan käyttää monenlaisen tiedon visualisointiin.

Kappaleissa 8 ja 9 (sekä kappaleissa 5,6 ja 7) on käsitelty seuraavia oppimistavoitteita:

Esitiedot

- hallitsee todennäköisyyslaskennan peruskäsitteet: todennäköisyys, muuttuja, jne (lukiomatematiikka)
- tuntee luonnollisten neuroverkkojen peruskäsitteitä: neuroni, verkko, aktivaatio (lukion biologia)

Lähestyy oppimistavoitetta

- osaa soveltaa todennäköisyyslaskennan peruskaavoja yksinkertaisissa tilanteissa (esim. Bayesin kaava)
- osaa arvioida yksinkertaisia todennäköisyysarvoja satunnaisotoksesta
- osaa selittää koneoppimisen eri lajien erot (ohjattu vs ohjaamaton oppiminen) sekä peruskäsitteitä (opetusjoukko, testijoukko)
- osaa kuvailla joitakin neuroverkkotyyppjä (eteenpäin syöttävä, takaisinkytketty, jne.)

Saavuttaa oppimistavoitteet

- osaa esittää ongelmanratkaisutilanteen todennäköisyysmallina (Bayes-verkkona)
- osaa generoida dataa Bayes-verkosta
- osaa tehdä pienimuotoista todennäköisyyspäättelyä joko eksaktisti tai stokastista approksimaatiota soveltaen
- osaa toteuttaa yksinkertaisia luokittelualgoritmeja kuten naivi Bayes- ja lähimmän naapurin luokitin
- tuntee vähintään kolmen eri neuroverkkotyyppiä edustavan neuroverkon toimintaperiaatteet

Syventää oppimistavoitteita

- osaa päätellä annetun Bayes-verkon implikoimat riippumattomuudet (*Todennäköisyysmallinnus*)

- osaa toteuttaa tehokkaan eksaktin päättelyalgoritmin Bayes-verkoille
- osaa oppia Bayes-verkon rakenteen datasta (*Todennäköisyysmallinnus*)
- osaa toteuttaa monia eri koneoppimis- ja neuroverkkomenetelmiä ja soveltaa niitä luontevasti eri tilanteissa (*Introduction to Machine Learning, Unsupervised Machine Learning, Supervised Machine Learning*)

10. Digitaalinen signaalinkäsittely

Digitaaliset kuva- ja äänisignaalit (pikselien vaaleutta tai äänen tuottamaa värähtelyä kuvaavat havainnot) esitetään joukkona lukuja, jotka kuvan tapauksessa on järjestetty niiden sijainnin mukaan kaksiulotteiseen taulukkoon. Pikselit voidaan luetella myös vektorina (eli yksiulotteisena taulukkona), jolloin ne käydään läpi esimerkiksi rivi kerrallaan. Värikuvassa jokaiselle pikselille on annettu useampi kuin yksi luku, joista jokainen vastaa yhtä "kanavaa" (tyypillisesti Red, Green ja Blue (RGB)). Havaintoarvojen voidaan myös ajatella olevan kuvassa x - ja y -koordinaattien funktio, $F(x,y)$.

Äänen tapauksessa signaalin voidaan ajatella olevan ajan funktio, $F(t)$, mutta äänisignaali on myös mahdollista esittää eri taajuuksien aaltomuotojen summana, jolloin esitysmuoto on taajuuden funktio, $F(f)$. Taajuusesitys on erityisen kätevä silloin, kun informaatiota halutaan tiivistää (tiedoston koon pienentämiseksi) ottamalla mukaan vain osa taajuuksista. Aika- ja taajuusesityksiä (*time domain, frequency domain*) vastaava signaalin esitysmuodon muunnos tulee vastaan alla kohinanpoiston yhteydessä.

Signaalinkäsittelyyn liittyviä aihealueita, joista osa liittyy vain löyhästi tekoälyyn, ovat tietokonegrafiikka, "photoshoppailu" eli kuvien keinotekoinen muokkaaminen, kuvantaminen (*imaging*), jossa kuva muodostetaan ilman valoa aistivaa kameraa (esim. magneettikuvaus), hahmontunnistus ja liikkeenkaappaus (*motion capture*). Näistä esitettiin luennoilla esimerkkejä Youtube-videoiden avulla.

Digitaaliset signaalit vaativat erilaisia tekoälymenetelmiä kuin symbolinen data. Tämä johtuu ennen kaikkea siitä, että kuvassa ja äänisignaaliassa oleva **informaation määrä** on usein valtava: 1000 x 1000 pikselin kuvassa on miljoona pikseliä ja äänitiedostoissa näytteenottotaajuuden määräämä havaintojen tiheys on tyypillisesti kymmeniä tai satoja *tuhansia* havaintoja sekunnissa (esim. CD-levyillä 44.1 kHz eli 44 100 havaintoa sekunnissa). Toinen digitaalisten signaalien ominaisuus on **kohina**, jota esiintyy käytännössä kaikissa mittauslaitteilla tallennetuissa signaaleissa. Sekä koko että kohinaisuus vaikeuttavat tekoälymenetelmien soveltamista digitaalisiin signaaleihin: ei voida esimerkiksi kovin helposti määritellä logiisten tai muunlaisten sääntöjen avulla, milloin kohde esittää ihmiskasvoja tai tiettyä henkilöä.

Kohinanpoistossa on tavoitteena poistaa kuvissa esiintyvää kohinaa. Yksi lähestymistapa perustuu ns. **allokkeisiin** (*wavelet*). Aaloke-esitys on

samantyyppinen vaihtoehtoinen esitysmuoto kuin taajuusesitys äänelle: signaali kuvataan erilaisessa muodossa, jolloin tietyt operaatiot helpottuvat. Aaloke-esityksen erityispiireitä ovat: i) "tyypillisillä" signaaleilla (esim. kuvilla) signaalia esittävän vektorin elementeistä suurin osa on nollia tai lähellä nollaa ja toistaalta pieni joukko elementtejä on itseisarvoltaan hyvin suuria; ii) kohina jakautuu tasaisesti kaikille ko. vektorin elementeille. Kohinan voi siksi suurelta osin eliminoida asettamalla elementit, joiden itseisarvo on piene nolliksi. Koska signaalin olennaisen osan (muu kuin kohina) nollasta poikkeavat elementit ovat itseisarvoltaan suuria, niihin ei kajota ja suurin osa signaalista säilyy.

Kohinanpoistoalgoritmi on esitetty pseudokoodina algoritmissa 11. Menetelmän yksityiskohtia ei tarvitse opetella: riittää sisäistää pääidea eli signaalin muuntaminen toiseen esitysmuotoon, jossa signaalin olennainen osa ja kohina voidaan helpommin erottaa. Esimerkki tähän perustuvasta kohinanpoistosta on kuvassa 10.

KOHINANPOISTO(x, t):

```
n ← length(x)
c ← dwt(x)      // aallokemuunnos (discrete wavelet transform)
for i = 1,...,n:
    if |c[i]| < t: c[i] = 0
end-for
x ← idwt(c)     // käänteismuunnos (inverse wavelet transform)
return x
```

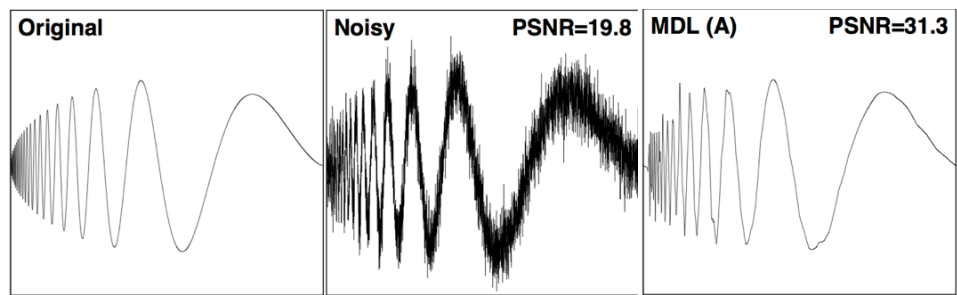
Algoritmi 11. Aallokkeisiin perustuva yksinkertainen kohinanpoisto-algoritmi. Syötteenä algoritmi saa n elementtiä pitkän signaalivektorin x ja kynnyksarvon t , jota (itseisarvoltaan) pienemmät aaloke-esityksen elementit asetetaan nolliksi.

Koon ja kohinaisuuden lisäksi signaalien esittämien hahmojen (kuvien tai äänten) tunnistamista hankaloittaa se, että signaaliin vaikuttavat itse kohteen lisäksi monet [olosuhteisiin](#) liittyvät tekijät, kuten kuvakulma, valaistus, etäisyys, kaiku, taustahäly, sekä mittalaitteiden väliset erot. Siksi kaksi eri signaalia eroavat toisistaan käytännössä aina, vaikka niiden esittämä hahmo olisikin sama.

Edellisisten huomioiden perusteella tullaan siihen johtopäätökseen, että kuvien ja äänen tunnistamisessa on olennaista pyrkiä löytämään [piirteet](#) (*feature*), jotka eivät ole herkkiä muuttumaan, kun samasta kohteesta saadaan uusi havainto. Tällaisia piirteitä sanotaan [invarianteiksi](#).

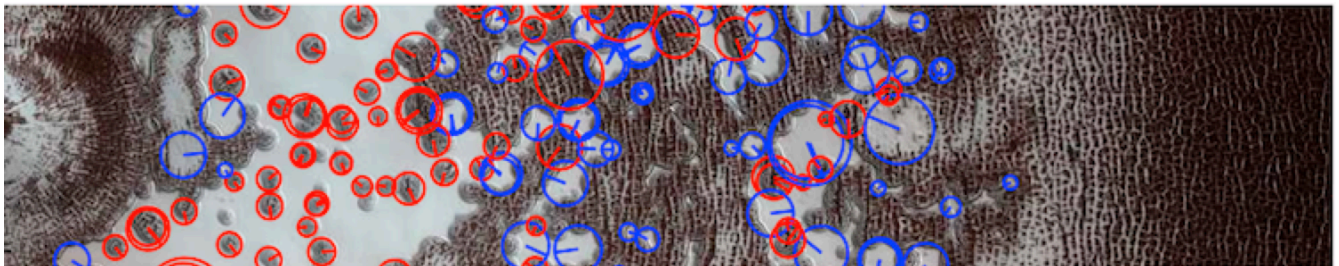
Hyviä esimerkki kuvantunnistuksessa käytettävistä piirteistä ovat ns. SIFT- ja [SURF](#)-piirteet. [SURF-piirteisiin](#) perustuva hahmontunnistus koostuu kolmesta vaiheesta:

1. [Avainpisteiden valinta](#): Etsitään kuvasta joukko pisteitä maksimoimalla ns. Hessen matriisin determinantti, joka kuvaa paikallisen intensiteetin vaihtelua. Tuloksena löydetään "blobeja". (Tästä on tarkoitus sisäistää yleinen idea. Teknisiä yksityiskohtia ei vaadita.)



Kuva 10. Esimerkki aallokkeisiin perustuvasta kohinanpoistosta. Vasemmalla ('original') on keinotekoinen kohinaton signaali. Sen vieressä ('noisy') on sen kohinainen versio, joka annetaan syötteeksi kohinanpoistoalgoritmille. Oikealla ('MDL (A)') on kohinanpoistoalgoritmin tulos, jossa kohina on onnistuttu lähes kokonaan poistamaan. Signaali on kärsinyt operaatioissa melko vähän.

2. **Piirteiden kuvaus:** Tarkastellaan avainpisteiden ympäristöä ja määritetään 64 lukuarvoa sisältävä piirrevektori (*descriptor*). Piirteellä on lisäksi skaala (koko) ja orientaatio (suunta).
3. **Piirteiden yhteensovitus:** Etsitään piirteet kahdesta kuvasta ja pyritään löytämään piirteitä, jotka toistuvat molemmissa. Piirreparit tunnustetaan piirrevektorien (euklidista) etäisyyttä vertaamalla. Tunnistusta voidaan vielä parantaa geometrisilla rajoitteilla, jos hahmon rakenteesta (esim. kasvot) on lisätietoa.



Kuva 11. Esimerkki kuvasta löydetyistä SURF-piirteistä. Jokainen ympyrä vastaa piirrettä. Ympyrän koko kuvaa piirteen skaalaa ja ympyrän säde osoittaa orientaation. Useimmat piirteet ovat kuvassa erottuvien "blobien" ("palleroiden") kohdalla. Siniset ympyrät on merkitty niiden piireiden kohdalla, joissa avainpisteen välittömässä läheisyydessä olevat pikselit ovat ympäristöään vaaleampia ja punaiset päinvastoin. *Lähde:* Nasa/Jet Propulsion Laboratory/University of Arizona.

11. Robotiikka

Robotiikka käsittää periaatteessa kaikki tekoälyn osa-alueet: robotin tulee osata aistia (nähdä, kuulla, tuntea, ...), ilmaista itseään (esim. puhe), liikkua käsitellä luonnollista kieltä, hakea tietoa, suorittaa loogista ja todennäköisyyspäättelyä, ja niin edelleen. Koneoppiminen on yksi tärkeimmistä robotiikan työkaluista: etenkin *vahvistusoppiminen* sopii hyvin robotiikan tarpeisiin. Myös tekoälyn filosofian monet kysymykset korostuvat, kun tekoäly siirtyy virtuaalisesta maailmasta fyysikaaliseen maailmaan. Vastaavasti suurin osa tekoälyn esiintymistä kulttuurissa liittyvät

nimenomaan robotteihin — merkittävänä poikkeuksena kurssin alussa keskustelun aiheena ollut HAL-tietokone.

Robotiikan erityispiirteisiin sisältyy digitaalisen signaalinkäsittelyn ongelmien lisäksi liikkumisen ja sensorihavaintojen epäluotettavuuden mukanaan tuomat "mausteet". Näihin kurssilla tutustutaan käytännön ja kantapään kautta Lego Mindstorm -robotteja käyttäen. Niihin liittyvää tietoa ja ohjeita löytyy luentokalvoista ja kurssin sivulta.

Kurssin loppuun liittyvät oppimistavoitteet:

Esitiedot

- tuntee digitaalisten signaalien esitysmuotoja (RGB-formaatti kuville, aaltomuoto äänelle)
- tunnistaa robotiikkaan (liikkumiseen, sensorihavaintoihin) liittyviä ongelmia

Lähestyy oppimistavoitetta

- osaa luetella erilaisia digitaalisen signaalinkäsittelyyn liittyviä sovelluksia (esim. hahmontunnistus, tietokonegrafiikka, liikkeenkaappaus)

Saavuttaa oppimistavoitteet

- osaa selittää vähintään yhden hahmontunnistuksen menetelmän (esim. SIFT/SURF) periaatteellisella tasolla
- osaa soveltaa jotakin valmista hahmontunnistusmenetelmää käytännössä
- osaa toteuttaa yksinkertaisia toiminnallisuuksia, kuten viivan seuraaminen, robottien avulla

Syventää oppimistavoitteita

- osaa toteuttaa digitaalisen signaalinkäsittelyn menetelmiä
- osaa toteuttaa monimutkaisia robotiikkasovelluksia (Robottiohjelmoinnin harjoitustyö)

Päivitetty 17.10.2012 T. Roos