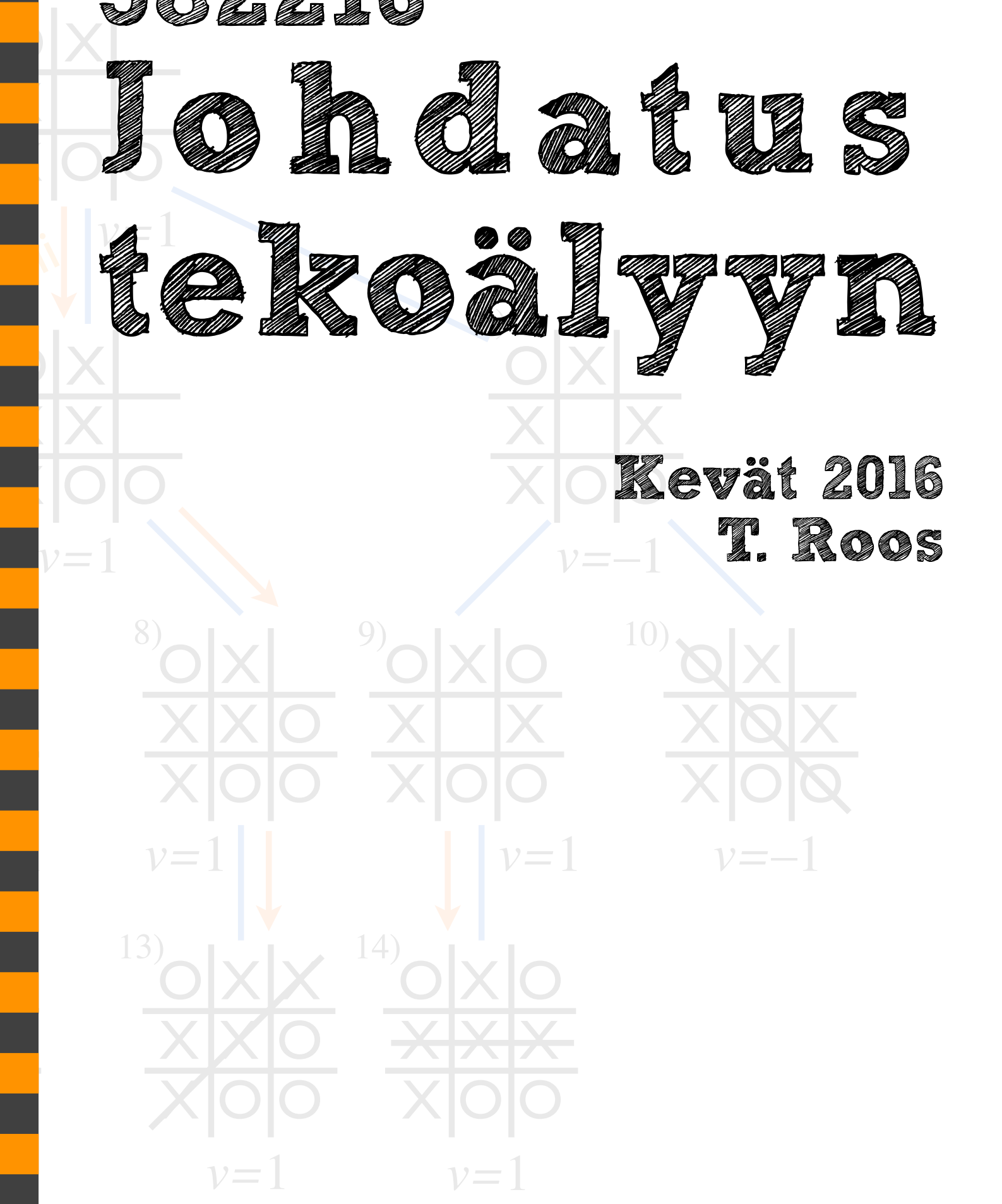
582216

Johdatus tekoälyyn

Kevät 2016

T. Roos



KURSSIKOODI: 582216
OPINTOPISTEET: 5.0
ERIKOISTUMISLINJA:
Algoritmit ja koneoppiminen
TASO: Aineopinnot
KUVAUS:
Kurssilla käydään läpi tekoälyn ongelma-alueita ja niihin liittyviä ratkaisumenetelmiä koostuen luennoista, ohjelmointitehtävistä, harjoitustehtävistä, sekä kurssikokeesta.
KURSSIKOE: 10.3.2016 klo 16–18.30,

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 luennolla opetettujen periaatteiden mukaisesti oikeasti hyödyllisiä työkaluja.

1. Mitä on tekoäly?

Tekoäly on aktiivinen tutkimusala, jonka merkitys korostuu jatkuvasti tekoälysovellusten yleistyessä. Kulttuurissa (elokuvat, kirjat, pelit, …) ja mediassa on tapana painottaa kauhuskenaarioita. Tällä kurssilla tutustutaan tekoälyn pääsuuntauksiin ja etenkin nykytutkimuksen kannalta keskeisimpiin aiheisiin. Niitä tarkastellessa huomataan, että akateemisen tutkimuksen pääpaino on suosittelujärjestelmien tai autonomisten ajoneuvojen kaltaisten yksittäisten käytännön ongelmien ratkaisemisessa. Tietoisuuden kaltaisten filosofisten ongelmien pohdinta on jätetty enimmäkseen hedelmättömänä taka-alalle.

Tekoälyn – kuten epäilemättä monen muunkin tutkimusalan – historiassa on vuosien mittaan nähty erilaisten lähestymistapojen aaltoliikettä, jossa yhteen suuntaukseen on suhtauduttu varauksettoman optimistisesti, minkä jälkeen on törmätty ylitsepääsemättömiin ongelmiin ja mielenkiinto on siirtynyt seuraavaan suuntaukseen. Esimerkiksi 1960-luvulla uskottiin vahvasti neuroverkkojen mahdollisuuksiin, mutta kiinnostus lopahti yht'äkkiä vuosikymmenen lopussa, kun törmättiin tiettyihin mahdottomuustuloksiin. Tätä seurasi niin sanottu "tekoälytalvi", kun rahahanat suljettiin. 1980-luvulla uskottiin jälleen kaikkien tekoälyongelmien ratkeavan muutaman vuoden kuluessa, tällä kertaa logiikkaan pohjautuvilla menetelmillä. Kun ylioptimistiset odotukset eivät tälläkään kertaa toteutuneet, koitti toinen tekoälytalvi 1980–1990-lukujen vaihteessa. Nykyään sekä logiikkaan että neuroverkkoihin perustuvat menetelmät ovat jälleen aktiivisen tutkimuksen kohteena ja etenkin syvät neuroverkot ovat saaneet 2010-luvulla valtavasti huomiota.

Tekoälytutkimus on perinteisesti jaoteltu kahteen luokkaan GOFAI ("Good Old-Fashioned AI") käsittää etenkin logiikkapohjaisia menetelmiä ja niiden elävästä elämästä etäällä olevia sovelluksia ns. leluongelmissa. "Modernilla AI:lla" viitataan nykyään vallalla olevaan tapaan ratkoa yksittäisiä käytännön ongelmia, joihin puhtaasti logiikkaan pohjaavat menetelmät sopivat heikosti johtuen havaintojen epätarkkuudesta, merkitysten epämääräisyydestä ja ongelmien monimutkaisuudesta.

Tekoälyn akselit

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.

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 jaotteluita

2. Etsintä ongelmanratkaisuna

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

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

```
ETSINTÄ(Alkusolmu)                % yleinen etsintäalgoritmi
    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")
```

Algoritmi 1. Yleinen etsintäalgoritmin runko.

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 syvyyssuuntainen 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.

```
LISÄÄ(Naapurilista, Solmulista)    % pino
    Uudet = Naapurilista – Käsitellyt
    return(PERÄKKÄIN(Uudet, Solmulista))
```

Algoritmi 3. LISÄÄ-funktion syvyyssuuntaisen etsinnän tuottava versio.

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.

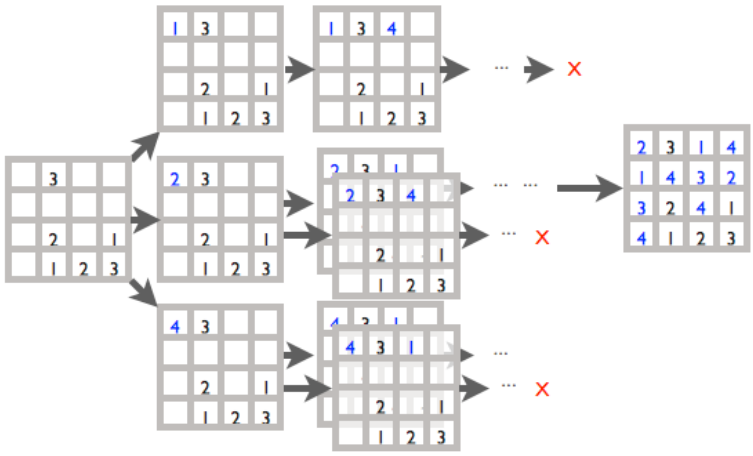
Kummassakin LISÄÄ-funktion versiossa ensimmäisellä rivillä poistetaan listasta sellaiset solmut, jotka on jo käsitelty (Käsitellyt -lista). Näin vältetään syklit eli etsinnän palaaminen takaisin jo aiemmin käsiteltyihin solmuihin. Muistin säästämiseksi leveyssuuntaisessa haussa lisäämättä jätetään myös 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, poispäin juuresta.

Ongelmanratkaisu hakualgoritmien avulla

Hakualgoritmeja voi käyttää ongelmanratkaisuun, kunhan ongelman saa muotoiltua sopivaan muotoon. Lähetyssaarnaajien 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 syvyyssuuntaista hakua, kun tiloja ovat kelvolliset osittaiset ratkaisut ja siirtymät vastaavat yhden numeron lisäämistä.



Kuva 2. Sudoku-etsintäavaruus puurakenteena.

Heuristinen haku: A-algoritmi*

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. Paras-ensin–versiossa solmulistassa olevat ja sinne lisättävät alkiot järjestetään lisäysvaiheessa niiden kustannusarvion mukaisesti. Huomaa, että toisin kuin leveys- ja syvyyssuuntaisessa 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.)

```
LISÄÄ(Naapurilista, Solmulista)    % prioriteettijono
        return(JÄRJESTÄ(Naapurilista, Solmulista))
```

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

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

$$f(N) = g(N) + h(N), \tag{1}$$

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 (katso myös kurssin sivu kohdasta Luento 1):

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

3. Pelit

Erilaiset pelit ovat yksi vanhimmista, edelleen tärkeimmistä ja kiinnostavimmista tekoälyn sovellusalueista. Luennolla käytiin läpi tekoälyn historiaa shakin osalta. Suurin virstanpylväs saavutettiin vuonna 1997, kun IBM:n Deep Blue voitti Garri Kasparovin kuuden ottelun turnauksessa.

Pelipuu, jossa kukin pelin tila vastaa solmua ja jokainen sallittu siirto vie eri alipuuhun, on keskeinen käsite ei-satunnaisia kahden pelaajan pelejä, kuten shakkia, pelattaessa. Puussa vuorottelevat Min- ja Max-pelaajien vuoroja vastaavat tasot. Puun lehtinä ovat pelin lopputilat, joissa suuret luvut ovat Max-pelaajan tavoitteena ja toisinpäin.

Puun sisäsolmujen arvo kertoo solmua vastaavasta pelitilanteesta jatkuvat pelin lopputuloksen arvon, jos molemmat pelaajat pelaavat optimaalisella tavalla. Esimerkiksi kuvan 3 pelissä toiseksi alimman tason (MIN) nuolilla olevan polun solmun arvo 3 kertoo, että Min-pelaajan valitessa vaihtoehdoista 3 ja 4 pienimmän, päättyy peli tilanteeseen, jonka arvo 3. Vastaavasti yhtä ylemmällä tasolla Max-pelaajan valittavana on tilanteet, joista päädytään lopputuloksiin 3 ja 1, joista Max-pelaaja valitsee ensin mainitun.

Minimax-algoritmi

Pienessä pelipuussa optimaalinen peli voidaan laskea minimax-algoritmilla, jossa kutakin vuoroa (Min tai Max) vastaa yksinkertainen rekursiivinen funktio. Kun jokaisen solmun arvo on laskettu, voidaan helposti valita optimaalinen pelisiirto kussakin solmussa.

Minimax-algoritmi ratkaisee periaatteessa optimaalisen tuloksen ja siihen johtavat pelisiirrot kaikissa kahden pelaajan peleissä, joihin ei sisälly

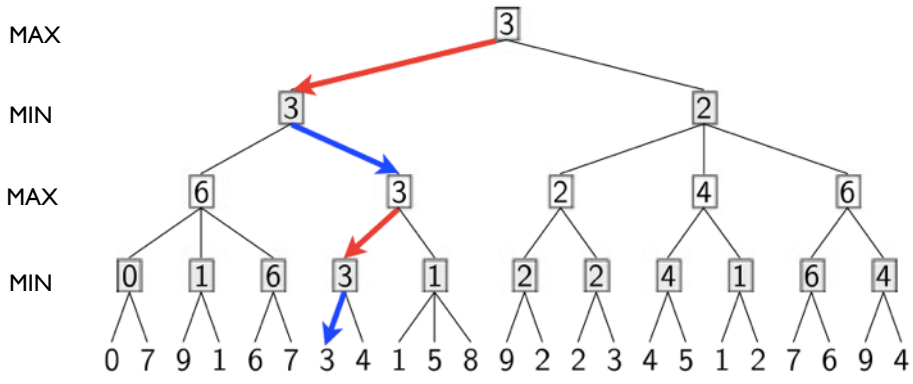
satunnaisuutta. Siten esimerkiksi shakkipelissä on teoriassa olemassa optimaalinen pelistrategia kummallekin pelaajalle, jolla musta tai valkea voittaa tai päädytään tasapeliin (kukaan ei ainakaan toistaiseksi tiedä mikä vaihtoehdoista on oikea). Käytännössä ongelmaksi muodostuu monessa pelissä pelipuun valtava koko. Näin on esimerkiksi shakin kohdalla.

```
MAX-ARVO(Solmu)                % Max-pelaajan funktio
    if LOPPUTILA(Solmu) return(ARVO(Solmu))
      v = −∞
      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 = +∞
      for each Lapsi in LAPSET(Solmu)
          v = MIN(v, MAX-ARVO(Lapsi))
    return(v)
```

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



Kuva 3. Esimerkki pelipuusta, johon on merkitty kunkin solmun arvo (luvut) ja optimaaliset pelisiirrot (nuolet). *Lähde:* Ertel, *Introduction to Artificial Intelligence*, Springer 2011.

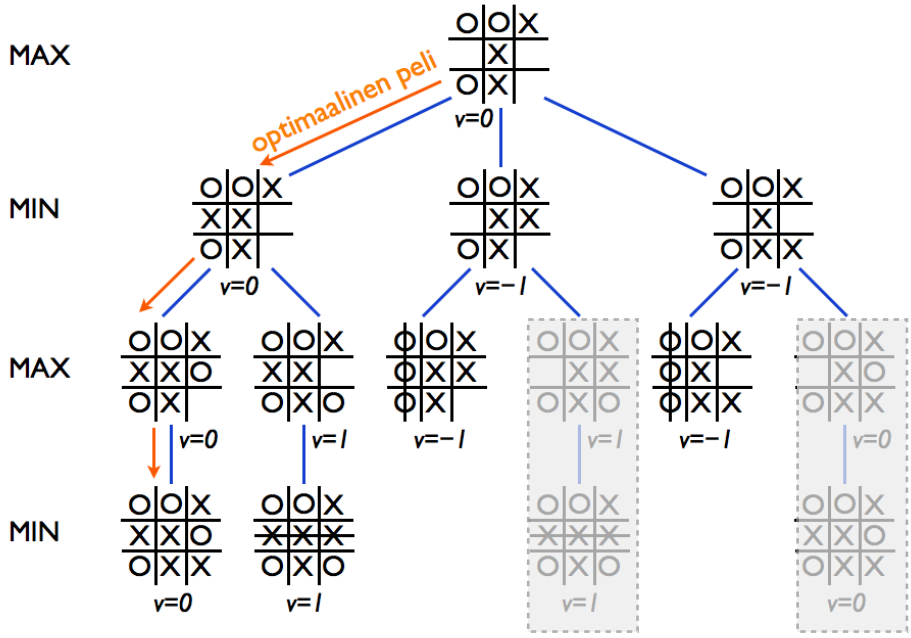
Heuristinen arviointikriteeri

Jotta pelissä, jossa peliä ei voida ratkaista minimax-algoritmin avulla pelipuun koon vuoksi, voidaan saada aikaan hyviä, mutta ei välttämättä optimaalisia, pelistrategioita, voidaan pelitilanteiden hyvyyttä arvioida erilaisilla heuristisilla arviointikriteereillä. Silloin minimax-algoritmin rekursio pysäytetään esim. kun pelipuussa on saavuttu tietylle syvyydelle vaikkei 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.

Alpha-beta–karsinta

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-algoritmilla.

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



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

Max- ja Min-pelaajien funktiot ovat nytkin (kuten minimax-algoritmissa) toistensa "peilikuvat". Huomaa etenkin if-lauseet, jotka lopettavat solmun lapsien läpikäynnin siinä vaiheessa, kun jostakin lapsisolmusta palautuva arvo ylittää beta-arvon max-solmussa tai alittaa alpha-arvon min-solmussa. Tämä tapahtuma johtaa tiettyjen alipuiden karsintaan.

```
MAX-ARVO(Solmu)                    % Max-pelaajan funktio
    if LOPPUTILA(Solmu) return(ARVO(Solmu))
    v = −∞
    for each Lapsi in LAPSET(Solmu)
      v = MAX(v, MIN-ARVO(Lapsi, α, β))
      if v ≥ β return v
      α = MAX(α, 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 = +∞
    for each Lapsi in LAPSET(Solmu)
      v = MIN(v, MAX-ARVO(Lapsi, α, β))
      if v ≤ α return v
      β = MIN(β, v)
    return(v)
```

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

Tähän aiheeseen liittyviä oppimistavoitteita:

Esitiedot

- perustietorakenteet (pino, jono)
- leveys- ja syvyyssuuntainen 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 Artificial Intelligence*, Springer 2011, kappale 6: "Search, Games and Problem Solving" (kurssikansiossa)
- Kurssin sivulla kohdassa Luento 2 löytyy aihetta käsittelevää verkkomateriaalia

4. Logiikka tekoälyssä

Logiikka oli 1980-luvulle saakka tekoälyn keskeisin lähestymistapa. Predikaattilogiikan lauseet voivat olla esimerkiksi muotoa

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

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

Päättely 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 (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:ille 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. myös sivu 4.

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

5. Päättely epävarmuuden vallitessa

5.1 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("Herra Azzip söi eilen pizzaa"). Todennäköisyydet riippuvat myös taustatiedosta ja voivat olla subjektiivisia eli riippua henkilöstä.

Todennäköisyysmalli

Tekoälysovelluksissa vaikein osa on muodostaa sopiva todennäköisyysmalli, joka kuvastaa riitävässä 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 $A,B,C,... \subseteq \Omega$, jotka ovat osajoukkoja mahdollisista maailmoista

c) todennäköisyysmalli, liittää jokaiseen alkeistapahtumaan $\omega \in \Omega$ todennäköisyyden $P(\omega) \in [0,1]$; näiden tulee summautua yhteen eli $\Sigma_{\omega \in \Omega} P(\omega) = 1$

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

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

Tapahtumista A,B,C,... 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

Kuva 5. Muistisääntö konjunktiolle: *Con hueso*, kivien kanssa.

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 ∧ B). Sekä konjunktion että disjunktion kohdalla pätee symmetrisyys: A ∧ B tarkoittaa samaa kuin B ∧ A ja vastaavasti A ∨ B tarkoittaa samaa kuin B ∨ A. Järjestysriippumattomuus koskee myös useamman tapahtuman konjunktioita ja disjunktioita, kuten A ∨ B ∨ C, jne.

Ehdollinen todennäköisyys, riippumattomuus

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

(4) $$P(A \mid B) = P(A,B) / P(B),$$

kunhan P(B) > 0. Ehdollistajana voi olla yksittäisen tapahtuman B lisäksi muitakin tapahtumia, jolloin merkitään P(A | B,C,...). 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) $$A \perp B \quad \Leftrightarrow \quad P(A \mid B) = P(A).$$

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

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.

Laskusääntöjä

Seuraavat laskusäännöt tulevat jatkossa tarpeeseen:

(7) $P(\neg A) = 1 - P(A)$ "negaatio"

(8) $P(A \vee B) = P(A) + P(B) - P(A, B)$ "inkluusio-eksluusio"

(9) $P(A) = P(A, B) + P(A, \neg B)$ "marginalisointi"

(10) $P(A_1,...,A_k) = P(A_1)\, P(A_2 \mid A_1)\, P(A_3 \mid A_1,A_2) \ldots P(A_k \mid A_1,...,A_{k-1})$ "ketjusääntö"

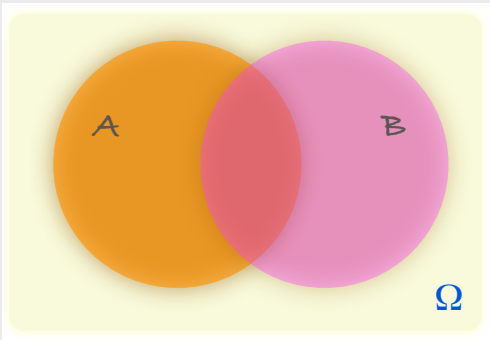
(11) $P(A \mid B) = P(A)\, P(B \mid A) / P(B)$ "Bayesin kaava"

Ketjusäännön kohdalla tapahtumia $A_1,...,A_k$ voi olla mielivaltainen määrä. Yksinkertaisimmillaan tapahtumia on kaksi, jolloin ketjusääntö sanoo $P(A,B) = P(A)\, P(B \mid A)$. Tapahtumien järjestyksellä ole ole tässä yhteydessä väliä.

Huomaa, että kaikki edellä olevat kaavat pätevät myös silloin, kun niiden ehdollistajaksi lisätään muita tapahtumia, esim. $P(\neg A \mid Z) = 1 - P(A \mid Z)$ tai $P(A \mid B,Z) = P(A \mid Z)\, P(B \mid A,Z) / P(B \mid Z)$, 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).

Satunnaismuuttujat ja niiden jakaumat

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 $\omega = (m,n) \Longrightarrow X(\omega) = m+n$ 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 satunnaisumuuttujista 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 $P_x = (P(X=2), P(X=3), P(X=4),..., P(X=12)) = (1/36, 2/36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/36, 2/36, 1/36)$, 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 merkille 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) $$P(X=x) = \sum_{y \in dom(Y)} P(X=x, Y=y) ,$$

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) $$P(X=x, Y=y) = P(X=x)\, P(Y=y \mid X=x)$$

tai lyhennysmerkinnällä:

(14) $$P(x, y) = P(x)\, P(y \mid x).$$

Bayesin kaava tilastollisessa päättelyssä

Edellä mainitulla 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) $$P(\text{tila} \mid \text{havainto}) = P(\text{tila})\, P(\text{havainto} \mid \text{tila}) / P(\text{havainto}).$$

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

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

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

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

(18) $$P(h) = \Sigma_{t \in dom(Tila)}\, P(Tila=t)\, P(h \mid Tila=t).$$

Voit vakuuttaa itsesi edelllisen 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)

5.2 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 määrääminen. Useimmiten mallinnettava tilanne on niin laaja, että kaikkien alkeistapahtumien luettelointi ja todennäköisyyden liittäminen niihin yksitellen ei ole mielekästä. Yksi tapa helpottaa todennäköisyyksien määräämistä 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 määräävät kunkin solmun ehdollisen todennäköisyysjakauman annettuna ko. solmun vanhempien arvot. Esimerkki parametriarvosta ja sen määräämästä todennäköisyydestä on

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

tai lyhynnysmerkinnöin

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

Autoesimerkissä Bayes-verkon parametrien määräämät todennäköisyydet ovat kokonaisuudessaan:

(21) $P(A) = 0.9$

(22) $P(R \mid A) = 0.9$
$P(R \mid \neg A) = 0$

(23) $p(S \mid A) = 0.95$
$P(S \mid \neg A) = 0$

(24) $P(B) = 0.95$

(25) $P(K \mid S,B) = 0.99$ $P(K \mid S,\neg B) = 0$
$P(K \mid \neg S,B) = 0$ $P(K \mid \neg S,\neg B) = 0$

(26) $P(L \mid K) = 0.99$
$P(L \mid \neg K) = 0$

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

Todennäköisyysmallin määrittely Bayes-verkon avulla

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

(27) $$P(A,R,S,B,\neg K,\neg L) = P(A)\,P(R|A)\,P(S|A)\,P(B)\,P(\neg K|S,B)\,P(\neg L|\neg K)$$
$$= 0.9 \times 0.9 \times 0.95 \times 0.95 \times (1 - 0.99) \times (1 - 0) \approx 0.00731$$

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

Edellistä kaavaa voidaan verrata aiemmin esitettyyn ketjusääntöön:

(28) $$P(A,R,S,B,\neg K,L) = P(A)\,P(R|A)\,P(S|A,R)\,P(B|A,R,S)$$
$$\times\, P(\neg K|A,R,S,B)\,P(L|A,R,S,B,\neg K)\ ,$$

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 määräämiseksi. Olemme siis saaneet määriteltyä 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:

(29) $$P(A,R,B,\neg K) = \Sigma_{\omega \in (A,R,B,\neg K)}\, P(\omega)$$
$$= P(A,R,S,B,\neg K,L)$$
$$+ P(A,R,S,B,\neg K,\neg L)$$
$$+ P(A,R,\neg S,B,\neg K,L)$$
$$+ P(A,R,\neg S,B,\neg K,\neg L)$$

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

Jos ollaan kiinnostuneita ehdollisista todennäköisyyksistä, kuten

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

voidaan soveltaa ehdollisen todennäköisyyden kaavaa (4):

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

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 = "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 reaaliluvun (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 -teknikalla. 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) $$\frac{\text{Kuinka monessa tapauksessa A pätee}}{\text{Tapausten lukumäärä}} \rightarrow P(A),$$

kun tapausten lukumäärä kasvaa rajatta.

Voit esimerkiksi kokeilla nostaa pakasta kaksi korttia (siten, 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)
$$P(A \mid B) \approx \frac{\text{Kuinka monessa tapauksessa pätee sekä A että B}}{\text{Kuinka monessa tapauksessa pätee B}}$$
,

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

Materiaalia:

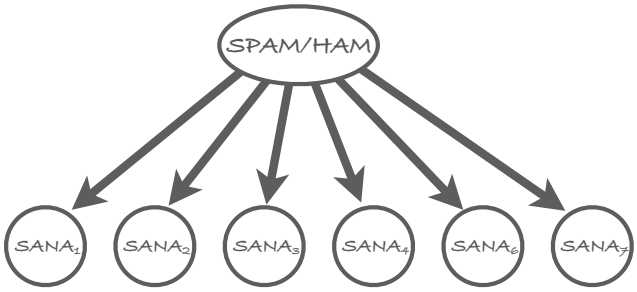
- W. Ertel, *Introduction to Artificial Intelligence,* Springer 2011, kappale 7.4 "Reasoning with Bayesian Networks".

5.3 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 -malli

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



Kuva 7. Naivi-Bayes -luokitin roskapostinsuodatukseen.

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

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

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

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

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 ≈ 8.2. Sana 'cheap' on siis roskaposteissa noin kahdeksan kertaa niin yleinen kuin asiallisissa posteissa.

Luokittimen toiminta

Olkoon nyt luokkamuuttujan jakauma sejä 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)
$$P(\text{Luokka} = \text{spam} \mid \text{Sana}_1 = \text{'buy'}, \text{Sana}_2 = \text{'cheap'}, \text{Sana}_3 = \text{'algorithm'}) ,$$

jonka lyhennämme ilman suurta sekaannuksen vaaraa

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

Ehdollinen todennäköisyys voidaan nyt kirjoittaa

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

Joka on olennaisesti Bayesin kaava – emme vain toistaiseksi halunneet jakaa osoittajan yhteistodennäköisyyttä osiin P(spam) P('buy', 'cheap', 'algorithm' | spam).

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

(39)
$$P(\text{spam}, \text{'buy'}, \text{'cheap'}, \text{'algorithm'}) = P(\text{spam})\, P(\text{'buy'} \mid \text{spam})\, P(\text{'cheap'} \mid \text{spam})\, P(\text{'algorithm'} \mid \text{spam}) ,$$

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.

Nimittäjän laskemisen välttäminen

Seuraavaksi olisi edessä kaavan (38) ("ärsyttävä") nimittäjä. Nyt kuitenkin sovellamme sivun 20 niksiä, jolla vältämme ainakin eksplisiittisesti

nimittäjän laskemisen. Niksi on siis laskea kaavan (38) asemesta roskapostin ja asiallisen postin suhteellinen todennäköisyys eli

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

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

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

$$(41) \qquad P(\text{ham}, \text{'buy'}, \text{'cheap'}, \text{'algorithm'}) = P(\text{ham})\, P(\text{'buy'} \mid \text{ham})\, P(\text{'cheap'} \mid \text{ham})\, P(\text{'algorithm'} \mid \text{ham}),$$

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:

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

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

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

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

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

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

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

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(spam | viesti) > P(ham | 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 roskaposti-suodatin, 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) $$\frac{\frac{P(\text{spam} \mid \text{viesti})}{P(\text{ham} \mid \text{viesti})}}{1 + \frac{P(\text{spam} \mid \text{viesti})}{P(\text{ham} \mid \text{viesti})}} = \frac{P(\text{spam} \mid \text{viesti})}{P(\text{ham} \mid \text{viesti}) + P(\text{spam} \mid \text{viesti})},$$

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(spam | viesti).

EPÄSYMMETRINEN KUSTANNUSFUNKTIO

Roskapostisuodattimeen liittyy vielä tärkeä seikka, joka liittyy suodattimen toimintaan riippuen todennäköisyydestä P(spam | 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(spam | viesti) > 0.5, vaan jollakin ykköstä lähempänä olevalla kynnysarvolla P(spam | viesti) > α, 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 neuroverkkotyyppejä (eteenpäin syöttävä, takaisinkytkeytyvä, 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*)

6. 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

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 muodostettavaa luokitinta voi soveltaa uusiin x-syötteisiin. Luokitteluvirhe mitataan soveltamalla luokitinta joukkoon testiesimerkkejä, eli testijoukkoon, ja laskemalla virheiden suhteellinen osuus, joka usein ilmoitetaan prosentteina.

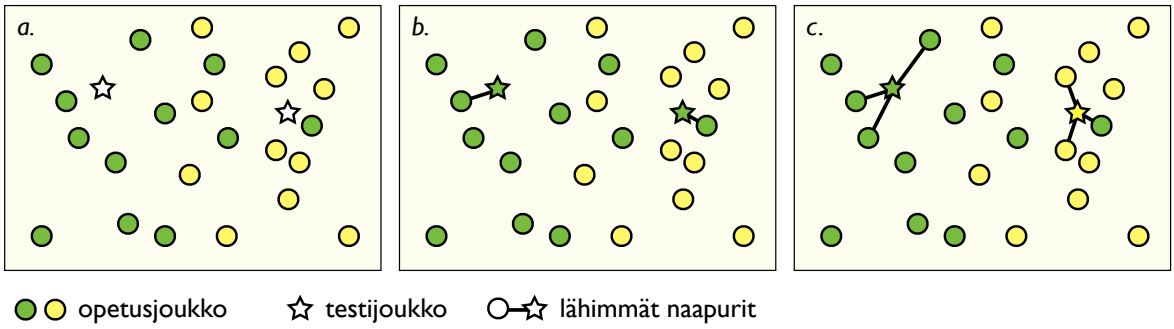
Yksi esimerkki luokittelutehtävästä on käsin piirrettyjen numeroiden tunnistaminen. Tällöin syötteenä on kuva ja oikea luokka on numero, jota kuva esittää, kuva 8. Hyvyysmitta on tällöin luokitteluvirhe, eli väärin luokiteltujen numeroiden osuus testijoukossa. Luokittimen muodostamiseen eli oppimiseen 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.

k-lähimmän naapurin luokitin

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



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.

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 alkioiden 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.

Naivi-Bayes -luokitin

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

missä X on p elementtiä pitkä syötevektori (edellä $p = 784$) ja X_j ovat sen elementit, kun $j \in \{1,...,784\}$. 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 $i \in \{0,\ldots,9\}$.

```
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],...,x[784] ja todennäköisyysmalli P.

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

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

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 20 laatikko.)

7. 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.

Keinotekoisiin 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.

Painokertoimet ja aktivaatiofunktio

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

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

missä w_{ij} ovat painokertoimet (tai painot), jotka ovat reaalilukuja. Neuronin tuloste saadaan soveltamalla painotettuun summaan aktivaatiofunktiota f, joka voi olla esimerkiksi kynnysfunktio:

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

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.

Käsittelemme kolmenlaisia neuroverkkoja:

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

Perseptronialgoritmi

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

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.)

```
PERCEPTRON-LUOKITTELIJA(Data):
    w = [0,...,0]      // painovektori. dimensio=n; sama kuin datan
    while Luokitteluvirhe(Data, w) > 0
        (x,y) = 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                // vektorien erotus
        if z < 0 and y = 1:          // luokiteltiin plus miinukseksi
            w = w + x                // vektorien summa
    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 esimerkkisyötteellä.

Perseptronialgoritmin ongelmana on se, että jos kaikkia esimerkkisyötteitä ei voi luokitella oikein yksinkertaisen neuronin mahdollistamilla säännöillä, jää algoritmi ikuisesti muuttelemaan 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 opetusaskeleita (eli algoritmin 10 silmukan läpikäyntejä) on suoritettu. Tällöin (toivottavasti) saadaan luokitin, joka luokittelee suurimman osan esimerkeistä oikein ja joka toimii hyvin myös testijoukossa.

Monikerrosperseptroni

Perseptronia vahvempi luokitin saadaan aikaan, kun monta perusneuronia kytketään yhteen monikerrosperseptroniksi, 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. Monikerrosperseptronin oppiminen on vaikeampaa kuin yhden neuronin, mutta niin sanottu takaisinvirtausalgoritmi (*backpropagation*) on nopea tapa löytää hyvät, muttei välttämättä parhaat mahdolliset, painokertoimet. Takaisinvirtausalgoritmi ja monikerrosperseptroni eivät kuitenkaan kuulu tämän kurssin oppimistavoitteisiin, eikä niitä siten tarvitse opetella.

Takaisinkytkeytyvät neuroverkot

Takaisinkytkeytyvissä 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 takaisinkytkeytyvästä neuroverkosta on Hopfieldin verkko. Siinä neuronit on kytkeytty toisiinsa symmetrisillä yhteyksillä, eli neuronit i ja j vaikuttavat toisiinsa samalla painokertoimella w_{ij}. Oppimisen aikana jokainen verkon neuroni saa arvon suoraan syötteen perusteella; syötevektoreissa on yhtä monta elementtiä kuin verkossa on neuroneita. Verkon painokertoimet lasketaan kaavalla

$$w_{ij} = \sum_{k=1}^{N} q_{ik}\, q_{jk} / N \,, \tag{50}$$

missä $q_{ik} = +1$, jos neuroni 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 neuroni on valinnut uuden tilansa tällä säännöllä, toistetaan sama uudelleen, jolloin uudet tilat toimivat uusina syötteinä. Samaa jatketaan, kunnes tila vakiintuu (jos vakiintuu).

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

Toinen samantyyppinen takaisinkytkeytyvä neuroverkko on Boltzmannin kone. Siinä aktivaatiosääntö on stokastinen, eli neuronit valitsevat tilansa satunnaisesti. Aktivoitumistodennäköisyydet perustuvat muiden neuronien tiloihin ja painokertoimiin.

Itseorganisoiva kartta

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 neuroverkkotyyppejä (eteenpäin syöttävä, takaisinkytkeytyvä, 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. Digitaalinen signaalinkäsittely

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.

Digitaalisten signaalien esitysmuodot

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 taajuisten 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 taajuksista. Aika- ja taajuusesityksiä (*time domain, frequency domain*) vastaava signaalin esitysmyodon muunnos tulee vastaan alla kohinanpoiston yhteydessä.

Digitaaliset signaalit vaativat erilaisia tekoälymenetelmiä kuin symbolinen data. Tämä johtuu ennen kaikkea siitä, että kuvassa ja äänisignaalissa 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.

Hahmontunnistus, invariantit piirteet

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. Luentokalvoissa näytetään esimerkki afgaanitytöstä, joka voitiin tunnistaa vielä vuosikymmenien jälkeen silmän iiriksen perusteella. Tässä tapauksessa iiris on invariantti piirre, joka säilyy myös iän myötä. Käytännössä hahmontunnistuksessa käytetään paljon yleisempiä piirteitä kuin silmän iiris.

Hyviä esimerkki hahmontunnistuksessa käytettävistä piirteistä ovat ns. SIFT- ja SURF-piirteet. SURF-piirteisiin perustuva hahmontunnistus koostuu kolmesta vaiheesta:

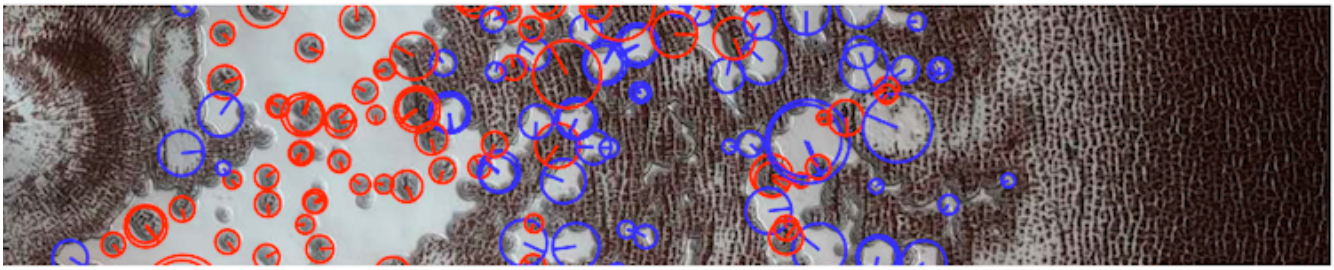
1. Avainpisteiden valinta: Etsitään kuvasta joukko pisteitä maksimoimalla ns. Hessen matriisin determinantti, joka kuvaa paikallisen intensiteetin vaihtelua. Tuloksena löydetään "blobeja". (Tästä on tarkoitus sisäistää yleinen idea. Teknisiä yksityiskohtia ei vaadita.)
2. Piirteiden kuvaus: Tarkastellaan 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 tunnistetaan piirrevektorien (euklidista) etäisyyttä vertaamalla. Tunnistusta voidaan vielä parantaa geometrisilla rajoitteilla, jos hahmon rakenteesta (esim. kasvot) on lisätietoa.

Kuvassa 10 on esimerkki löydetyistä avainpisteistä. Lisää esimerkkejä löytyy luentokalvoista.

9. Luonnollisen kielen käsittely

Tekoälysovellukset, jotka liittyvät ihmisten väliseen kommunikaatioon tai jotka kommunikoivat itse ihmisten kanssa, ovat yleensä tekemisissä luonnollisen kielen kanssa. Luonnolliseen kieleen liittyy ominaisuuksia, jotka on syytä ottaa huomioon tällaisissa sovelluksissa:

- vaikka kieliaineistoissa onkin periaatteessa lineaarinen, yksiulotteinen järjestys, jossa teksti alkaa tietystä kohdasta ja etenee sana sanalta kohti



Kuva 10. 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.

loppua, on tekstissä myös hierarkkinen teksti–kappale–lause–sana-rakenne, jossa jokainen yksikkö muodostaa oman kokonaisuutensa,

- kielioppi rajoittaa mahdollisia (tai sallittuja) ilmaisuja, mutta jättää silti sijaa moniselitteisyydelle (toisin kuin keinotekoisissa kielissä, kuten ohjelmointikielissä),

- verrattuna digitaalisiin signaaleihin, kuten ääneen tai kuvaan, kieliaineistot ovat paljon lähempänä semantiikkaa: tuolin idea liittyy kiinteästi sanaan ”tuoli”, mutta missä tahansa tuolia esittävässä kuvassa on huomattavasti muutakin informaatiota, kuten väri, koko, materiaali sekä kuvakulma, valaistus, jne.

Luonnollisen kielen käsittelyssä sovelletaan monia teoreettisen tietojenkäsittelytieteen käsitteitä, jotka sopivat myös keinotekoisten (formaalien) kielten käsittelyyn. Formaali kielioppi tarkoittaa yleisesti merkkijonojen joukkoa. Esimerkiksi sallittujen matemaattisten lausekkeiden joukkoon kuuluu merkkijono ”(1+2) × 6 – 3”, mutta ei merkkijono ”1+(+ ×5(”.

Yksi formaalien kielien luokka on ns. yhteydettömät kielet (*context-free language*). Yhteydettömän kielen määrittelevä kielioppi koostuu joukosta muotoa $V \longrightarrow w$ olevia sääntöjä, missä V on välike ja w on jono välikkeitä tai päätesymboleja. Sääntöjä voi soveltaa mihin tahansa välikkeeseen riippumatta sitä ympäröivistä välikkeistä.

Esimerkki valaissee asiaa. Seuraavassa on yksinkertainen yhteydettömän kieliopin määrittelevä sääntöjoukko.

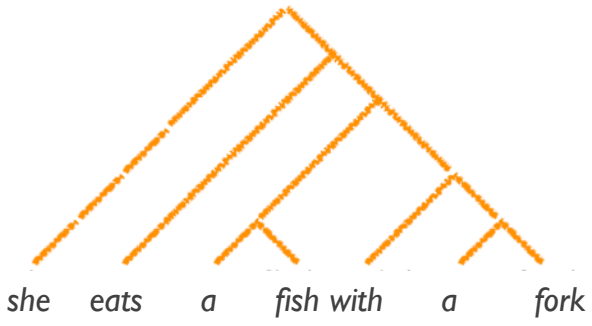
$S \longrightarrow SS$
$S \longrightarrow$
$S \longrightarrow (S)$
$S \longrightarrow [S]$

S : aloitussymboli
( ) [ ] : päätesymbolit

Näiden kielioppisääntöjen määrittelevään kieleen kuuluvat täsmälleen ne merkkijonot, jotka voidaan tuottaa aloitussymbolista S soveltamalla yllä mainittuja neljää sääntöä mielivaltainen määrä kertoja siten, että lopputuloksena on päätesymboleista koostuva merkkijono.

Esimerkiksi merkkijono ”()” voidaan tuottaa soveltamalla aloitussymboliin S kolmatta sääntöä $S \longrightarrow (S)$ ja sen jälkeen toista sääntöä $S \longrightarrow$, joka korvaa sulkeiden välissä olevan aloitussymbolin tyhjällä merkkijonolla. Merkkijono ”(())[]” puolestaan saadaan soveltamalla aloitussymboliin ensimmäistä sääntöä $S \longrightarrow SS$ ja sen tuloksena saatavan merkkijonon ensimmäiseen S-symboliin kaksi kertaa kolmatta sääntöä ja kerran toista sääntöä, ja toiseen S-symboliin kerran neljättä sääntöä ja kerran toista sääntöä. Merkkijonoa ”(()[])”, jossa sulkeet eivät ole tasapainossa, ei sen sijaan saa aikaiseksi millään ko. kielioppiin kuuluvilla säännöillä eli se ei kuulu kieleen.

Yhteydettömän kielen tunnistamiseksi on olemassa tehokkaita jäsennysmenetelmiä. Jäsennys tarkoittaa lauseen (eli sanojen muodostaman jonon) jakamista lauseenjäseniin. Jäsennyksen tulos voidaan esittää jäsennyspuuna, joka tuo esiin lauseen hierarkkisen rakenteen.



Kuva 11. Lauseen ”she eats a fish with a fork” yksi mahdollinen jäsennyspuu.

Kuvan 11 jäsennyspuu vastaa hiukan epätodennäköistä, mutta mahdollista tulkintaa, jossa syömisen kohteena on ”a fish with a fork” eli kala, jolla on haarukka. Todennäköisempi tulkinta on tietenkin se, että ”with a fork” sanat viittaavat ilmaukseen ”eats a fish”, eli kala syödään haarukalla. Luentokalvoista löytyy jälkimmäistä tulkintaa vastaava jäsennyspuu.

CYK-algoritmi

Cooke–Younger–Kasami-algoritmi on ajassa $O(n^3)$ toimiva jäsennysalgoritmi yhteydettömille kielille. Aikavaativuudessa n tarkoittaa lauseen sanojen määrää.

CYK-algoritmi soveltaa dynaamisen ohjelmoinnin periaatetta jäsennystaulukon täyttämiseen solu kerrallaan. Taulukosta lasketaan kolmion muotoinen alue, joka esitetään yleensä joko neliön muotoisen alueen oikeana yläkulmana tai vasempana alakulmana (ks. esimerkki alla).

```
CYK-jäsennys(lause, kielioppi G):
    alusta tyhjä taulukko T      // dimensio=n×n
    lisää lauseen sanat i=1,…,n soluihin T[i,i]
    for pituus = 1…n:            // käsiteltävän osan pituus
        for i = 1…(n–pituus+1):  // alkaa sanasta i
            j = i+pituus–1       // loppuu sanaan j
            jatka = True
            while jatka:
                jatka = False
                if V ⟶ X ∈ G and X ∈ T[i,j] and V ∉ T[i,j]:
                    lisää V soluun T[i,j]
                    jatka = True
                for k = i…(j–1):        // katkaisukohta
                    if V ⟶ X Y ∈ G and X ∈ T[i,k]
                        and Y ∈ T[k+1,j] and V ∉ T[i,j]:
                            lisää V soluun T[i,j]
                            jatka = True
                end-for
            end-while
        end-for
    end-for
```

Algoritmi 11. CYK-lauseenjäsennys yhteydettömälle kieliopille.

Seuraavassa esimerkissä jäsennetään lause ”Robots fish fish today”, kun kielioppi sisältää säännöt:

$S \longrightarrow NP\ VP$
$VP \longrightarrow V\ NP$
$VP \longrightarrow VP\ ADV$
$VP \longrightarrow V$
$NP \longrightarrow N$
$V \longrightarrow fish$
$N \longrightarrow Robots$
$N \longrightarrow fish$
$ADV \longrightarrow today$

Taulukon solut numeroidaan lukuparein (i,j), missä i kertoo mistä sanasta solua vastaava lauseen osa alkaa ja j kertoo mihin sanaan se päättyy. Esimerkiksi solu (2,4) vastaa lauseen osaa, joka alkaa toisesta sanasta ja päättyy neljänteen sanaan. Alla solujen numerot on merkitty niiden oikeaan yläkulmaan.

S (1,4)			
S (1,3)	S,VP (2,4)		
S (1,2)	S,VP (2,3)	VP (3,4)	
N,NP (1,1)	V,N,VP,NP (2,2)	V,N,VP,NP (3,3)	ADV (4,4)
Robots	*fish*	*fish*	*today*

Algoritmin 11 mukaisesti taulukossa täytetään ensin solut, jotka vastaavat yhden sanan mittaisia osia eli tässä tapauksessa solut (1,1), (2,2), (3,3) ja (4,4). Pseudokoodissa lauseen sanat lisätään ko. soluihin, mutta yllä olevassa taulukossa ne on selkeyden vuoksi sijoitettu alarivin solujen alapuolelle.

Alarivin soluissa sovellettavaksi sopivat vain muotoa $V \longrightarrow X$ olevat säännöt. Huomaa, että sääntöjä voidaan soveltaa samaan soluun toistuvasti, kuten esimerkiksi sanan ”*Robots*” kohdalla, jossa voidaan ensin soveltaa sääntöä $N \longrightarrow Robots$ ja sen tuloksena saatavaan symboliin N voidaan sen jälkeen soveltaa sääntöä $NP \longrightarrow N$. Näin soluun (1,1) saadaan symbolit N (substantiivi) ja NP (substantiivilauseke). Sanan *fish* kohdalla voidaan soveltaa sekä verbi- että substantiivisääntöjä, joten soluihin (2,2) ja (3,3) tulee kumpaankin neljä symbolia.

Ylemmillä olevissa soluissa voidaan soveltaa myös muotoa $V \longrightarrow X\ Y$ olevia sääntöjä. Soluun (i,j) voidaan lisätä symboli V mikäli on olemassa k, jolla solu (i,k) sisältää symbolin X ja solu (k+1,j) sisältää symbolin Y. Tämä tarkoittaa, että sanasta i alkava ja sanaan j päättyvä lauseen osa voidaan jakaa kahtia osiin (i,k) ja (k+1,j) siten, että ensimmäinen osa voidaan jäsentää muotoon X ja toinen osa voidaan jäsentää muotoon Y. Esimerkiksi solussa (2,4), joka vastaa sanoja ”fish fish today”, voidaan soveltaa sääntöä $S \longrightarrow NP\ VP$, koska solua (2,2) vastaava sana ”fish” voidaan jäsentää muotoon NP (substantiivilauseke) ja solua (3,4) vastaavat sanat ”fish today” voidaan jäsentää muotoon VP (verbilauseke). Toisin sanoen ko. sanat voidaan jäsentää kokonaiseksi lauseeksi. (Se tarkoittaisi ”kalat kalastavat tänään”.)

Jäsennyspuu

CYK-algoritmin tuloksena saadaan täytetty taulukko, jonka soluihin on merkitty välikkeet, jotka vastaavat lauseen osien mahdollisia jäsennyksiä. Näistä voidaan johtaa kaikki mahdolliset jäsennyspuut.

Jäsennyspuun rakentaminen aloitetaan juurisolmusta. Jos juurisolmu sisältää aloitussymbolin S, kuten yllä olevassa esimerkissä, voidaan annetut sanat jäsentää kokonaiseksi lauseeksi. Jos juurisolmu sisältää muita välikkeitä, voidaan sanat jäsentää muunlaisiksi lausekkeiksi (esimerkiksi sanat “Fish fish today” voitaisiin jäsentää sekä kokonaiseksi lauseeksi että verbilausekkeeksi VP).

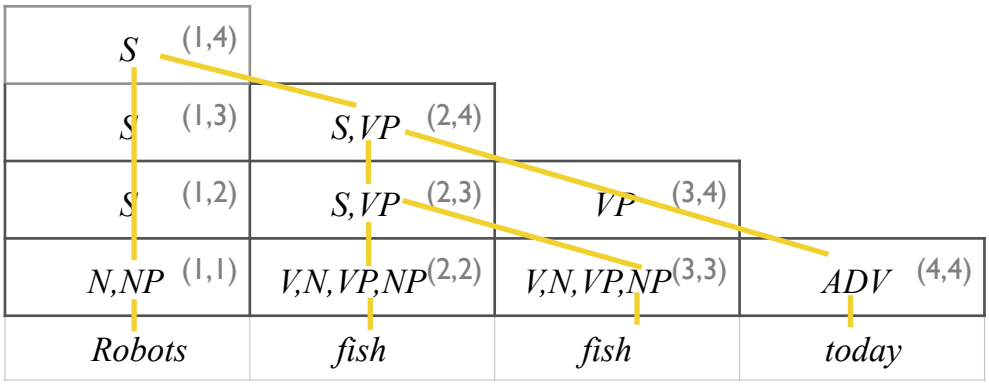
Jäsennystaulukon juurisolun kohdalla luodaan jäsennyspuun juurisolmu. Puun juurisolmun lapsiksi lisätään kaksi solmua, jotka vastaavat juurisolussa olevan välikkeen tuottamiseen sovellettuun sääntöön liittyviä kahta taulukon solua (i,k) ja (k+1,j) (ks. pseudokoodi). Yllä olevan esimerkin taulukossa juurisolun S tuotettiin säännöllä $S \longrightarrow NP\ VP$, missä säännön oikean puolen termit löytyivät taulukon soluista (1,1) (NP ”Robots”) ja (2,4) (VP ”fish fish today”).

Näin luoduissa kahdessa lapsisolmussa jatketaan samaan tapaan lisäämällä edelleen niiden lapsiksi solmut, jotka vastaavat niissä olevan välikkeen johtamiseen sovellettua sääntöä. Niiden sääntöjen kohdalla, joissa oikealla puolella on vain yksi välike, tuotetaan vain yksi lapsisolmu.

Jos päädytään soluun, joka sisältää useampia välikkeitä (poislukien välikkeet, jotka on johdettu saman solun välikkeistä), voidaan tuottaa

useampia erilaisia jäsennyspuita. Joka kerta, kun näin käy, jäsennyspuiden lukumäärä moninkertaistuu, eli mahdollisten jäsennysten määrä kasvaa eksponentiaalisesti.

Alla on edellisen esimerkin lauseen jäsennyspuu, joka on tällä kertaa yksikäsitteinen.



Muita jäsennysmenetelmiä

Koska yhteydettömän kielen lauseenjäsennys tuottaa usein monta erilaista jäsennyspuuta, jotka kukin yleensä johtavat erilaiseen tulkintaan, on tarpeen ottaa käyttöön täydentävää informaatiota. Tätä varten voidaan soveltaa ns. leksikaalistettua jäsennystä (*lexicalized parsing*), jossa sanaston (*lexicon*) semantiikka otetaan laskuun jäsennyksessä.

CYK-algoritmi voidaan melko suoraviivaisesti muuntaa muotoon, jossa se tuottaa todennäköisimmän jäsennyspuun, kun kieliopin sääntöihin on liitetty todennäköisyydet, jotka riippuvat sanoista, joihin sääntöjä sovelletaan. Esimerkiksi ”eat … with a fork” -rakenne on todennäköisempi kuin ”a fish with a fork”, minkä perusteella saadaan jäsennettyä kuvan 11 lause järkevällä tavalla.

Sanoihin sovellettavien sääntöjen todennäköisyyksien arvioiminen on sinällään vaikea ongelma, joka yleensä ratkaistaan koneoppimisen menetelmillä, eli tutkimalla suurta joukkoa valmiiksi jäsennettyjä lauseita ja estimoimalla tarvittavat todennäköisyydet esiintymisfrekvensseistä.

NLP:n sovelluksia

Jäsennyksen avulla ei-rakenteinen kieliainesto voidaan saattaa rakenteisempaan muotoon. Toisin sanoen luonnollinen kieli, jossa lauserakenteet ovat monimuotoisia ja moniselitteisiä, saatetaan vakiomuotoon, jossa sitä on helpompi hyödyntää laskennallisin menetelmin.

Tiedon kerääminen (*knowledge extraction*) voi tarkoittaa esimerkiksi tiettyyn aihepiiriin kuuluvien faktojen louhintaa ei-rakenteisesta tekstiaineistosta, kuten uutisista. Lauseesta ”Google bought DeepMind for $500M in January” voidaan siten johtaa lause

Omistaa(Google, DeepMind),

jota voidaan sen jälkeen hyödyntää yhdessä muiden samantyyppisten lauseiden kanssa, jos halutaan päätellä jotakin yritysomistuksista. Vastaavasti lauseesta ”*Valse triste* (Sad Waltz), Op. 44, No. 1, is a short orchestral work by the Finnish composer Jean Sibelius” voidaan johtaa mm. lause

Säveltäjä(Jean Sibelius, *Valse Triste*).

Tiedon keräämisen avulla voidaan tuottaa tietokantoja, joita voidaan edelleen soveltaa tekstin tiivistämisessä (merkityksessä tiivistelmän kirjoittaminen), kysymyksiin vastaamisessa (esim. START, start.csail.mit.edu) tai dialogijärjestelmissä (vrt. Turingin koe). Myös konekäännöksessä voidaan hyödyntää lauseenjäsennystä.

Haasteelliseksi tiedon keräämisen ja sen sovellukset tekee mm. se, että vaikka internetistä löytyy helposti valtavia määriä tekstiä, on hyvin vaikea päätellä automaattisesti, mikä osa siitä on luotettavaa. Olisi ikävää, jos tietoa keräävä tekoälyalgoritmi sattuisi ”lukemaan” fiktiivistä tekstiä ja tekisi siihen perustuvia päätelmiä. Tieto saattaa myös vanhentua hyvinkin nopeasti eikä kukaan välttämättä päivitä vanhentuneita tietoja.

Luonnollisen kielen käsittelyyn liittyviä oppimistavoitteita ei ole vielä päivitetty kurssin virallisiin oppimistavoitteisiin, mutta ne ovat suurin piirtein seuraavat:

Esitiedot

- osaa erottaa luonnollisen kielen lauseesta tärkeimmät lauseenjäsenet (subjekti, predikaatti, objekti)
- hahmottaa dynaamisen ohjelmoinnin periaatteen (taulukon täyttäminen solu kerrallaan)

Lähestyy oppimistavoitetta

- osaa tuottaa merkkijonoja soveltamalla formaalin kieliopin sääntöjä
- osaa tulkita jäsennyspuita

Saavuttaa oppimistavoitteet

- osaa soveltaa CYK-algoritmia yhteydettömän kielen lauseen jäsentämiseen

Syventää oppimistavoitteita

- osaa soveltaa leksikaalistettua lauseenjäsennystä todennäköisimmän jäsennyspuun tuottamiseen
- osaa arvioida formaalin kieliopin sääntöjen todennäköisyyttä kunkin pääsanan yhteydessä
- osaa kerätä tietoa lauseenjäsennyksen avulla suurista tekstiaineistoista
- osaa soveltaa kerättyä tietoa muissa NLP-sovelluksissa

10. 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 fysikaaliseen 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*)