

# Koneoppimisesta lyhyesti

Kai Hakala & Filip Ginter  
Turun yliopisto, Informaatioteknologian laitos  
Syksy 2015

# Koneoppimisesta lyhyesti

- **Koneoppiminen:** algoritmeja, jotka mukauttavat toimintaansa (koulutus)datasta oppimansa perusteella
- vrt. perinteiset algoritmit, jotka ovat joukko ennalta määriteltyjä sääntöjä
- Tavoitteena, että koneoppimisalgoritmi toimii paremmin opittuaan datasta jotakin
- Menetelmät on siis koulutettava ennen kuin niitä voidaan käyttää
- Koneoppimismenetelmiä useita, soveltuvat vain tietyn tyyppisiin ongelmiin.

- Vaikka luonnollisilla kielillä on yleinen rakenne, ne sisältävät usein poikkeuksia ja poikkeuksen poikkeuksia
- Kielen kuvaaminen manuaalisesti luoduilla säännöillä äärimmäisen työlästä, eikä kaikkia sääntöjä edes tunneta
- Lähes kaikki modernit NLP-menetelmät perustuvat koneoppimiseen

# Koneoppimisesta lyhyesti

- Koneoppimisen tavoitteena usein löytää hypoteesi (käytännössä jokin matemaattinen funktio), joka kuvaa koulutusdatan säännönmukaisuuksia
- Toivottavaa on toki, että koulutusdata edustaa sen taustalla olevaa ilmiötä hyvin
- Valittu koneoppimismenetelmä asettaa hypoteesille tiettyjä reunaehdoja
- Tuotetun hypoteesin avulla voidaan joko analysoida koulutusdatan taustalla olevaa ilmiötä tai ennustaa sen käyttäytymistä uusissa tapauksissa
- Tällä kurssilla ollaan kiinnostuneita nimenomaan ennustamisesta

# Ohjattu ja ohjaamaton oppiminen

- Kaksi yleisintä koneoppimissuuntausta ovat **ohjattu** ja **ohjaamaton** oppiminen (engl. supervised/unsupervised learning).
- Ohjatussa oppimisessa jokaisella koulutusdatan esimerkillä on jokin entuudestaan tunnettu leima (luokka/lukuarvo tms.)
- Koulutusdata on siis muotoa  $[(x_1, y_1), (x_2, y_2), \dots]$
- Koneoppimismenetelmän tavoitteena on oppia ennustamaan jokaiselle esimerkille kuuluva leima
- ts. jokainen syötearvo on sovitettava tunnettuun tulosteeseen.
- Yleisimpiä ohjattuja ongelmia ovat luokittelu ja regressio

# Ohjattu ja ohjaamaton oppiminen

- Ohjaamattomassa oppimisessä koulutusdata ei sisällä entuudestaan tunnettuja leimoja
- Data:  $[x_1, x_2, x_3, \dots]$
- Tällöin koneoppimismenetelmien tavoitteena on löytää datasta jonkinlainen rakenne tai säännönmukaisuus
- Yleisin ohjaamaton oppimisongelma on klusterointi, jossa tehtävänä on jakaa koulutusdata alijoukkoihin
- Klusterointi on siis kuten luokittelu, mutta luokkia ei tunneta entuudestaan ja koneoppimismenetelmälle annetaan "vapaat kädet" ryhmien muodostamiseen

# Regressio ja luokittelu

- **Regressio** ja **luokittelu** yleisimpiä ohjattuja ongelmia
- Regressiossa jokaiselle esimerkille pyritään ennustamaan jokin reaalilukuarvo
- Luokittelussa ennustetut arvot ovat puolestaan diskreettejä luokkia
- Yksinkertaisimmillaan luokkia on vain kaksi (binääriluokittelu)
- Koska koulutusdatan leimat ovat entuudestaan tunnettuja, voidaan koneoppimismenetelmän löytämän hypoteesin hyvyttä arvioida laskemalla ennustettujen leimojen ja tunnettujen leimojen eroavuus (kustannusfunktio, cost function) ja täten valita hypoteesi, joka sopii parhaiten dataan

# Moniluokkaluokittelu

- Multiclass classification: luokkia enemmän kuin kaksi, mutta valitaan vain yksi per esimerkki
- Multilabel classification: yksi esimerkki voi kuulua useaan luokkaan
- Molemmat ongelmat voidaan ratkaista kouluttamalla useita luokittelijoita, yksi jokaiselle luokalle (1-vs-rest, binary relevance)
- Jos halutaan vain yksi luokka, valitaan luokittelija, jolla suurin "varmuus" (confidence)
- Jos halutaan useita luokkia, hyväksytään jokaisen luokittelijan ennuste
- Parempiakin menetelmiä toki olemassa (ottavat huomioon luokkien väliset riippuvuudet jne.)



# Sekvenssiluokittelu

- Sekvenssiluokittelussa koneoppimismenetelmä ei tarkastele yksittäisiä esimerkkejä, vaan esimerkkien oletetaan olevan riippuvaisia toisistaan
- Tällöin menetelmien on tarkasteltava useiden esimerkkien muodostamaa kokonaisuutta, tehdäkseen jokaista esimerkkiä koskevat päätökset
- Koska sanat ovat kirjainsekvenssejä ja virkkeet sanasekvenssejä, ovat tällaiset menetelmät hyvin yleisiä luonnollisen kielen käsittelyssä
- Tällä kurssilla ei tarkastella näiden menetelmien toteutuksia syvällisesti, mutta menetelmiin törmätään myöhemmillä luennoilla
- Esim. viittako sana "Washington" henkilöön vaiko osavaltioon? Entä jos edellinen sana oli "George"?

# Esimerkkejä

- Regressio: Arvioi asunnon hinta, kun sen koko ja etäisyys Turun keskustasta tunnetaan
- Luokittelu: Päättelä, mihin sanaluokkaan annettu sana kuuluu (sanaluokat entuudestaan määritelty)
- Klusterointi: Jaa joukko sanoja neljään alijoukkoon niiden samankaltaisuuteen perustuen (entuudestaan ei tietoa, mitkä nämä neljä joukkoa ovat)

# Piirteet

- Tähän saakka ollaan puhuttu koulutusdatan esimerkeistä määrittelemättä niitä sen tarkemmin
- Käytännössä jokainen esimerkki esitetään sen ominaisuuksia kuvaavilla numeerisilla arvoilla, joita kutsutaan **piirteiksi** (engl. feature)
- Tilastotieteessä näitä kutsutaan riippumattomiksi tai selittäviksi muuttujiksi (engl. predictor)
- Jokainen esimerkki on siis joukko piirteitä, jotka esitetään piirrevektorina  $x$
- Tällöin jokainen esimerkki voidaan esittää jonakin piirreavaruuden pisteenä  $x = [x_1, x_2, x_3, \dots]$
- (Älä sekoita aiempien kalvojen annotaatioon, jossa  $x_1$  ja  $x_2$  olivat erilliset esimerkit)
- Joukko esimerkkejä on luonnollista esittää matriisina

# Piirteet

- Joskus käyttämämme data on valmiiksi piirre-esityksenä
- NLP:ssä piirteet on yleensä muodostettava itse
- Tärkein osa toimivaa ratkaisua onkin usein hyvin valitut piirteet, valitulla koneoppimismenetelmällä on pienempi vaikutus
- Koodarin tehtävänä siis rakentaa järjestelmä, joka tuottaa järkeviä piirteitä
- Nykyisin trendikäs "Deep learning" pyrkii pääsemään eroon manuaalisesti luoduista piirteistä

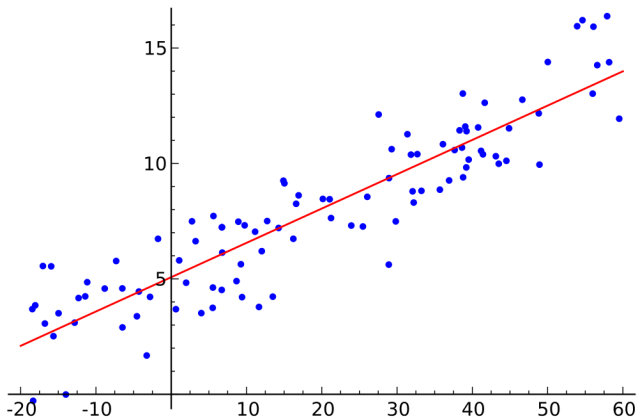
# Piirteet

- Piirteet on esitettävä numeerisesti, mutta tämä ei estä kategoristen ominaisuuksien kuvaamista
- Totuusarvot voidaan esittää numeerisesti 0/1
- Kategoriset ominaisuudet useana totuusarvona
- Esim. auton tyyppi: pakettiauto, farmari, *urheiluauto*  $\Rightarrow$  pakettiauto=0, farmari=0, urheiluauto=1
- Huom: jos tuotettua hypoteesia halutaan tulkita, yhden tyyppin tulisi olla referenssinä ( $[0,0,0] \Rightarrow$  ei mikään auto?)

# Lineaarinen regressio

- Yksinkertainen regressiomalli, joka sovittaa koulutusdataan hypertason
- Hypoteesi on tällöin muotoa
$$h(x) = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 \dots$$
- $x$  on esimerkkimme piirre-vektori
- $\theta$  sisältää hypoteesin parametrit, jotka on valittava oikein oppimisvaiheessa, jotta  $h(x) \approx y$  kaikilla koulutusdatan  $(x,y)$ -pareilla

# Lineaarinen regressio



Kuva: By Sewaqu (Own work) [Public domain], via Wikimedia Commons

# Lineaarinen regressio

- Kun jokin koulutusdatan esimerkki  $x$  on valittu, voidaan hypoteesin oikeellisuutta verrata todelliseen arvoon  $y$
- Usein käytetään ennustetun ja todellisen arvon erotuksen neliötä
- Kustannusfunktio arvioi hypoteesin hyvyttä koko koulutusdatalle, esim. erotusten neliöiden keskiarvolla (engl. mean squared errors, MSE)
- Täten kustannusfunktioiksi saadaan
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (y_i - h_{\theta}(x_i))^2$$
- Nyt varsinaisen oppimisalgoritmin tehtävänä on minimoida tämä kustannus, ts. löytää parametrin  $\theta$  arvot, jotka minimoivat sen



# Gradient descent

- Gradient descent on yleinen tapa etsiä funktion minimi (tarkemmin sanottuna muuttujien arvot, jotka tuottavat funktion minimin)
- Menetelmässä lasketaan funktion gradientti (osittaisderivaatat) jossakin aloituspisteessä ja "astutaan" pieni askel gradientin vastaiseen suuntaan (tästä nimitys "descent")
- (Edellytyksenä siis on, että kyseinen funktio on differentoituva)
- Tätä jatketaan iteratiivisesti, kunnes päädytään lokaaliin minimiin (tai jokin muu lopetuskriteeri täyttyy)
- Mikäli minimoitava funktio on konvekksi, löydetty lokaali minimi on myös globaali minimi
- Menetelmä ei takaa globaalin minimin löytämistä, mikäli tarkasteltu funktio ei ole konvekksi

# Gradient descent

- Matemaattisempi määritelmä kustannusfunktion tapauksessa:
- $\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta)$
- Tämä päivitys tehdään jokaiselle  $\theta_i$  samanaikaisesti, iteroidaan kunnes minimi löydetty!
- $\alpha$  säätelee askeleen suuruutta (engl. learning rate tai step size)
- Liian pieni  $\alpha$  tekee menetelmästä hitaan ja liian suuri estää minimin löytämisen (algoritmi loikkaa minimin yli puolelta toiselle)
- Lineaarisen regression kustannusfunktio on konvekksi ja sen osittaisderivaatat ovat melko yksinkertaisia, mutta jätetään niiden laskeminen kuitenkin tämän kurssin ulkopuolelle

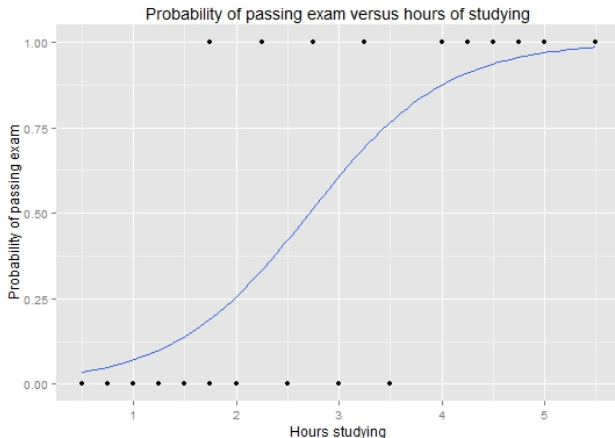
# Logistinen regressio

- Logistisen regression hypoteesi on muotoa  $h(x) = \frac{1}{1+e^{-h_{linear}(x)}}$
- Kuuluu yleistettyjen lineaaristen mallien joukkoon
- Nimestään huolimatta käytetään yleensä luokitteluun
- Logistinen funktio rajoittaa hypoteesin arvot välille (0,1), miksi?
- Hypoteesi tulkitaan todennäköisyytenä, että  $y=1$

# Logistinen regressio

- Esim.  $h(x) = 0.7 \Rightarrow 70\%$  todennäköisyys että  $y = 1$  (30% että  $y=0$ )
- Jos  $h(x) > 0.5$ , ennustetaan luokka 1
- Miksei lineaarinen regressio toimi luokittelussa, jos asetetaan samanlainen raja-arvo?

# Logistinen regressio



Kuva: By Michaelg2015 (Own work) [CC BY-SA 4.0  
(<http://creativecommons.org/licenses/by-sa/4.0>)]

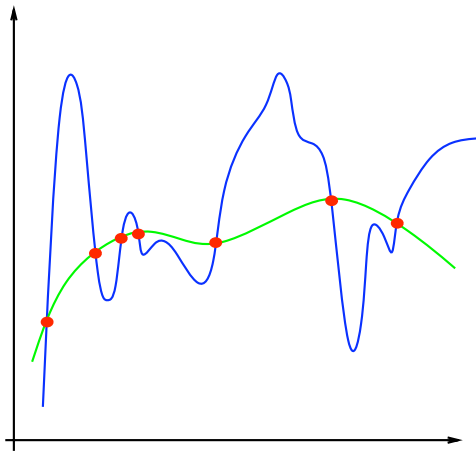
# Logistinen regressio

- Logistisen regression kanssa ei ole järkevää käyttää samaa kustannusfunktiota kuin lineaarisssa regressiossa, miksi?
- Sen sijaan
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m (y_i * \log(h_{\theta}(x_i)) + (1 - y_i) * \log(1 - h_{\theta}(x_i)))$$
- Muistetaan, että kyseessä on luokittelu, joten  $y$  on aina joko 1 tai 0
- Täten funktion alkuosa käsittelee tapauksia, jossa  $y = 1$ , ja loppuosa tapauksia, jossa  $y = 0$
- Kustannus on konvekssi ja voidaan minimoida Gradient Descent -menetelmällä

# Ylisovittaminen

- Kuvitellaan, että meillä on lineaarisen mallin sijaan monimutkainen polynomiaalinen malli, ts. sovitamme koulutusdataan korkeamman asteen polynomin
- Arvioidaan lineaarisen regression tavoin hypoteesin virhettä kustannusfunktiolla ja minimoidaan se
- On mahdollista, että oppimisalgoritmimme tuottaa täysin virheettömän hypoteesin, mutta sillä tehdyt ennusteet ovat käyttökelvottomia. Miksi?

# Ylisovittaminen




Kuva: By Tomaso Poggio [Attribution], via Wikimedia Commons




# Ylisovittaminen

- On aina mahdollista tuottaa niin monimutkaisia hypoteeseja, että ne selittävät minkä tahansa datan
- Mikäli tällainen malli ei kuitenkaan pysty tuottamaan järkeviä ennustuksia koulutusdatan ulkopuolella, se on **ylisovitettu** (engl. overfitting) koulutusdataan
- Toisin sanoen oppimisalgorithmi ”oppii ulkoa” koulutusesimerkit, muttei osaa yleistää tätä tietoa muuhun dataan
- Jotta tältä vältyttäisiin on syytä valita yksinkertainen hypoteesi, joka selittää koulutusdatan halutulla tasolla.

# Regularisointi

- Yksi tapa välttää ylisovittamista on lisätä kustannusfunktioon termi, joka mittaa hypoteesin kompleksisuutta.
- Tätä kutsutaan **regularisoinniksi** (engl. regularization)
- Esim. lineaarisessa regressiossa
$$J(\theta) = \frac{1}{m}(\sum_{i=1}^m (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{j=1}^n \theta_j^2)$$

- Tässä  $\lambda$  on regularisointiparametri, joka säätelee regularisoinnin määrää
- Liian voimakas regularisointi yksinkertaistaa hypoteesia liikaa, jolloin malli alisovittuu (engl. underfitting)

# Regularisointi

- Miten siis valita sopiva  $\lambda$ ?
- Mikäli dataa on riittävästi tarjolla, on se tapana jakaa kolmeen osaan:
  - Ensimmäinen osa käytetään kouluttamiseen (train)
  - Toinen osa regularisointiparametrin validointiin (devel) 
  - Kolmas osa varsinaiseen ennusteiden arviointiin (test)
- Tarkoituksena on kouluttaa useita hypoteeseja eri  $\lambda$ :n arvoilla koulutusdataa käyttäen
- Jokaisen hypoteesin suorituskyky evaluoidaan devel-datalla ja valitaan hypoteesi, joka suoriutuu ennustuksista parhaiten
- Tehdään varsinaiset test-datan ennusteet tällä mallilla ja evaluoidaan ne
- devel- ja test-datajoukot ovat erilliset, sillä hypoteesi voi ylisovittua myös devel-dataan

# Ristiinvalidointi

- Joskus dataa ei ole riittävästi, jotta voisimme jakaa sen erillisiin train- ja devel-joukkoihin
- Tällöin regularisointiparametri voidaan valita ristiinvalidointimenetelmällä (engl. cross-validation)
- Ideana on jakaa koulutusdata  $n$ :ään osaan, joista  $n-1$ :tä käytetään kouluttamiseen ja jäljelle jäänyttä validointiin
- Tämä toistetaan  $n$  kertaa, aina eri validointiosalla ja lasketaan näistä esim. keskiarvo
- Yleisimpiä 10-fold CV ( $n=10$ ), leave-one-out CV (LOOCV,  $n$ =esimerkkien määrä)
- Ongelma: koneoppimismenetelmä koulutettava  $n$  kertaa, mikä voi olla laskennallisesti kallista

- Kustannusfunktiot yleensä valittu siten, että ne voidaan optimoida nopeasti
- Eivät kuitenkaan aina parhaita menetelmän evaluointiin todellisessa tehtävässä
- devel- ja test-datan kanssa käytetään yleensä jotakin muuta metriikkaa, joka soveltuu paremmin kyseiseen sovellukseen

- **Accuracy:** Oikeellisten päätösten suhteellinen määrä luokittelussa
- Ongelma: mikäli luokkien jakauma ei ole tasainen, on accuracy vaikeasti tulkittavissa
- Esimerkki: Ennustetaan onko henkilö miljonääri
  - Suomessa 5M asukasta ja 25K miljonääriä
  - Ennustamalla aina "Henkilö ei ole miljonääri" luokittelija vastaa oikein 99,5% tapauksista

# Evaluointimetriikat

- Parempi metriikka luokittelulle on **F-score**, jonka määrittelyyn tarvitaan käsitteet **precision** ja **recall**
- Ajatellaan jälleen binääriluokittelua, jossa olemme kiinnostuneet positiivisesta luokasta (esim. "henkilö on miljonääri")
- Nyt:
- **Precision** =  $\text{True Positives} / (\text{True Positives} + \text{False Positives})$
- **Recall** =  $\text{True Positives} / (\text{True Positives} + \text{False Negatives})$
- **Precision**: Miten suuri osuus luokittelijan positiivisista ennustuksista on tosia
- **Recall**: Miten suuren osuuden testijoukossa olleista positiivisista esimerkeistä luokittelija tunnisti
- On helppo luoda menetelmä, jonka joko precision- tai recall-arvo on korkea (muttei molemmat)

- F-score: precision- ja recall-arvojen harmoninen keskiarvo
- $F\text{-score} = 2 * \text{precision} * \text{recall} / (\text{precision} + \text{recall})$
- Harmoninen keskiarvo painottaa aina pienempää arvoa
- $\Rightarrow$  Korkean F-scoren saavutetaan ainoastaan, kun sekä precision että recall ovat korkeita



- Palataan takaisin miljonääri-esimerkkiin
- Kuvitellaan, että luokittelijamme on tehnyt ennusteet koko Suomen väestölle
- Luokittelija on tunnistanut 20000 miljonääriä, 5000 on jäänyt havaitsematta, ja 30000 ei-miljonääriä on virheellisesti luokiteltu miljonääreiksi
- $\Rightarrow TP=20000, FP=30000, FN=5000$
- $Precision = 20000 / (20000 + 30000) = 0.4$
- $Recall = 20000 / (20000 + 5000) = 0.8$
- $F\text{-score} = 2 * 0.4 * 0.8 / (0.4 + 0.8) \approx 0.53$

# Evaluointimetriikat

- Kun luokkia on enemmän kuin kaksi, F-scoresta voidaan laskea keskiarvo (usealla eri tavalla)
- **Macro-averaged F-score:** Lasketaan Precision ja Recall jokaiselle luokalle erikseen ja otetaan näiden aritmeettiset keskiarvot. Sen jälkeen näistä harmoninen keskiarvo.
- Jokainen luokka on tällöin keskiarvossa yhtä tärkeä, riippumatta niiden todellisesta jakaumasta
- **Micro-averaged F-score:** Lasketaan F-score kuten binääriluokittelussa, mutta kaikki positiiviset esimerkit luokasta riippumatta samalla kertaa
- Tällöin luokat, joilla on enemmän esiintymiä testijoukossa painottuvat enemmän
- Jälkimmäinen tapa yleisempi

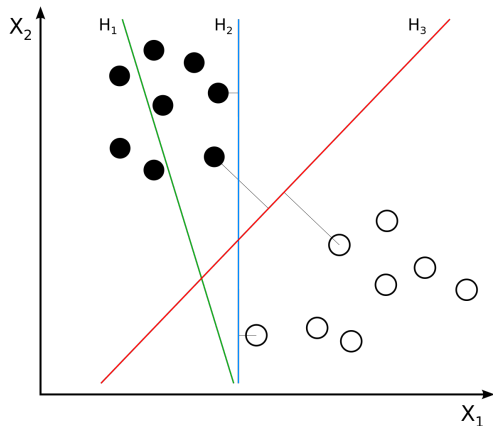
# Piirteiden arvoasteikoista

- Jotkin koneoppimismenetelmät ovat herkkiä piirteiden arvoasteikoiden vaihtelulle (esim. SVM)
- Eri piirteiden arvot kannattaa tästä syystä pakottaa samalle arvoasteikolle
- Yksinkertaisinta lienee piirteiden skaalaaminen välille  $[0, 1]$
- Vaihtoehtoisesti piirre voidaan standardoida keskiarvoon 0 ja keskihajontaan 1
- Tällä kurssilla näitä saattaa tarvita, mikäli käyttää esim. totuusarvopiirteitä ja lukumääräpiirteitä samassa luokittelijassa

# Tukivektorikoneet

- Yleensä luokitteluun käytetty menetelmä, jonka tarkempi kuvaus jätetään tämän kurssin ulkopuolelle
- Lineaarisen tukivektorikoneen (engl. support vector machine, SVM) kustannusfunktio ei poikkea suuresti logistisesta regressiosta (hieman yksinkertaisempi)
- Yrittää jakaa koulutusdatan hypertasolla, jonka marginaali koulutusdatan pisteisiin on mahdollisimman suuri

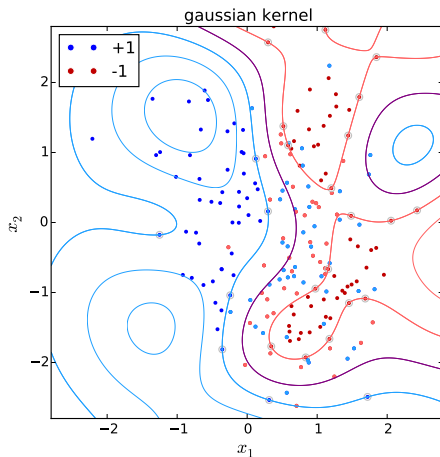
# Lineaarinen tukivektorikone



**Kuva:** By User:ZackWeinberg, based on PNG version by User:Cyc [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>)], via Wikimedia Commons

- Usein formalisoitu siten, että regularisointiparametri ( $C$ ) onkin varsinaisen virhetermin kerroin, ei regularisointitermin kerroin
- $\Rightarrow$  pieni  $C$ :n arvo vastaa voimakasta regularisointia
- Käytetään tällä kurssilla mustina laatikkoina
- Vaikka SVM on lineaarinen malli, voidaan sillä tuottaa epälineaarisia hypoteeseja "ytimien" (engl. kernel) avulla
- Huom: kerneleillä on yleensä omia hyperparametreja, jotka on asetettava sopiviksi regularisointiparametrin lisäksi, jotta yli/alisovittamiselta vältytään.

# Kernelimenetelmä



Kuva: By Kkddkkdd (Own work) [CC BY-SA 4.0  
(<http://creativecommons.org/licenses/by-sa/4.0>)]

# Koneoppimista Pythonilla

- Koneoppimiseen on tarjolla useita työkaluja ja ohjelmointikirjastoja
- Kattavin Pythonille lienee Scikit-learn:  
<http://scikit-learn.org/stable/>
- Sisältää useita koneoppimisalgoritmeja, joita voidaan käyttää mustina laatikkoina
- Sisältää paljon työkaluja evaluointiin, datan esikäsittelyyn jne.
- Myös valmiita aineistoja menetelmien kokeiluun:  
<http://scikit-learn.org/stable/datasets/>
- Dokumentoitu kattavasti ja oppikirjamaisesti



# Koneoppimista Pythonilla

- Luokittelijan kouluttaminen:

```
from sklearn import svm
```

```
train_X = [[0, 0], [1, 1]]
```

```
train_y = [0, 1]
```

```
clf = svm.SVC(kernel='linear', C=1)
```

```
clf.fit(train_X, train_y)
```

# Koneoppimista Pythonilla

- Scikit tarjoaa myös kattavan valikoiman evaluointimetriikoita:

[http://scikit-learn.org/stable/modules/model\\_evaluation.html](http://scikit-learn.org/stable/modules/model_evaluation.html)

```
from sklearn.metrics import f1_score
```

```
validation_y_pred = clf.predict(validation_X)  
f1_score(validation_y_true, validation_y_pred)
```

- Datan pilkkominen ja ristiinvalidointi:

[http://scikit-learn.org/stable/modules/cross\\_validation.html](http://scikit-learn.org/stable/modules/cross_validation.html)

# Koneoppimista Pythonilla

- Jos vaikuttaa siltä, että tuottamanne piirteet vaativat skaalausta: <http://scikit-learn.org/stable/modules/preprocessing.html>

# Koneoppimista Pythonilla

- Esitetyissä esimerkeissä on piirteitä ollut vain pari
- NLP:ssä käytetään usein esim. dokumentissa esiintyvien uniikkien sanojen lukumääriä jne.
- Tällöin piirteiden lukumäärä voi kasvaa satoihin tuhansiin, mutta suurin osa niistä on nolliä
- Nollien tallentaminen listaan/vektoriin/matriisiin syö tuhottomasti muistia
- Kannattaa siis yleensä käyttää "harvoja" (engl. sparse) tietorakenteita

- Esim harvat matriisit scipy-kirjastosta:  
<http://docs.scipy.org/doc/scipy/reference/sparse.html>
- Olettaa, että jokainen matriisin lukuarvo, jota ei eksplisiittisesti mainittu on 0
- Scikit-learnin kanssa on myös helppo käyttää Pythonin dict-olioita, jotka voidaan konvertoida matriiseiksi valmiilla apufunktiolla:  
[http://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.DictVectorizer.html](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html)

# Koneoppimista Pythonilla

- Mikäli mustat laatikot eivät riitä, kannattaa tutustua Theano-kirjastoon:  
<http://deeplearning.net/software/theano/>
- Theano mahdollistaa matemaattisten lausekkeiden esittämisen symbolisesti
- Sisältää funktiot esim. gradientin laskemiseen
- Täten on "helppo" määritellä esim. logistisen regression tai neuroverkkojen hypoteesi- ja kustannusfunktiot ja kouluttaa ne.
- Kääntää matemaattiset lausekkeet taustalla C-koodiksi, joten suoritus on nopeaa
- Tukee myös GPU-laskentaa

## Bonus: Neuroverkot

- Neuroverkot jäljittelevät hermosolujen toimintaa
- Yksi solmu/solu (engl. node) koostuu painotetuista syötteistä, jotka summataan, sekä aktivointifunktiosta, joka tuottaa solmun output-arvon syötteiden summasta
- Mikäli aktivointifunktiona käytetään logistista funktiota, yksi solmu vastaa logistista regressiota
- Kokonainen verkko rakennetaan pinoamalla solmuja päällekkäin, ts. yhden solmun tuloste on seuraavassa tasossa olevien solmujen syöte

## Bonus: Neuroverkot

- Verkko oppii päivittämällä jokaisen solmun syötearvojen painokertoimia (kuten  $\theta$  regressioesimerkeissä)
- Kouluttaminen on kuitenkin regressiota hankalampaa, sillä painoarvojen muutokset tulee "vyöryttää" läpi koko verkon, kaikille tasoille
- Gradient descent -menetelmän sijaan verkko koulutetaan usein stochastic (mini-batch) gradient descent sekä backpropagation -menetelmillä



## Bonus: Neuroverkot

