Luokitteluvirheen estimaattoreiden ja luokittelumenetelmien vertailu

Jesse Sorvali

7. kesäkuuta 2023

Tiivistelmä

Jyväskylän yliopisto, Matematiikan ja tilastotieteen laitos. Sorvali Jesse.
Luokitteluvirheen estimaattoreiden ja luokittelumenetelmien vertailu. Pro gradu -tutkielma. 28 sivua + 1 liite (22 sivua). Tilastotiede. 18. toukokuuta 2023.

Tutkimuksessa pohjaeläinaineiston pohjalta haluttiin simuloida yksilöitä, joita ryhdyttiin luokittelemaan eri luokittelumenetelmillä. Tarkastelun kohteena oli, miten luokittelumenetelmät vertailevat keskenään normaalijakautuneisuusoletuksen ollessa voimassa ja miten eri estimaattorien tulokset vertautuvat keskenään. Tavoitteena oli estimoida yleistämisvirhettä.

Kun mallissa on kaksi luokkaa ja kaksi piirettä otoskoon ollessa 50, havaittiin kvadraattisen luokittelijan olevan optimi normaalijakautunaisuusoletuksella tuottaessa pienimpiä virheitä. Lähimmäksi kvadraattisen luokittelijan tuloksia pääsi lineaarinen luokittelija. Vertaillessa luokittelijoita huonoiten suoriutui lähinaapurinmenetelmä. Erot lineaarisen luokittelijan ja tukivektorikoneiden välillä eivät olleet suuria, vaikkakin lineaarisen luokittelijan vaihtelu oli pienempää.

Kun otoskoko oli 500 ja mallit monimutkastuivat sisältäen 8 luokkaa ja 8 piirrettä, suoriutui lineaarinen luokittelija edelleen toisiksi parhaiten kvadraattisen luokittelijan antaessa pienimmät virheet. Neuroverkot ja lähinaapurinmenetelmä havaittiin suoriutuvan huonoiten. Tukivektorikone lineaarisella ytimellä suoriutui sädeperusteista paremmin ja myös sen vaihtelu pysyi pienempänä.

Kolmannessa tapauksessa, missä luokkia oli 50 ja piirteitä 8 otoskoon ollessa 10000, suoriutui lähinaapurinmenetelmä edelleen muita menetelmiä huonommin. Alhaisimmat virheet olivat edelleen kvadraattisella luokittelijalla, mutta toiseksi parhaimmat tulokset tuotti satunnaismetsä.

Estimaattoreiden tulokset vaihtelivat tapauskohtaisesti. Toistuva jako opetus- ja testiaineistoksi jaolla 90/10, ristiinvalidointi arvolla $K{=}N$, sekä $\hat{Err}^{(.632)}$ havaittiin useasti eroavan vähiten yleistämisvirheestä. Toisaalta $\hat{Err}^{(.632)}$ havaittiin myös useasti aliestimoimassa yleistämisvirhettä.

Huonoiten estimaattoreista suoriutuivat toistuva jako opetus- ja testiaineistoksi 50-50-jaolla, $\hat{Err}^{(1)}$, sekä ristiinvalidointi arvolla $K = 2$.

Sisällys

1 Johdanto

Tutkielman tarkoituksena on vertailla luokitteluvirheen estimaattoreita sekä luokittelumenetelmiä. Taustana toimii Lasse Moision gradututkielma (2016), jossa tutkittiin kvadraattisen luokittelijan ennustevirheitä käyttäen eri evaluointimenetelmiä, kuten ristiinvalidointia ja uudelleenotantaa. Tässä työssä tutkielmaa laajennetaan käyttämällä muita luokittelumenetelmiä kvadraattisen luokittelijan lisäksi. Tarkastelun kohteena on, mikä evaluointimenetelmä antaa pienimmän virheen, ja miten luokittelumenetelmät eroavat toisistaan.

Luokittelumenetelmiä käytetään kvalitatiivisen vastemuuttujan luokan ennustamisessa. Koneoppimisessa luokittelua sanotaan ohjatuksi oppimiseksi, missä tietokoneelle annetaan syötteenä luokiteltua dataa, josta opitaan ja jonka avulla luokitellaan uusia luokittelemattomia havaintoja. Luokittelussa luokkia on joko 2 tai useampia. Luokittelijan halutaan toimivan hyvin opetusaineiston lisäksi testiaineistossa, jota ei ole käytetty luokittelijan opettamisessa (Hastie, T., 2016).

Luokittelijan hyvyyttä voidaan estimoida vertaamalla oikeiden ennusteiden osuutta jo valmiiksi luokitellussa testiaineistossa. Virhe saadaan kahdelle luokalle ja samanarvoisille virheille laskettua kaavalla

$$\frac{FP + FN}{TP + TN + FP + FN},$$

missä TP on oikein luokitellut positiiviset tapaukset, FP väärin luokitellut positiiviset tapaukset, TN oikein luokitellut negatiiviset tapaukset, ja FN väärin luokitellut negatiiviset tapaukset (Lopez-Bernal, 2021). Testivirheeksi, eli yleistämisvirheeksi, sanotaan odotettua ennustevirhettä riippumattomasta testiaineistosta N kokoiselle kiinteälle opetusaineistolle $O = ((w_{k_1}, x_1), ..., (w_{k_N}, x_N))$ (Moisio, 2016)

$$Err_O = E_{x,w|O}[\lambda(\hat{\alpha}(x))|O],$$

missä $\lambda(\hat{\alpha(x)}|w)$ on tappiofunktio, joka tässä tapauksessa on indikaattorifunktio $1(\hat{\alpha}(x) \neq w)$, missä w on oikea luokka ja $\hat{\alpha}(x)$ päätössääntöfunktio, joka estimoidaan opetusaineistosta. Lisäksi on syytä verrata virhettä Bayes-virheeseen, mikä on pienin mahdollinen luokitteluvirheen todennäköisyys (Hastie, T., 2016).

Aineisto, johon simulointikoe perustuu, on pohjaeläindata, joka sisältää 6966 havaintoa vesistössä asuvista pohjaeläimistä (Ärje et al., 2013). Se sisältää 50 pohjaeläinten luokkaa ja lisäksi pohjaeläinten kuvista laskettuja tietoja kuten pohjaeläinten pituudet ja värit, sekä harmaasävypiirteitä kuten maksimi, moodi, minimi jne. Näistä piirteistä valitaan parhaiten luokkia erottelevat piirteet, joiden avulla pohjaeläimiä luokitellaan.

Virheet estimoidaan eri luokittelumenetelmillä, jolloin tarkastasteluna on, mitkä luokittelumenetelmät ja mitkä evaluointimenetelmät antavat pienimmät virheet. Tätä tutkitaan käyttäen hyödyksi uusio-otantaa ja ristiinvalidointia. Virheitä verrataan Bayes-virheeseen ja yleistämisvirheeseen.

2 Luokittelumenetelmät

Tutkimuksessa vertaillaan ennustevirheitä, jotka on saatu käyttäen ristiinvalidointia, uusiootantaa ja aineiston toistuvaa jakoa opetus- ja testiaineistoksi. Luokittelumenetelmistä on valittu tutkimuksen kohteeksi lineaarinen luokittelija (LDA), kvadraattinen luokittelija (QDA), satunnaismetsä, neuroverkot, lähinaapurinmenetelmä ja tukivektorikone sädeperusteisella sekä lineaarisella ytimillä.

2.1 Lineaarinen ja kvadraattinen luokittelija

Oletetaan yksinkertaisuuden vuoksi, että jokaisen luokan tiheyttä voidaan mallintaa multinormaalijakaumalla (Moisio, 2016).

$$f_k(x) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma_k|^{1/2}} \exp(-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k),$$

missä Σ_k on kovarianssimatriisi, μ_k keskiarvo ja p on ulottuvuus.

Lineaarisessa luokittelijassa oletetaan, että luokilla on sama kovarianssimatriisi $\Sigma_k = \Sigma \; \forall k$.

Esimerkiksi kahden luokan tapauksessa havainto luokitellaan luokkaan 2, jos

$$x^T\hat{\Sigma^{-1}}(\hat{\mu_2} - \hat{\mu_1}) > \frac{1}{2}\hat{\mu_2}^T\hat{\Sigma^{-1}}\hat{\mu_2} - \frac{1}{2}\hat{\mu_1}^T\hat{\Sigma^{-1}}\hat{\mu_1} + \log(N_1/N) - \log(N_2/N).$$

Kun kovarianssimatriiseja ei oleteta yhtäsuuriksi, päädytään kvadraattiseen luokittelijaan

$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log\pi_k.$$

Kaikkien luokkaparien k ja l päätösrajaa kuvaa kvadraattinen yhtälö $\{x : \delta_k(x) = \delta_l(x)\}$.

Yksilö luokitellaan siihen luokkaan, jossa $\delta_k(x)$ on suurin (Fulcomer, 1974).

Sekä lineaarinen luokittelija ja kvadraattinen luokittelija olettavat normaalijakautuneisuuden. Lineaarisen luokittelijan tapauksessa päätösraja on suora, ja kvadraattisella luokittelijalla se on käyrä. Kvadraattisen luokittelijan on aiemmissa tutkimuksissa havaittu olevan lineaarista parempi luokittelija, kun normaalijakautuneisuuden oletus pätee luokkien jakaumalle (Agostini, et al. 2003). Toisaalta lineaarisella luokittelijalla on tapana olla parempi luokittelija kuin kvadraattinen, jos opetushavaintoja on vähän ja varianssin pienentäminen on aiheellista (James, et al. 2013).

2.2 Satunnaismetsä

Päätöspuu on eräs luokittelumenetelmä, missä juurisolmusta kuljetaan alaspäin tietyn ehdon täyttyessä. Kun puuta on kuljettu alimmalle solmulle, on sille löydetty luokka. (Duda, et al. 2000). Satunnainen metsä on ratkaisu päätöspuiden suurempiin luokitteluvirheisiin (Hastie, et al. 2016). Siinä käytetään laukutusta, missä ideana on keskiarvoistaa monia kohinaa sisältäviä mutta harhattomia malleja ja siten pienentää varianssia. Lisäksi koska jokainen laukutuksella saatu puu on samoin jakautunut, on niiden sisältämä harha sama kuin yksittäisten puisten. Satunnaisen metsän algoritmi on seuraavanlainen:

1. Toista $b = 1, ..., B$

1.1) arvo otoskoon N kokoinen uusio-otantaotos Z^* opetusdatasta.

1.2) Sovita satunnaisen metsän puu T_b uusio-otannalla saatuun dataan rekursiivisesti toistamalla seuraavia askeleita jokaiseen lehtisolmuun, kunnes pienin solmun kokoa $n_{\min}$ on saavutettu:

1.2.1) Valitse m muuttujaa satunnaisesti p muuttujista.

1.2.2) Valitse paras jakokohta m muuttujista.

1.2.3) Jaa solmu kahteen lapsisolmuun.

2. Palauta puiden kooste T_{b1}^B

Uuden havainnon luokaksi valitaan eniten ääniä saanut luokka.

Mikäli B kappaletta muuttujia ovat samoin jakautuneita, mutta eivät riippumattomia sisältäen parittaisen korrelaation ρ, on keskiarvon varianssi $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. B:n kasvaessa toinen termi katoaa, mutta ensimmäisen termin korrelaatio rajoittaa keskiarvoistamisen hyötyjä (Hastie, et al. 2016). Ideana on siten parantaa laukutuksen aikaansaamaa varianssin pienennystä vähentämällä puiden välistä korrelaatiota siten, ettei varianssi kasva liikaa. Se saadaan aikaan valitsemalla satunnaisesti syötteen muuttujat. Erityisesti ennen jokaista jakoa, valitaan $m \leq p$ syötemuuttujia satunnaisesti jakokohdan ehdokkaiksi. Breiman (2001) suosittelee luokittelua varten $m = \lfloor \sqrt{n} \rfloor$.

Metsän päätöspinta jakaa lineaarisen päätöspinnan suorakaiteisiin (James, et al, 2013). Satunnaisen metsän huonoina puolina ovat herkkyys opetusaineiston pienille muutoksille ja taipumus ylisovittamiseen (Botaeng, 2020).

2.3 Neuroverkot

Neuroverkkojen keskeisenä ideana on poimia syötteiden lineaarikombinaatiot johdettuina piirteinä ja mallintaa kohde näiden piirteiden epälineaarisena funktiona. Tutkielmassa käytetään yhden piilokerroksen päättelintä. Neuroverkkoja kuten satunnaista metsää voidaan käyttää regressioon ja luokitteluun.

Olkoon Y_k dikotominen muuttuja, joka kertoo, kuuluuko yksilö luokkaan k, jossa $k = 1, ..., K$. Luokitellessa K luokkaa johdetut piirteet Z_m luodaan piirteiden $X = (X_1, ..., X_p)^T$ lineaarikombinaatioista, jonka jälkeen Y_k mallinnetaan Z_m:n lineaarikombinaationa seuraavasti (Hastie, et al. 2016):

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, ..., M,$$

$$T_k = \beta_{0k} + \beta_k^T Z, k = 1, ..., K,$$

$$f_k(X) = g_k(T), k = 1, ..., K,$$

missä $Z = (Z_1, Z_2, ..., Z_m)$, ja $T = (T_1, T_2, ..., T_K)$.

Aktivointifunktioksi $\sigma(v)$ valitaan sigmoidifunktio $\sigma(v) = 1/(1 + \exp^{-v})$ ja ulostulofunktioksi $softmax$- funktion $g_k(T) = \frac{e^{T_k}}{\Sigma_{l=1}^K e_l^T}$, mikä antaa ulos positiivisia estimaatteja, jotka summautuvat ykköseen. Ulostulofunktio tekee T:n viimeisen muunnoksen. Muuttujia Z_m kutsutaan piiloyksiköiksi, koska niitä ei havaita suoranaisesti. Mikäli aktivointifunktio σ on identiteettifunktio, muuttuu koko malli syötteissä lineaariseksi funktioksi. Siten neuroverkkoja voidaan ajatella lineaarisen mallin epälineaariseksi yleistykseksi sekä regressiossa, että luokittelussa.

Tässä tutkielmassa on käytetty R:n *nnet*-paketin funktiota *nnet* neuroverkoille (Venables, W. N & Ripley, B.D. 2022). Funktiolla voi myös automaattisesti valita korkeimman tarkkuuden antavan piiloyksiköiden määrän ja sakkotermin. Tutkielmassa käytetään yhden piilokerroksen neuroverkkoa, missä piiloyksiköiden määrä on 5 ja sakkotermi 0.1. Sakkotermin tehtävä on estää mallin ylisovittamista.

Myös neuroverkoissa varianssia voidaan pienentää ottamalla keskiarvo ennustustuloksista. On teoreettisesti osoitettu, ettei koosteen suorituskyky voi olla huonompi, kuin minkään yksittäisen mallin, jos yksittäisen luokittelijan ennusteet ovat harhattomia ja korreloimattomia. On myös havaittu, että pienen varianssin omaava virhe-estimaattori on yleisesti suotavempi, kuin harhaton estimaattori suurella varianssilla (Perrone et al., 1993).

2.4 Lähinaapurin menetelmä

K-lähimmän naapurin luokittelijat ovat epäparametrisia eivätkä ne tarvitse mallin sovittamista. Vaikka ne ovat helppoja sovittaa, ne voivat muuttua huomattavasti hitaiksi, kun aineiston koko kasvaa (Tran, 2020). Kun havaintoja on N-kappaletta, ja ennustavia muuttujia p-kappaletta, vaatii lähinaapurin menetelmä Np-kappaletta operaatiota löytääkseen kyselypisteen naapurit.

Yksinkertaisuudesta huolimatta lähinaapurin menetelmät ovat osoittautuneet tehokkaiksi monissa luokitteluongelmissa, kuten käsin kirjoitettujen numeroiden tunnistamisessa tai sateliittikuvien tunnistamisessa (Hastie, et al. 2016). Lähinaapurinmenetelmä on usein hyvä luokittelija päätösrajan ollessa hyvin epäsäännöllinen.

Ideana on havainnolle x_0 löytää lähimmät k opetuspistettä $x_{(r)}, r = 1, ..., k$ ja luokitella k:n lähimpien naapureiden antamien äänien enemmistöllä. Tasahavainnot voidaan ratkaista satunnaisesti (Bishop, 2006).

Etäisyydeksi tässä tutkielmassa valitaan euklidinen etäisyys $d_{(i)} = ||x_{(i)} - x_0||$ saman skaalan vuoksi. Tässä tutkielmassa k:n kiinteäksi arvoksi on valittu 7. Tutkimuksissa on osoitettu, että datan koon lähestyessä ääretöntä, $k = 1$-lähinaapurin luokitteluvirhe ole koskaan enemmän, kuin kaksikertainen Bayes-virheeseen verrattuna (Cover et al.,1967). K:n ollessa 1 luokitellaan piste yksinkertaisesti lähimpään pisteeseen. Tällöin sen harha on usein pieni, mutta varianssi suuri (Hastie, et al. 2016).

2.5 Tukivektorikoneet

Tukivektorikoneiden ideana on rakentaa optimaalinen suurimman marginaalin tuottava hypertaso. Tukivektorikoneiden huonona puolena ovat rajoittuminen binäärisiin luokitteluihin, sekä laskennallinen raskaus ja hitaus.(Karamizadeh, 2014). Koska dataa harvoin voi pelkällä suoralla erotella, sallitaan poikkeamat pehmeällä marginaalilla. Lisäksi tarvitaan sopiva ydinfunktio hypertason laskemiseksi, mikäli lineaarista päätösrajaa ei voida sovittaa.

Tarkoituksena tukivektorikoneissa on etsiä optimaalinen erottava hypertaso siten, että luokat rajataan mahdollisimman kauas toisistaan. Kahden hypertason rajaamaa aluetta sanotaan marginaaliksi M. Määritellään hypertaso

$$x : f(x) = x^T\beta + \beta_0 = 0,$$

missä β on yksikkövektori $||\beta|| = 1$. Nyt luokittelusääntönä on

$$G(x) = sign[x^T\beta + \beta_0]$$

(Hastie, et al. 2016). Kun luokat ovat eroteltavissa voidaan etsiä funktio $f(x) = x^T\beta + \beta_0$, missä $y_i f(x_i) > 0 \forall i$.

Voidaan siis löytää hypertaso, joka luo isoimman marginaalin M luokan 1 ja luokan -1 välillä. Kyseessä on optimointiongelma

$$max_{\beta,\beta_0,||\beta||=1} M \; s.e.$$

$$y_i(x_i^T\beta + \beta_0) \geq M, i = 1, ..., N.$$

Asettamalla $M = 1/||B||$ päästään minimointiongelmaan

$$min_{\beta,\beta_0}||\beta|| \; s.e.$$

$$y_i(x_i^T\beta + \beta_0) \geq 1, = 1, ..., N.$$

Kun luokat erottuvat lineaarisesti, on kyse kovasta marginaalista ja vastaavasti, kun luokat eivät erotu lineaarisesti, on kyse pehmeästä marginaalista.

Pehmeässä marginaalissa halutaan maksimoida M, mutta sallitaan luokkien osittainen päällekkyys $\xi = (\xi_1, \xi_2, ..., \xi_N)$.

Nyt rajoitteena on

$$y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i).$$

Koska $M = 1/||B||$ tapauksessa rajoitteeksi saadaan

$$min_{\beta,\beta_0}||\beta|| \; s.e.$$

$$\begin{cases} y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, \forall i = 1, .., N \\ \xi_i \geq 0, \Sigma\xi_i \leq C \end{cases} \tag{1}$$

missä C on vakio.

Rajoitteen perusteella luokan rajojen sisällä olevat pisteet eivät ole isossa osassa rajoitteen muodostamisessa (Hastie, et al. 2016).

Koska (1) on konveksi optimointiongelma, on laskennallisesti hyödyllistä ilmaista se muodossa

$$\begin{cases} min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 + C\Sigma_{i=1}^N \xi_i \; s.e. \\ \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \leq 1 - \xi_i \forall i. \end{cases} \tag{2}$$

Lagrangen muunnoksella päästään ratkaisuun

$$\hat{\beta} = \Sigma_{i=1}^N \hat{\alpha_i} y_i x_i,$$

jossa $\hat{\alpha_i} \neq 0$ tukivektoreille, jotka täyttävät ehdot

$$\begin{cases} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] \geq 0 \\ \mu_i\xi_i = 0 \\ y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) \geq 0. \end{cases} \tag{3}$$

Lisäksi asetetaan rajoitteet $0 \leq \alpha_i \leq C$ sekä $\Sigma_{i=1}^{N}\alpha_i y_i = 0$.

Ehdoista (3) ratkaistaan β_0 ja tyypillisesti otetaan keskiarvo ratkaisuista numeerisen stabiiliuden vuoksi.

Lopullinen luokittelusääntö on nyt

$$\hat{G}(x) = sign[x^T\hat{\beta} + \beta_0].$$

Kun lineaarinen päätösraja ei toimi, tarvitaan muutosfunktio $h(x)$ ja sitä varten tarvitaan ydin $K = \langle h(x), h(x_i)\rangle$ hypertason laskemiseksi. Ytimiksi tässä tutkielmassa on valittu lineaarinen ydin $x^T x_i$ ja sädeperusteinen ydin $\exp(-\gamma||x - x_i||^2)$.

Tukivektorikoneet ovat yleensä vain $k = 2$ luokan luokittelua varten, mutta niillä voidaan myös luokitella useampia luokkia. R:n *e1071*-kirjaston funktio *svm* luokittelee $k > 2$ luokkaa siten, että kehitetään $k(k-1)/2$ binääristä luokittelijaa, joita vertaillaan pareittain ja joista eniten ääniä saanut luokka valitaan luokaksi (Meyer, et al. 2022).

3 Evaluointimenetelmät

Tutkielmassa tarkastellaan eri evaluointimentelmien tuottamia virheitä ja näitä vertaillaan yleistämisvirheeseen Err_O, mikä on riippumattoman testiaineiston odotettu ennustevirhe kiinteälle opetusaineistolle O (Moisio, 2016). Ottamalla keskiarvon yleistämisvirheestä saadaan estimaatti odotetulle yleistämisvirheelle Err. Lisäksi tuloksia vertaillaan Bayes-virheeseen.

Aiemmissa tutkimuksissa $K = 10$ ristiinvalidoinnin suorituskyky on havaittu .632+ estimaattoria paremmaksi (Ji-Hyun, 2009). Tässä tutkimuksessa kiinnostuksen kohteena ovat estimaattorit $\bar{Err}$, $\hat{Err}_{boot}$, $\hat{Err}^{(1)}$, $\hat{Err}^{(632.)}$, $\hat{Err}^{(.632+)}$, Err, ristiinvalidointi arvoilla $K = 2, 3, 5, 10, N$, toistettu ristiinvalidointi arvoilla $K = 5, K = 10$, sekä toistuva jako opetus- ja testiaineistoksi suhteilla 50/50, 80/20, 90/10.

Olkoon R_1 ja R_2 piirreavaruuden alueita, joille muuttuja x on jakautunut. Piirreavaruudessa R_1 luokittelijan päätös on w_1 ja vastaavasti R_2:ssa w_2. Bayes-virhe on pienin mahdollinen luokitteluvirheen todennäköisyys, mikä määritellään kahden luokan tapauksessa seuraavasti (Duda, et al. 2000):

$$\begin{aligned} P(virhe) &= P(x \in R_2, w_1) + P(x \in R_1, w_2) \\ &= P(x \in R_2|w_1)P(w_1) + P(x \in R_1|w_2)P(w_2) \\ &= \int_{R_2} p(x|w_1)P(w_1)dx + \int_{R_1} p(x|w_2)P(w_2)dx. \end{aligned}$$

Virhe voi siis tapahtua kahdella tapaa, joko havainto x on R_2:ssa, ja oikea arvo on w_1, tai x on R_1:ssä ja oikea arvo on w_2. Bayes-virhe siis asettaa alarajan yleistämisvirheelle.

Bayes-virhe lasketaan tutkielmassa Monte-Carlo integroinnilla analyyttisen hankaluuden takia. Merkitään α_1 toiminnoksi, kun havainto luokitellaan luokkaan w_1 ja α_2 toiminnoksi, kun havainto luokitellaan luokkaan w_2. Nyt päätössääntöfunktion ollessa $\alpha(x)$ ja 0-1-tappiofunktion $\lambda(\alpha(x))|w_k)$, voidaan virheen todennäköisyys esittää kaavalla (Moisio, 2016)

$$P(virhe) = \int_{R_1} \lambda(\alpha(x)|w_2)P(w_2)p(x|w_2)dx + \int_{R_2} \lambda(\alpha(x)|w_1)P(w_1)p(x|w_1)dx.$$

Nyt $\alpha(x) = \alpha_1$, kun $x \in R_1$, jolloin $\lambda(\alpha(x)|w_2) = 1$, muutoin 0. Vastaavasti $\alpha(x) = \alpha_2$, kun $x \in R_2$, jolloin $\lambda(\alpha(x)|w_1) = 1$, muutoin 0.

Integroimalla yli koko reaaliakselin $\mathbb{R}$ kaavasta saadaan

$$P(virhe) = \int_{\mathbb{R}} \Sigma_{k=1}^{2} \lambda(\alpha(x))|w_k)P(w_k)p(x|w_k)dx.$$

Nyt suurten lukujen lain nojalla integraali voidaan korvata otoksesta lasketulla empiirisellä keskiarvolla

$$\hat{\mu} = \frac{1}{m}\Sigma_{j=1}^{m} \lambda(\alpha(x_j)|w_{k_j}),$$

missä m on simuloitujen havaintojen lukumäärä, j luokka ja $\alpha(x_j)$ tunnettu tai estimoitu päätösfunktio. Lisäksi $\lambda(\alpha(x_j)|w_{k_j}) = 1$, kun havainto luokitellaan väärin, ja nolla muutoin.

Usean luokkien K ja piirteiden p tapauksessa virhettä voidaan arvioida kaavalla

$$P(virhe) = \int_{\mathbb{R}} ... \int_{\mathbb{R}} \Sigma_{k=1}^{K} \lambda(\alpha(x)|w_k)P(w_k)p(x|w_k)dx_1...dx_p,$$

jossa $x \in \mathbb{R}^p$ ja K on luokkien määrä (Moisio, 2016).

Lisäksi lasketaan estimaattoreille keskineliövirhe ja keskihajonta. Keskineliövirhe saadaan harha-varianssihajotelmalla

$$MSE(\hat{X}) = Var(\hat{X}) + (Bias(\hat{X}))^2,$$

missä X on muuttuja ja satunnaismuuttuja $\hat{X}$ sen estimaatti.

3.1 Uusio-otanta

Uusio-otantaa käytetään paljon tilastollisen tarkkuuden arvioimiseen. Uusio-otanta estimoi suoraan odotettua yleistämisvirhettä ristiinvalidoinnin kanssa. Ideana on satunnaisesti poimia otoskoon kokoisia otoksia opetusaineistosta takaisinpalauttaen B kertaa ja laskea tilastollinen tarkkuus. Tässä tutkielmassa uusio-otantojen lukumäärä on kiinnitetty $B = 50$ kaikissa tapauksissa. Eräs tapa on sovittaa malli uusio-otantaotoksiin, ja katsoa, kuinka hyvin se ennustaa alkuperäistä opetusdataa

$$\hat{Err}_{boot} = \frac{1}{B}\frac{1}{N}\Sigma_{b=1}^{B}\Sigma_{j=1}^{N}\lambda(\hat{\alpha}^{*b}(x_j|w_{k_j}))$$

missä $\hat{\alpha}^{*b}(x_j|w_{k_j})$ on piirteiden x_j arvoilla bootstrap otoksen b ennustettu luokka (Moisio, 2016). Estimaatin $\hat{Err}_{boot}$ ajatuksena on soveltaa uusio-otantaa ennustevirheen estimointiin. Malli sovitetaan uusio-otannalla saatuun näytejoukkoon ja pidetään kirjaa siitä, kuinka hyvin malli ennustaa alkuperäistä opetusaineistoa. Koska uusio-otannassa poimitaan otokset takaisinpalauttaen, tulee samoja otoksia opetus- ja testidataan ja siten $\hat{Err}_{boot}$ ei yleisesti anna hyviä estimaatteja, sillä uusio-otannalla saatu data toimii opetusdatana ja alkuperäinen opetusdata toimii testidatana. Koska näillä otoksilla on yhteisiä havaintoja, voivat ennustukset olla epärealistisen hyviä (Hastie, et al. 2016)

Parempana vaihtoehtona on yksittäisuusio-otanta.

$$\hat{Err}^{(1)} = \frac{1}{N}\Sigma_{j=1}^{N}\frac{1}{|C^{-j}|}\Sigma_{b\in C^{-j}}\lambda(\hat{\alpha}^{*b}(x_j|w_{kj})),$$

missä C^{-j} ovat indeksit niistä uusio-otannan otoksista b, jotka eivät sisällä havaintoa j ja $|C^{-j}|$ niiden otosten lukumäärä. Kyseinen validointitapa ei ylisovitu $\hat{Err}_{boot}$ tapaan, mutta sisältää harhaa.

Estimaattorin $\hat{Err}^{(1)}$ harhan korjaukseksi ehdotetaan $\hat{Err}^{(.632)}$ estimaattoria (Hastie, et al. 2016)

$$\hat{Err}^{(.632)} = .368 \cdot \bar{err} + .632 \cdot \hat{Err}^{(1)}.$$

Tässä $\bar{err}$ on opetusvirhe $\bar{err} = \frac{1}{N}\Sigma_{j=1}^{N}\lambda(\alpha(x_j|w_{kj}))$, mikä on opetusaineiston keskimääräinen tappio.

$\hat{Err}^{(.632)}$ alentaa yksittäisuusio-otannan estimaattia opetusvirhettä kohti vähentäen harhaa. $\hat{Err}^{(.632)}$ estimaattoria voidaan vielä parantaa ottamalla ylisovituksen määrän huomioon, ja päästään $\hat{Err}^{(.632+)}$ estimaattoriin.

$\hat{Err}^{(.632+)}$ estimaattori saadaan kaavalla

$$\hat{Err}^{(.632+)} = (1 - \hat{w}) \cdot \bar{err} + \hat{w} \cdot \hat{Err}^{(1)},$$

missä $\hat{w} = \frac{.632}{1-.368\hat{R}}$.

Tätä varten tarvitsee laskea suhteellinen ylisovittamisen osuus

$$\hat{R} = \frac{\hat{Err}^{(1)} - e\bar{r}r}{\hat{\gamma} - e\bar{r}r}.$$

Suhteellinen ylisovittamisen osuus $\hat{R}$ vaihtelee välillä [0,1]. Kun ylisovittamista ei ole $\hat{R} = 0$ eli $\hat{Err}^{(1)} = e\bar{r}r$.

$\hat{\gamma}$ on ei-tiedon virhemäärä, ja se on virheen todennäköisyyden estimaatti tilanteessa, jossa luokat ja piirteet olisivat riippumattomia (Moisio, 2016).

3.2 Toistuva jako opetus- ja testiaineistoksi

Toistuvassa jaossa opetus-ja testiaineistoksi data jaetaan ennalta määrättyyn jakosuhteeseen p. Esimerkiksi $p = 1/3$ jakaa aineiston 1/3 testiaineistoksi ja 2/3 opetusaineistoksi (Molinaro et al., 2005). Menetelmän etuina on laskennallinen helppous. Harhaa voi kuitenkin syntyä, sillä jokainen havainto kuuluu vain joko testi- tai opetusaineistoon, sekä harhaa pienestä opetusaineistosta. Koska opetusaineisto on koko aineistoa pienempi, yliestimoi opetusaineistoon perustuvan mallin testivirhe koko aineistoon rakennetun mallin yleistämisvirhettä. Tutkielmassa aineisto jaetaan opetus- ja testiaineistoksi 50-kertaa jakosuhteilla 1/2, 4/5 ja 9/10. Näin saadaan ennustevirheiden keskiarvosta estimaatti yleistämisvirheelle (Moisio, 2016).

3.3 Ristiinvalidointi

Ristiinvalidointia käytetään estimoimaan testivirhettä suoraan ilman validointidataa. K-kertaisessa ristiinvalidoinnissa data jaetaan K:hon osaan, ja $K-1$ osaan sovitetaan malli, minkä jälkeen lasketaan ennustevirhe pois jätetyllä osalla. Prosessia toistetaan $k = 1, 2, ..., K$ kertaa ja lopulta yhdistetään ennustevirheet.

Ennustetun virheen estimaatti saadaan kaavalla

$$CV = \frac{1}{N}\Sigma_{j=1}^{n}\lambda(\hat{\alpha}^{-K(j)}(x_j|w_{k_j})),$$

missä $K : 1, ..., N \mapsto 1, ..., K$ on indeksifunktio, mikä kertoo mihin jakoon havainto j on päätynyt satunnaistamisessa ja $\alpha^{-K(j)}(x_j)$ on sovitettu malli, josta on poistettu k:s osa (Moisio, 2016).

Tyypilliset K:n arvot ovat 5 ja 10. Vaihtoehtoisesti käytetään myös $K = N$, jolloin harha on alhainen mutta varianssi voi olla suurempi, koska opetusdatat ovat samanlaisia. (Hastie, et al. 2016). Varianssin pienentämiseksi ristiinvalidointia voi toistaa T kertaa, minkä jälkeen lasketaan estimaattien keskiarvo. Tyypillisesti suositellut arvot K:lle ovat 5 tai 10-kertainen ristiinvalidointi, vaikka laskenta riittäisikin isompiinkin K arvoihin (Breiman, L. 1992). Luokittelumenetelmiä ja evaluointimenetelmiä varten tutkielmassa käytetään R:n *caret*-pakettia (Kuhn M, 2022).

4 Simulaatiokoe

Tavoitteena on simuloida aineiston pohjalta yksilöitä, joille arvotaan luokka ja kiinnitetty määrä piirteitä.

Aineisto sisältää 6966 havaintoa ja 50 eri luokkaa. Piirteitä on aineistossa 64 kappaletta, joihin kuuluu mm. pohjaeläinten kuvista laskettuja harmaasävyn eri tunnuslukuja kuten keskiarvo, moodi, sekä geometrisiä ja RGB-sävyjen tunnuslukuja (Joutsijoki, et al. 2012). Suurin luokka aineistossa on Taenneb, missä on 633 havaintoa ja pienin Agapetus, missä on 24 havaintoa.

Aineiston pohjalta simuloidaan piirteitä normaalijakaumasta, jossa parametrit μ_k ja Σ_k ovat estimoitu pohjaeläinaineistosta. Kvadraattinen luokittelija on optimaalinen luokittelumenetelmä normaalijakaumien sekoitukselle. R:llä luodaan funktio, joka estimoi aineistosta luokkien prioritodennäköisyydet, keskiarvovektorit ja kovarianssimatriisit, minkä jälkeen lasketaan luokittelupistemäärää varten luokan k kovarianssimatriisin käänteismatriisi sekä determinantti. Nyt voidaan arpoa yksilölle luokka, piirteet ja luokan luokittelupistemäärä. Huomioitavaa on, että oletuksesta johtuen pirteet voivat saada myös negatiivisia arvoja.

Simuloinnissa luokkien ja piirteiden lukumäärä vaihtelee kolmessa tapauksessa. Ensimmäisessä tapauksessa piirteitä ja luokkia on kaksi, ja toisessa tapauksessa luokkia ja piirteitä on kahdeksan. Kolmannessa tapauksessa luokkia on 50 ja piirteitä kahdeksan. Kun mallista tulee monimutkaisempi, se käyttää opetusdataa enemmän ja varianssi pienenee, mutta malli ylisovittuu (Hastie, et al. 2016). Kun luokkia ja piirteitä on molempia 2, käytetään otoskokona *N=50*. Kahdeksan luokan ja piirteen tapauksessa otoskokona on *N=500*. 50 luokalle ja 8 piirteelle käytetään otoskokoa *N=10000*. Simuloitavien aineistojen lukumäärä on asetettu *N=100* kaikilla otoskokoilla. Isoimmat luokat ja parhaiten erottavat piirteet valitaan *stepclass* funktiolla (Weihs, et al. 2016), jossa askelettain lisätään muuttuja kerrallaan malliin, estimoidaan suorituskyky ristiinvalidoinnilla ja lasketaan oikeellisuus arvo $1 - \textit{virheiden osuus}$, minkä perusteella katsotaan, jätetäänkö muuttuja malliin.

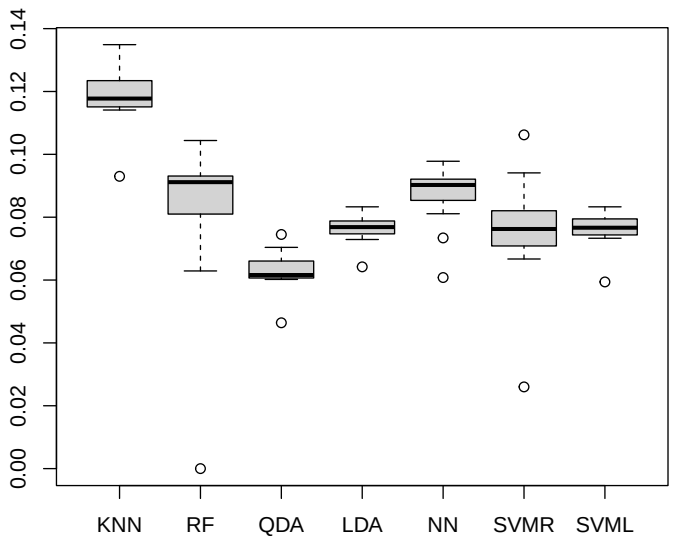
Tarkastelussa halutaan estimoida yleistämisvirhettä, joka on riippumattoman testiotoksen ennustevirhe.

5 Tuloksia

Tulokset saatiin 2 luokan ja 2 piirteen tapauksessa kaikille luokittelijoille, sekä myös 8 luokan ja 8 tapauksessa. Laskennallisen raskauden vuoksi 10000 aineiston, 50 luokan ja 8 piirteiden tapauksessa tuloksia saatiin vain satunnaismetsästä, kvadraattisesta ja lineaarisesta luokittelijasta ja lähinaapurimenetelmästä.

5.1 2 luokkaa ja 2 piirrettä, N=50

2 luokan tapauksessa valittiin pohjaeläinaineiston 2 isointa luokkaa Taenneb ja Isoperla. Parhaiten erottaviksi piirteiksi saatiin "Mode.green"ja "YM.gray". Vertaillessa eri menetelmiä, suurimmat virhe-ennusteet saivat lähimmän naapurin menetelmä ja alhaisimmat kvadraattinen luokittelija (Kuva 1).



Kuva 1: Ennustevirheet eri menetelmillä, kun 2 luokkaa ja 2 piirrettä. Kuvassa KNN = Lähinaapurimenetelmä, RF = Satunnaismetsä, QDA = Kvadraattinen luokittelija, LDA = Lineaarinen luokittelija, NN= Neuroverkot, SVMR = Tukivektorikone (sädeperusteinen ydin), SVML = Tukivektorikone (lineaarinen ydin).

Lähinaapurin tapauksessa lähimpänä yleistämisvirhettä oli $K = N$ ristiinvalidointi. Kaikki virheet olivat kuitenkin suhteellisen kaukana Bayes-virheestä (Taulukko 1). Lähimpänä Bayes-virhettä oli yleistämisvirhettä aliestimoiva $\hat{Err}^{(.632)}$

Satunnaismetsän tapauksessa 632 ja 632+ estimaattorit suoriutuivat parhaiten. Huonoiten suoriutui $Err^{(1)}$, joka oli lähes puolet Bayes-virheestä. Lisäksi virheillä oli suuri vaihteluvuus (Taulukko 2).

Kvadraattinen luokittelija sai pienimmät virhe-ennusteet toistetulla 10-kertaisella ristiinvalidoinnilla, joka kuitenkin aliestimoi testivirhettä. Pienillä otoskoilla on suositeltavaa verrata odotettuun yleistämisvirheeseen, sillä se ottaa aineiston koon huomioon (Moisio, 2016). Toisaalta kaikista pienin virhe oli opetusvirhe, mutta siinä toimii opetusaineisto sekä testissä, että opetuksessa, mikä aiheuttaa aliestimointia, eikä sitä voida pitää hyvänä estimaattina. Lähimpänä yleistämisvirhettä on $K = 10$-kertainen ristiinvalidointi (Taulukko 3). Virhe oli suurimmillaan, kun aineisto jaettiin toistetusti opetus- ja testiaineistoon 50 %:n jaolla.

Lineaarisen luokittelijan tapauksessa lähimpänä Bayes-virhettä oli $K = 10$ ristiinvalidointi. Muuten vaihtelu oli pientä. Lähimpänä yleistämisvirhettä oli taas $K = N$ ristiinvalidointi (Taulukko 4).

Neuroverkkojen tapauksessa aineiston toistuva jako 50/50-suhteella pääsi lähelle yleistämisvirhettä, ja kaikki muut aliestimoivat yleistämisvirhettä. Tässä tapauksessa suurin virhe oli yleistämisvirheellä, mikä myös erosi paljon Bayes-virheestä (Taulukko 5).

Tukivektorikoneella sädeperusteisella ytimellä vaihtelu oli lineaarista ydintä suurempaa ja 90/10-jako antoi tuloksen lähimpänä yleistämisvirhettä. Toisaalta lähimpänä Bayes-virhettä olivat 632 ja 632+ estimaattorit. (Taulukko 6). Lineaarisella ytimellä oli vaihtelu pientä, ja lähimpänä yleistämisvirhettä oli toistettu ristiinvalidointi arvolla $K = 5$ ja $K = 10$ (Taulukko 7). Pienin virhe saatiin ristiinvalidoinnilla jaolla $K = 2$.

Taulukko 1: Lähinaapurin virheet, 2 piirrettä 2 luokkaa, N=50.

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.0930	0.0340	0.00278
$\hat{Err}^{(.632)}$	0.1141	0.0365	0.0051
$CV\ (K = 10)$	0.1148	0.0447	0.00586
$RCV\ (K = 10)$	0.1150	0.0419	0.00564
$CV\ (K = N)$	0.1152	0.0446	0.00589
Err	0.1160	0.0147	0.00422
90/10	0.1172	0.0460	0.00627
$\hat{Err}^{(.632+)}$	0.1176	0.0374	0.00561
$RCV\ (K = 5)$	0.1179	0.0397	0.00582
$CV\ (K = 3)$	0.1193	0.0419	0.00619
80/20	0.1201	0.0444	0.00651
$CV\ (K = 5)$	0.1204	0.0428	0.00641
$\hat{Err}^{(1)}$	0.1265	0.0409	0.00711
$CV\ (K = 2)$	0.1277	0.0478	0.00791
$\hat{Err}_{boot}$	0.1300	0.0409	0.00765
50/50	0.1349	0.0421	0.00852
P(virhe)	0.0527	7e-5	

Taulukko 2: Satunnaismetsän virheet, 2 piirrettä 2 luokkaa, N=50.

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0	0	0.00278
$\hat{Err}^{(.632+)}$	0.0629	0.0271	0.00084
$\hat{Err}^{(.632)}$	0.0660	0.0221	0.00067
Err	0.0788	0.0156	0.00093
90/10	0.0832	0.0434	0.00281
80/20	0.0851	0.0409	0.00272
$RCV\ (K = 10)$	0.0907	0.0393	0.00299
$CV\ (K = 3)$	0.0911	0.0445	0.00345
$CV\ (K = 10)$	0.0912	0.0415	0.0032
$\hat{Err}_{boot}$	0.0914	0.0371	0.00287
50/50	0.0917	0.0351	0.00276
$CV\ (K = N)$	0.0922	0.0422	0.00334
$RCV\ (K = 5)$	0.0940	0.0371	0.00308
$CV\ (K = 5)$	0.0944	0.0422	0.00352
$CV\ (K = 2)$	0.1029	0.0493	0.00495
$\hat{Err}^{(1)}$	0.1044	0.0350	0.0039
P(virhe)	0.0527	7e-5	

Taulukko 3: Kvadraattisen luokittelijan virheet, 2 piirrettä 2 luokkaa, N=50.

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.0464	0.0307	0.00098
$RCV\ (K = 10)$	0.0602	0.0319	0.00107
$CV\ (K = N)$	0.0602	0.0328	0.00113
$RCV\ (K = 5)$	0.0606	0.0310	0.00102
$CV\ (K = 3)$	0.0607	0.0304	0.00099
$\hat{Err}^{(.632)}$	0.0609	0.0293	0.00093
Err	0.0613	0.0084	0.00014
$CV\ (K = 10)$	0.0615	0.0338	0.00122
90/10	0.0617	0.0373	0.00147
$CV\ (K = 5)$	0.0619	0.0350	0.00131
$\hat{Err}^{(.632+)}$	0.0620	0.0290	0.00093
80/20	0.0628	0.0323	0.00115
$\hat{Err}^{(1)}$	0.0693	0.0302	0.00119
$CV\ (K = 2)$	0.0703	0.0389	0.00182
$\hat{Err}_{boot}$	0.0704	0.0293	0.00117
50/50	0.0745	0.0293	0.00133
P(virhe)	0.0527	7e-5	

Taulukko 4: Lineaarisen luokittelijan virheet, 2 piirrettä 2 luokkaa, N=50.

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.0642	0.0326	0.00119
$CV\ (K = 10)$	0.0729	0.0356	0.00167
$RCV\ (K = 5)$	0.0746	0.0346	0.00168
$CV\ (K = 5)$	0.0747	0.0374	0.00189
$RCV\ (K = 10)$	0.0747	0.0356	0.00176
$\hat{Err}^{(.632)}$	0.0750	0.0315	0.00149
$\hat{Err}^{(.632+)}$	0.0756	0.0318	0.00154
Err	0.0765	0.0100	0.00067
$CV\ (K = N)$	0.0772	0.0376	0.00201
$CV\ (K = 2)$	0.0785	0.0369	0.00203
$CV\ (K = 3)$	0.0785	0.0366	0.00200
90/10	0.0787	0.0425	0.00248
80/20	0.0789	0.0380	0.00213
$\hat{Err}^{(1)}$	0.0812	0.0323	0.00186
$\hat{Err}_{boot}$	0.0818	0.0325	0.00190
50/50	0.0833	0.0334	0.00205
P(virhe)	0.0527	7e-5	

Taulukko 5: Neuroverkkojen virheet, 2 piirrettä 2 luokkaa, N=50.

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.0608	0.0757	0.00579
$\hat{Err}^{(.632+)}$	0.0734	0.0338	0.00157
$\hat{Err}^{(.632)}$	0.0811	0.0517	0.00348
$CV\ (K = 2)$	0.0831	0.0436	0.00282
$RCV\ (K = 10)$	0.0876	0.0437	0.00313
$\hat{Err}_{boot}$	0.0889	0.0375	0.00271
$CV\ (K = N)$	0.0898	0.0477	0.00365
$RCV\ (K = 5)$	0.0900	0.0450	0.00341
$CV\ (K = 10)$	0.0905	0.0577	0.00475
90/10	0.0906	0.0523	0.00417
$CV\ (K = 5)$	0.0912	0.0528	0.00426
$CV\ (K = 3)$	0.0918	0.0600	0.00513
80/20	0.0924	0.0486	0.00394
$\hat{Err}^{(1)}$	0.0928	0.0614	0.00537
50/50	0.0959	0.0453	0.00392
Err	0.0978	0.0806	0.00853
P(virhe)	0.0527	7e-5	

Taulukko 6: Tukivektorikoneen (sädeperusteinen ydin) virheet, 2 piirrettä 2 luokkaa, N=50.

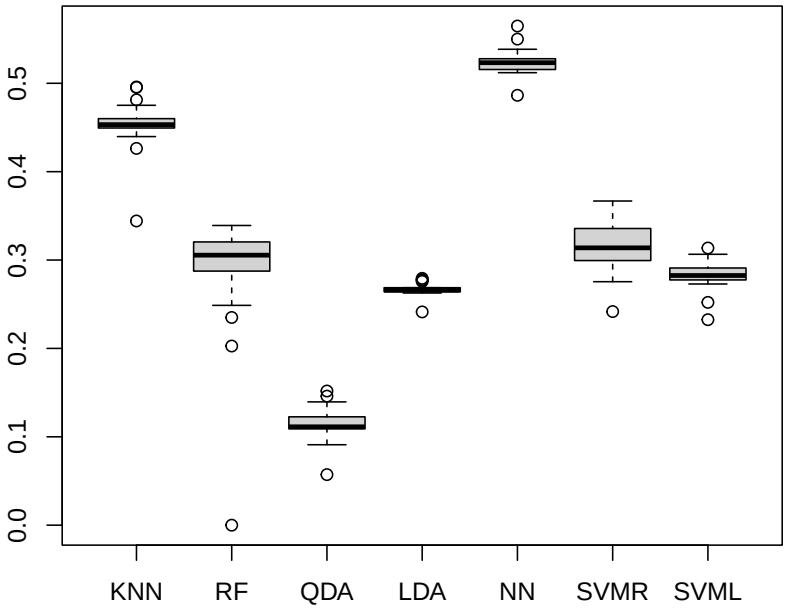
method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.0260	0.0216	0.00118
$\hat{Err}^{(.632+)}$	0.0667	0.0304	0.00112
$\hat{Err}^{(.632)}$	0.0690	0.0298	0.00116
$CV\ (K = 10)$	0.0698	0.0344	0.00148
$CV\ (K = 5)$	0.0719	0.0344	0.00156
$CV\ (K = N)$	0.0719	0.0369	0.00173
$CV\ (K = 3)$	0.0740	0.0368	0.0018
$RCV\ (K = 10)$	0.0762	0.0389	0.00207
$\hat{Err}_{boot}$	0.0763	0.0335	0.00168
$RCV\ (K = 5)$	0.0779	0.0376	0.00205
Err	0.0793	0.0200	0.00111
90/10	0.0799	0.0453	0.0028
80/20	0.0842	0.0417	0.00273
$CV\ (K = 2)$	0.0874	0.0483	0.00354
$\hat{Err}^{(1)}$	0.0941	0.0370	0.00308
50/50	0.1062	0.0384	0.00434
P(virhe)	0.0527	7e-5	

Taulukko 7: Tukivektorikoneen (lineaarinen ydin) virheet, 2 piirrettä 2 luokkaa, N=50.

method	$\widehat{Err}$	SD	MSE
$e\bar{r}r$	0.0594	0.0333	0.00116
$CV\ (K=2)$	0.0733	0.0374	0.00182
$CV\ (K=3)$	0.0738	0.0370	0.00181
$\hat{Err}^{(.632)}$	0.0738	0.0339	0.00159
$\hat{Err}^{(.632+)}$	0.0749	0.0341	0.00165
$CV\ (K=5)$	0.0753	0.0404	0.00214
$CV\ (K=10)$	0.0759	0.0388	0.00204
$RCV\ (K=5)$	0.0766	0.0361	0.00187
Err	0.0767	0.0099	0.00067
$RCV\ (K=10)$	0.0768	0.0381	0.00203
$CV\ (K=N)$	0.0786	0.0420	0.00243
80/20	0.0793	0.0393	0.00225
90/10	0.0796	0.0450	0.00275
$\hat{Err}^{(1)}$	0.0821	0.0355	0.00212
$\hat{Err}_{boot}$	0.0830	0.0355	0.00218
50/50	0.0833	0.0355	0.00219
P(virhe)	0.0527	7e-5	

5.2 8 luokkaa ja 8 piirrettä, N=500

8 luokan tapauksessa luokiksi valittiin 8 suurinta luokkaa. Näihin kuuluivat Aselaqua, Baetmuti, Diura, Isoperla, Micrseti, Nemoura, Protintr ja Taenneb. Parhaiten erottaviksi piirteiksi saatiin "Minor","Mean.green.", "Mean.gray.", "Mean.blue.", "Major", "Solidity", "Median.red.", ja "Min.green.". Kun simuloituja luokkia on 8 ja piirteitä 8, sai kvadraattinen luokittelija taas pienemmät virhe-ennusteet kuin muut menetelmät. Mallin monimutkaistuessa myös luokittelijoiden virheet kasvavat. Suurimmat virheet tulivat neuroverkoilla ja sen jälkeen lähinaapurin menetelmällä (Kuva 2). Lineaarisen luokittelijan virheiden vaihtelu oli kuitenkin muita huomattavasti pienempää.



Kuva 2: Ennustevirheet eri menetelmillä, kun 100 aineistoa, 8 luokkaa ja 8 piirrettä.

Lähinaapurin kohdalla yleistämisvirhe sai alhaisimman tuloksen, joka on myös lähimpänä Bayes-virhettä, vaikkakin silti suuri siihen verrattuna. Lähimmäksi yleistämisvirhettä pääsi $\hat{Err}^{(.632)}$ Huonoiten suoriutui opetusaineiston toistuva jako opetus- ja testiaineistoksi 50/50 jaolla. Satunnaismetsän kohdalla on taas nähtävissä ylisovittumista opetusvirheessä. Pienin virhe oli $\hat{Err}^{.632}$:lla, mutta sekä $\hat{Err}^{.632}$ ja $\hat{Err}^{.632+}$ aliestimoivat yleistämisvirhettä. Lähimmäksi yleistämisvirhettä pääsi toistuva jako opetus- ja testiaineistoon jaolla 90/10.

Kvadraattisen luokittelijankin tapauksessa testivirhe Err lähestyy Bayes-virhettä otoskoon kasvaessa (Taulukko 10). Alhaisimman virheen sai yleistämisvirhe, jonka jälkeen ristiinvalidointi $K{=}N$-jaolla pääsi lähimmäksi.

Lineaarisessa luokittelijassa virheet olivat noin kolmenkertaisia Bayes-virheeseen verrattuna, mutta vaihtelu oli hyvin pientä. $\hat{Err}^{.632}$ sai alhaisimman virheen ja lähimmäksi yleistämisvirhettä pääsi toistuva jako opetus- ja testiaineistoksi 90/10 jaolla (Taulukko 11).

Neuroverkkojen virheet olivat kaikista suurimpia. Ristiinvalidointi arvolla K=10 sai pienimmän virheen, vaikkakin se aliestimoi yleistämisvirhettä. Lähimmäksi yleistämisvirhettä pääsi $\hat{Err}^{.632}$ (Taulukko 12).

Tukivektorikone sädeperusteisella ytimellä suoriutui lineaarista ydintä huonommin (Taulukko 13). Yleistysvirhe sai alhaisimman virheen, ja lähimmäksi yleistämisvirhettä pääsi taas toistuva jako opetus- ja testiaineistoksi 90/10 jaolla. $\hat{Err}^{.632+}$ aliestimoi yleistämisvirhettä. Isoimman virheen sai $\hat{Err}^{(1)}$. Lineaarisella ytimellä toistettu ristiinvalidointi arvolla $K = 10$ pääsi lähimmäksi yleistämisvirhettä (Taulukko 14). Tässä tapauksessa puolestaan toistuva jako testi- ja opetusaineistoksi jaolla 50/50 sai suurimman virheen.

Taulukko 8: Lähinaapurin virheet, 8 luokkaa 8, piirrettä, N=500.

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.4137	0.0205	0.11303
Err	0.4562	0.0082	0.143
$\hat{Err}^{(.632)}$	0.4564	0.0191	0.14342
$RCV\ (K = 10)$	0.4579	0.0212	0.14471
$CV\ (K = 10)$	0.4580	0.0233	0.14485
$CV\ (K = N)$	0.4586	0.0244	0.14539
$\hat{Err}^{(.632+)}$	0.4601	0.0200	0.14627
$CV\ (K = 5)$	0.4613	0.0235	0.1474
$RCV\ (K = 5)$	0.4634	0.0212	0.14885
90/10	0.4651	0.0244	0.15035
80/20	0.4695	0.0219	0.15365
$CV\ (K = 3)$	0.4706	0.0251	0.15463
$\hat{Err}^{(1)}$	0.4812	0.0198	0.16285
$\hat{Err}_{boot}$	0.4824	0.0205	0.16384
$CV\ (K = 2)$	0.4863	0.0248	0.16721
50/50	0.4976	0.0196	0.17637
P(virhe)	0.0781	8.5e-5	

Taulukko 9: Satunnaismetsän virheet, 8 luokkaa 8, piirrettä, N=500.

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.0000	0	0.00278
$\hat{Err}^{(.632)}$	0.2027	0.0113	0.02262
$\hat{Err}^{(.632+)}$	0.2350	0.0147	0.03346
Err	0.2875	0.0086	0.05521
90/10	0.2971	0.0191	0.06009
80/20	0.3024	0.0190	0.06272
$CV\ (K = 10)$	0.3055	0.0205	0.06434
$CV\ (K = N)$	0.3055	0.0190	0.06428
$RCV\ (K = 10)$	0.3071	0.0188	0.06505
$CV\ (K = 5)$	0.3131	0.0199	0.06821
$RCV\ (K = 5)$	0.3138	0.0184	0.06852
$\hat{Err}_{boot}$	0.3205	0.0173	0.07201
$\hat{Err}^{(1)}$	0.3207	0.0178	0.07213
$CV\ (K = 3)$	0.3225	0.0217	0.07323
50/50	0.3360	0.0173	0.08055
$CV\ (K = 2)$	0.3391	0.0214	0.08249
P(virhe)	0.0781	8.5e-5	

Taulukko 10: Kvadraattisen luokittelijan virheet, 8 luokkaa 8, piirrettä, N=500.

method	$\widehat{Err}$	SD	MSE
$e\bar{r}r$	0.0572	0.0110	0.00056
Err	0.1054	0.0051	0.00077
$CV\ (K = N)$	0.1072	0.0137	0.00103
$\hat{Err}^{(.632)}$	0.1090	0.0108	0.00107
$CV\ (K = 10)$	0.1090	0.0134	0.00113
$RCV\ (K = 10)$	0.1092	0.0126	0.00112
90/10	0.1111	0.0136	0.00127
$\hat{Err}^{(.632+)}$	0.1113	0.0109	0.00122
$RCV\ (K = 5)$	0.1139	0.0126	0.00144
$CV\ (K = 5)$	0.1144	0.0153	0.00155
80/20	0.1162	0.0130	0.00162
$CV\ (K = 3)$	0.1226	0.0149	0.0022
$\hat{Err}^{(1)}$	0.1392	0.0119	0.00387
$\hat{Err}_{boot}$	0.1395	0.0118	0.00391
$CV\ (K = 2)$	0.1460	0.0186	0.00495
50/50	0.1518	0.0114	0.00556
P(virhe)	0.0781	8.5e-5	

Taulukko 11: Lineaarisen luokittelijan virheet, 8 luokkaa 8, piirrettä, N=500.

method	$\widehat{Err}$	SD	MSE
$e\bar{r}r$	0.2413	0.0190	0.02698
$\hat{Err}^{(.632)}$	0.2640	0.0180	0.03486
$CV\ (K = 10)$	0.2641	0.0203	0.0350
$RCV\ (K = 10)$	0.2642	0.0192	0.0350
$CV\ (K = 5)$	0.2644	0.0197	0.03507
$\hat{Err}^{(.632+)}$	0.2650	0.0179	0.03523
$RCV\ (K = 5)$	0.2651	0.0189	0.03532
$CV\ (K = N)$	0.2658	0.0199	0.03563
90/10	0.2682	0.0205	0.03656
Err	0.2682	0.0059	0.03615
$CV\ (K = 3)$	0.2687	0.0215	0.03679
80/20	0.2687	0.0201	0.03671
$CV\ (K = 2)$	0.2758	0.0208	0.03952
$\hat{Err}^{(1)}$	0.2772	0.0179	0.03994
$\hat{Err}_{boot}$	0.2779	0.0178	0.04023
50/50	0.2793	0.0182	0.0408
P(virhe)	0.0781	8.5e-5	

Taulukko 12: Neuroverkkojen virheet, 8 luokkaa 8, piirrettä, N=500.

method	$\widehat{Err}$	SD	MSE
$e\bar{r}r$	0.4864	0.0695	0.1715
$CV\ (K = 10)$	0.5120	0.0312	0.18925
$CV\ (K = 3)$	0.5155	0.0404	0.19291
$RCV\ (K = 10)$	0.5156	0.0202	0.19183
$CV\ (K = 5)$	0.5186	0.0301	0.19496
$RCV\ (K = 5)$	0.5207	0.0210	0.19633
$CV\ (K = N)$	0.5215	0.0246	0.19721
90/10	0.5233	0.0237	0.19877
80/20	0.5240	0.0188	0.19916
Err	0.5255	0.0573	0.20342
$\hat{Err}^{(.632)}$	0.5266	0.0330	0.20222
$\hat{Err}^{(.632+)}$	0.5278	0.0542	0.20515
$CV\ (K = 2)$	0.5306	0.0416	0.20646
50/50	0.5385	0.0203	0.21235
$\hat{Err}^{(1)}$	0.5500	0.0646	0.22686
$\hat{Err}_{boot}$	0.5648	0.0527	0.23959
P(virhe)	0.0781	8.5e-5	

Taulukko 13: Tukivektorinkoneen virheet (sädeperusteinen ydin), 8 luokkaa 8, piirrettä, N=500.

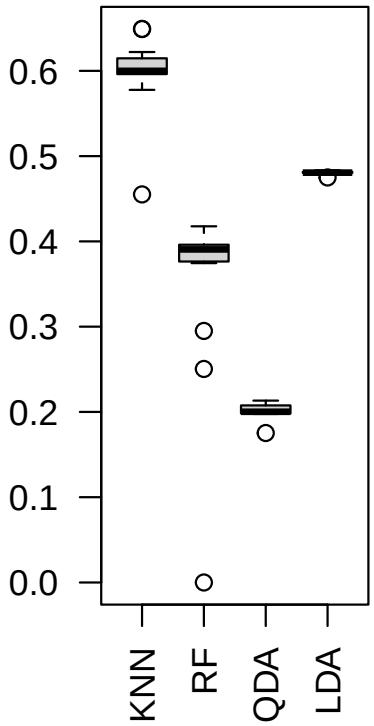
method	$\widehat{Err}$	SD	MSE
$\hat{Err}^{(.632+)}$	0.2755	0.0187	0.03931
Err	0.2843	0.0110	0.04261
90/10	0.2903	0.0213	0.04549
80/20	0.2994	0.0201	0.04935
$\hat{Err}_{boot}$	0.3021	0.0188	0.0505
$CV\ (K = N)$	0.3113	0.0203	0.0548
$RCV\ (K = 10)$	0.3135	0.0193	0.05576
$CV\ (K = 10)$	0.3138	0.0199	0.05595
$e\bar{r}r$	0.3153	0.0229	0.05676
$CV\ (K = 5)$	0.3225	0.0196	0.06008
$RCV\ (K = 5)$	0.3229	0.0189	0.06026
50/50	0.3357	0.0199	0.06672
$CV\ (K = 3)$	0.3379	0.0215	0.06793
$\hat{Err}^{(.632)}$	0.3478	0.0208	0.07318
$CV\ (K = 2)$	0.3659	0.0220	0.08328
$\hat{Err}^{(1)}$	0.3668	0.0208	0.08378
P(virhe)	0.0781	8.5e-5	

Taulukko 14: Tukivektorinkoneen virheet (lineaarinen ydin), 8 luokkaa 8, piirrettä, N=500.

method	$\widehat{Err}$	SD	MSE
$e\bar{r}r$	0.2325	0.0214	0.0243
$\hat{Err}^{(.632)}$	0.2729	0.0184	0.03826
$\hat{Err}^{(.632+)}$	0.2759	0.0174	0.03941
$CV\ (K = N)$	0.2775	0.0210	0.04021
$CV\ (K = 10)$	0.2781	0.0198	0.04038
$RCV\ (K = 10)$	0.2782	0.0192	0.04039
Err	0.2795	0.0106	0.04068
90/10	0.2825	0.0215	0.04224
$RCV\ (K = 5)$	0.2827	0.0192	0.04222
$CV\ (K = 5)$	0.2831	0.0200	0.04242
80/20	0.2894	0.0203	0.04505
$CV\ (K = 3)$	0.2910	0.0216	0.04578
$\hat{Err}^{(1)}$	0.2963	0.0181	0.04794
$\hat{Err}_{boot}$	0.2979	0.0168	0.04858
$CV\ (K = 2)$	0.3065	0.0218	0.05264
50/50	0.3136	0.0177	0.05574
P(virhe)	0.0781	8.5e-5	

5.3 50 luokkaa ja 8 piirrettä, N=10000

50 luokan tapauksessa luokkiin valittiin kaikki aineiston luokat. Satunnaismetsän ajoaika oli liitteessä olevalla koodilla noin 2 kuukautta, siinä missä kvadraattinen luokittelija ja lähinaapurinmenetelmä ja veivät muutamia päiviä. Neuroverkot ja tukivektorikoneet jäivät laskennallisen raskauden takia ajamatta. Parhaiten vertailussa suoriutui kvadraattinen luokittelija ja sen ohella toiseksi parhaiten suoriutui satunnaismetsä (Kuva 3). Lähinaapurinmenetelmä suoriutui vertailussa huonoiten.



Kuva 3: Ennustevirheet eri menetelmillä, kun 10000 aineistoa, 50 luokkaa ja 8 piirrettä.

Lähimpänä yleistämisvirhettä pääsi lähinaapurin tapauksessa $K{=}N$-ristiinvalidointi. $\hat{Err}^{(.632)}$ puolestaan aliestimoi yleistämisvirhettä. Huonoiten suoriutui $\hat{Err}^{(1)}$ (Taulukko 15).

Satunnaismetsän kohdalla sekä $\hat{Err}^{(.632)}$ ja $\hat{Err}^{(.632+)}$ aliestimoivat yleistämisvirhettä. Huonoimpaan tulokseen päästiin $K = 2$- ristiinvalidoinnilla (Taulukko 16).

Kvadraattisen luokittelijan tapauksessa parhaiten suoriutui $\widehat{Err}^{.632}$, joka oli myös lähimpänä Bayes-virhettä sekä yleistämisvirhettä (Taulukko 17). Kvadraattisen luokittelijan virheet olivat muita menetelmiä lähempänä Bayes-virhettä.

Lineaarisen luokittelijan vaihtelu oli muihin luokittelijoihin verrattuna pienintä, ja lisäksi virheiden ero yleistämisvirheestä oli pientä (Taulukko 18). Lähimmäksi yleistämisvirhettä pääsi $\hat{Err}^{(.632+)}$, sekä ristiinvalidointi arvolla $K = 5$ ja toistuva ristiinvalidointi arvolla $K = 5$. Alhaisimman virheen sai ristiinvalidointi arvolla $K = 10$, joka aliestimoi yleistämisvirhettä vaikkakin hyvin pienin määrin.

Taulukko 15: Lähinaapurin virheet, 50 luokkaa 8, piirrettä, N=10000

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.4551	0.0046	0.07258
$\hat{Err}^{(.632)}$	0.5777	0.0046	0.15367
$CV\ (K = N)$	0.5944	0.0059	0.16707
Err	0.5951	0.0051	0.16762
$CV\ (K = 10)$	0.5973	0.0061	0.16941
$RCV\ (K = 10)$	0.5973	0.0055	0.16945
$\hat{Err}^{(.632+)}$	0.5978	0.0049	0.16983
90/10	0.5980	0.0055	0.16997
$CV\ (K = 5)$	0.6020	0.0059	0.17335
$RCV\ (K = 5)$	0.6021	0.0052	0.17335
80/20	0.6029	0.0050	0.17407
$CV\ (K = 3)$	0.6089	0.0058	0.17914
$CV\ (K = 2)$	0.6206	0.0061	0.18915
50/50	0.6221	0.0050	0.19043
$\hat{Err}_{boot}$	0.6491	0.0050	0.2147
$\hat{Err}^{(1)}$	0.6491	0.0053	0.21474
P(virhe)	0.1857	0.00012	

Taulukko 16: Satunnaismetsän virheet, 50 luokkaa 8, piirrettä, N=10000

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.000	0	0.0345
$\hat{Err}^{(.632)}$	0.2504	0.0033	0.00419
$\hat{Err}^{(.632+)}$	0.2950	0.0046	0.01196
Err	0.3745	0.0059	0.03566
90/10	0.3783	0.0057	0.03713
80/20	0.3836	0.0054	0.03916
$CV\ (K = N)$	0.3871	0.0055	0.04057
$CV\ (K = 10)$	0.3906	0.0056	0.04199
$RCV\ (K = 10)$	0.3906	0.0055	0.04198
$CV\ (K = 5)$	0.3956	0.0058	0.04409
$RCV\ (K = 5)$	0.3956	0.0054	0.04407
$\hat{Err}_{boot}$	0.3961	0.0053	0.04429
$\hat{Err}^{(1)}$	0.3961	0.0053	0.04429
$CV\ (K = 3)$	0.4038	0.0055	0.04757
50/50	0.4062	0.0052	0.04865
$CV\ (K = 2)$	0.4177	0.0061	0.05385
P(virhe)	0.1857	0.00012	

Taulukko 17: Kvadraattisen luokittelijan virheet, 50 luokkaa 8, piirrettä, N=10000

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.1753	0.0042	0.00013
Err	0.1976	0.0044	0.00016
$\hat{Err}^{(.632)}$	0.1979	0.0042	0.00017
$CV\ (K = N)$	0.1982	0.0044	0.00017
$\hat{Err}^{(.632+)}$	0.1983	0.0042	0.00017
$CV\ (K = 10)$	0.1991	0.0044	2e-04
$RCV\ (K = 10)$	0.1991	0.0043	2e-04
90/10	0.1998	0.0047	0.00022
$CV\ (K = 5)$	0.2006	0.0044	0.00024
$RCV\ (K = 5)$	0.2006	0.0044	0.00024
80/20	0.2016	0.0045	0.00027
$CV\ (K = 3)$	0.2043	0.0048	0.00037
$\hat{Err}_{boot}$	0.2110	0.0042	0.00066
$\hat{Err}^{(1)}$	0.2111	0.0042	0.00066
$CV\ (K = 2)$	0.2114	0.0048	0.00068
50/50	0.2133	0.0042	0.00078
P(virhe)	0.1857	0.00012	

Taulukko 18: Lineaarisen luokittelijan virheet, 50 luokkaa 8, piirrettä, N=10000

method	$\widehat{Err}$	SD	MSE
$\bar{err}$	0.4750	0.0048	0.08372
$CV\ (K = 10)$	0.4803	0.0050	0.0868
$RCV\ (K = 10)$	0.4803	0.0048	0.08677
$\hat{Err}^{(.632)}$	0.4803	0.0048	0.08679
$\hat{Err}^{(.632+)}$	0.4803	0.0047	0.0868
Err	0.4804	0.0055	0.08686
$CV\ (K = 5)$	0.4805	0.0049	0.08693
$RCV\ (K = 5)$	0.4805	0.0049	0.08693
$CV\ (K = 3)$	0.4808	0.0051	0.08708
$CV\ (K = N)$	0.4809	0.0048	0.08716
80/20	0.4815	0.0048	0.08752
$CV\ (K = 2)$	0.4816	0.0051	0.08754
90/10	0.4817	0.0051	0.0876
$\hat{Err}_{boot}$	0.4833	0.0046	0.08858
$\hat{Err}^{(1)}$	0.4833	0.0049	0.0886
50/50	0.4834	0.0048	0.08863
P(virhe)	0.1857	0.00012	

6 Pohdintaa

Kaikissa tapauksissa kvadraattinen luokittelija osottautui optimaaliseksi luokittelijaksi. Kahden luokan ja kahden piirteen tapauksessa kvadraattisen luokittelijan ohella suoriutui lineaarinen luokittelija parhaiten. Lähinaapurinmenetelmä puolestaan sai korkeimmat virheet. On syytä kuitenkin huomioida, miten k:n valinta voi vaikuttaa tuloksiin. 8 luokan ja 8 piirteen tapauksessa antoi lineaarinen luokittelija edelleen toisiksi alhaisimmat virheet. Suurimmat virheet tulivat neuroverkoilla ja lähinaapurinmenetelmällä. 50 luokan ja 8 piirteen tapauksessa toiseksi pienimmät virheet tuotti satunnaismetsä. Lähinaapurimenetelmä tuotti tässäkin tapauksessa korkeimmat virheet.

Lähimmäksi yleistämisvirhettä päästiin toistuvalla jaolla opetus- ja testiaineistoksi jaolla 90/10, ja ristiinvalidoinnilla arvolla K=N. Lisäksi $\hat{Err}^{(.632)}$ suoriutui hyvin yleistämisvirheen estimoinnissa, vaikkakin se myös useasti aliestimoi virhettä. Toistuva jako opetus- ja testiaineistoksi 50-50, $\hat{Err}^{(1)}$, sekä $K = 2$ ristiinvalidointi puolestaan suoriutuivat huonoiten.

Laskennallisesti satunnaismetsä ja tukivektorikone sädeperusteisella ytimillä ovat olleet raskaimpia suorittaa, siinä missä lineaarinen ja kvadraattinen luokittelija, sekä lähimmän naapurin menetelmä ovat olleet nopeampia. Tukivektorikoneen hitaukseen voi liittyä sen olevan suunniteltu luokittelemaan vain kahta luokkaa, ja tässä tapauksessa $k(k-1)/2$ binäärisen luokan luokittelu on laskennallisesti raskasta. Lisäksi erityisesti $K = N$ ristiinvalidointi on hidasta, sillä malli evaluoidaan jokaiselle otokselle.

Tutkielmassa on lisäksi käytetty suurimmiksi osaksi R:n *caret*-paketin avulla luotuja funktioita (Kuhn M, 2022), vaikka vaihtoehtoisia funktioitakin voisi olla. Lähiaikoina myös yleistynyt ROC-käyrä voisi olla mielenkiinnon kohteena (Hand, 2001). Sitä yleisesti käytetään kaksiluokkaista luokittelua varten, mutta on myös olemassa laajennuksia koskemaan useampia luokkia.

On myös syytä huomioida, kuinka *klaR*-kirjaston *stepclass*-funktion valitsemat parhaat piirteet vaikuttavat tuloksiin (Weihs, 2005). Lisäksi neuroverkkojen suuria virheitä voi olla syytä tutkia. Piilokerrosten ja piiloyksiköiden määrä, sekä sakkotermin arvo vaikuttavat olennaisesti neuroverkon antamiin luokituksiin. Näiden muuttaminen voi olla tärkeässä roolissa neuroverkon antamissa tuloksiin.

Tutkielma näyttää, miten muut luokittelijat suoriutuvat normaalijakautuneisuuden ollessa voimassa optimaaliselle kvadraattiselle luokittelijalle. Myös vaihtoehtoisia luokittelijoita olisi voinut käyttää, kuten painotettua lähinaapurinmenetelmää tai esimerkiksi additiivista Bayes-regressiopuuta (Chipman, 2010).

Toisaalta myös normaalijakautuneisuuden oletusta on syytä harkita, ja kokeilun arvoista olisi nähdä, miten luokittelijat suoriutuvat keskenään, kun normaalijakautuneisuus ei pädekään. Yllättävää on myös satunnaisen metsän pienempi virhetulos verrattuna lineaariseen luokittelijaan, kun luokitellaan 50 luokkaa ja 8 piirrettä. Lineaarisen ja kvadraattisen luokittelijan virheiden erona voidaan nähdä kovarianssimatriisien ero, sillä niiden ollessa sama, palataan takaisin lineaariseen luokittelijaan.

Lähteet

Hastie, T., Tibshirani, R., & Friedman, J. (2016). The Elements of Statistical Learning, 2nd edition.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: with Applications in R. Springer.

Moisio, L. (2016). Luokittelumenetelmän evaluointimenetelmien vertailu pienten aineistojen tapauksessa simulointikokein. Jyväskylän Yliopisto.

Boateng Ernest et al. (2020). "Basic Tenets of Classification Algorithms K-Nearest-Neighbor, Support Vector Machine, Random Forest and Neural Network: A Review".Journal of Data Analysis and Information Processing, Vol.8 No.4.

Tran, Huy Quang, and Cheolkeun Ha. (2020). "High precision weighted optimum K-nearest neighbors algorithm for indoor visible light positioning applications."IEEE Access 8 (2020): 114597-114607.

Duda Richard, Hart Peter E, Stork David G. (2000). "Pattern Classification"Wiley -Interscience; 2nd edition.

Ärje, J., Kärkkäinen, S., Turpeinen, T., & Meissner, K. (2013). Breaking the curse of dimensionality in quadratic discriminant analysis models with a novel variant of a Bayes classifier enhances automated taxa identification of freshwater macroinverte- brates. Environmetrics, 24 (4): 248-259.

S. Karamizadeh, S. M. Abdullah, M. Halimi, J. Shayan and M. j. Rajabi. (2014). "Advantage and drawback of support vector machine functionality,"2014 International Conference on Computer, Communications, and Control Technology (I4CT), Langkawi, Malaysia, pp. 63-65, doi: 10.1109/I4CT.2014.6914146.

Lopez-Bernal, D.; Balderas, D.; Ponce, P.; Molina, A. (2021). Education 4.0: Teaching the Basics of KNN, LDA and Simple Perceptron Algorithms for Binary Classification Problems. Future Internet, 13, 193. https://doi.org/10.3390/fi13080193.

Weihs, C., Ligges, U., Luebke, K. and Raabe, N. (2005). klaR Analyzing German Business Cycles. Data Analysis and Decision Support, 335-343, Springer-Verlag, Berlin.

Kuhn M. (2022). _caret: Classification and Regression Training_. R package version 6.0-93, https://CRAN.R-project.org/package=caret.

Venables, W. N. & Ripley, B. D. (2002). Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F. (2022). _e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R package version 1.7-12, https://CRAN.R-project.org/package=e1071.

Fulcomer, M.C., Schönemann, P.H. & Molnar, G. (1974). Classification by linear and quadratic discriminant scores. Behavior Research Methods & Instrumentation 6, 443-445. https://doi.org/10.3758/BF03200398.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning (1st ed.). Springer. ISBN 978-0-387-31073-2.

Agostini, G., Longari, M., & Pollastri, E. (2003). Musical Instrument Timbres Classification with Spectral Features. EURASIP Journal on Advances in Signal Processing. doi:10.1155/s1110865703210118.

Breiman, L. (2001). Random Forests. Machine Learning 45, 5-32. https://doi.org/10.1023/A:1010933404324.

Joutsijoki, H. & Juhola, M. (2012). DAGSVM vs. DAGKNN: an experimental case study with benthic macroinvertebrate dataset. Machine Learning and Data Mining in Pattern Recognition, 7376: 439-453.

T. Cover and P. Hart. (1967). "Nearest neighbor pattern classification,"in IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21-27, doi: 10.1109/TIT.1967.1053964.

Breiman, L. and Spector, P. (1992). Submodel selection and evaluation in regression: the X-random case, International Statistical Review 60: 291-319.

Molinaro AM, Simon R, Pfeiffer RM. (2005). Prediction error estimation: a comparison of resampling methods. Bioinformatics. Aug 1;21(15):3301-7. doi: 10.1093/bioinformatics/bti499. Epub 2005 May 19. PMID: 15905277.

M. P. Perrone and L. N. Cooper. (1993). "When networks disagree: Ensemble methods for hybrid neural networks,"in Neural Networks for Speech and Image Processing, R. J. Mammone, Ed. London, U.K.: Chapman & Hall, pp. 126-142.

Ji-Hyun Kim. (2009), "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap". Computational Statistics & Data Analysis, Volume 53, Issue 11.

Hugh A. Chipman. Edward I. George. Robert E. McCulloch. (2010). "BART: Bayesian additive regression trees."Ann. Appl. Stat. 4 (1) 266 - 298. https://doi.org/10.1214/09-AOAS285

Hand, D.J., Till, R.J. (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. Machine Learning 45, 171-186. https://doi.org/10.1023/A:1010920819831

Liitteet

R-koodi

```
###################################
# Lähinaapuri
###################################
# 1. Aineiston muokkaaminen
# Kirjastot
library(MASS)
library(mixtools)
library(bootstrap)
library(class)
library(ipred)
library(caret)
library(bootstrap)
library(RSNNS)
library(nnet)
library(randomForest)
library(klaR)
library(e1071)

# Aineiston lukeminen
bugs <- read.table("pohjikset50.dat",
                   header=TRUE)
myvars <- names(bugs)
bugs2 <- bugs[!myvars]

# Poistetaan tarpeettomat muuttujat id (id),
# Label (kuvan numero) ja Set (kaikki samoja).
bugs2 <- bugs2[,4:68]

# Valitaan pohjaeläinaineiston 2 isointa luokkaa.
newclasses <- c("Taenneb","Isoperla")

########################################
# 8-8 tapauksessa
# clas <- levels(bugs2$Class)
# l_cl <- table(bugs2$Class)>240
# newclasses <- clas[l_cl]

# 50-8 tapauksessa
# newclasses <- levels(bugs2$Class)
########################################



# Etsitään 2 "parasta" piirrettä R-funktiolla stepclass.
```

```
# maxvar: valittavien piirteiden määrä
# direction: "forward" eli eteenpÃCin askeltava algoritmi
# criterion: correctness rate
# fold: 10-kertainen ristiinvalidointi
# method: sovitettava malli QDA
#
# library(klaR)
# stepclass(Class ~ ., data=bugs2, maxvar=2, direction="forward",
#           criterion="CR", fold=10, method="qda")

# Valitaan 2 "parasta" piirrettä
# Lasse Moisio, 2016
newvar <- c("Class","Mode.green.","YM.gray.")
newdata <- bugs2[newvar]
newdata <- subset(newdata, Class %in% newclasses)

##########################################

# 8-8 tapauksessa, 50-8 tapauksessa
#newvar <- c("Class","Minor","Mean.green.","Mean.gray.","Mean.blue.",
#            "Major","Solidity","Median.red.","Min.green.")
#newdata <- bugs2[newvar]
#newdata <- subset(newdata, Class %in% newclasses)
#newdata <- droplevels(newdata)

##############################################


# 2. Funktiot Bayes-virheen laskemiseksi

#Todennäköisyyttä arvioidaan ns. MC-estimaatilla
#Tarvittavat funktiot Bayes-virheen laskemiseksi

# Funktio estimoi aineistosta luokkien prioritodennäköisyydet,
# keskiarvovektorit sekä kovarianssimatriisit.
# Funktion parametreina ovat aineisto ja valitut luokat.
# Aineiston ensimmäisessä sarakkeessa tÃCytyy olla luokat.
# Lasse Moisio, 2016
calc_stat <- function(data, newclasses){
  new_data <- NULL
  priors <- NULL
  means <- matrix(0, length(newclasses), ncol(data)-1)
  covars <- list()
  for (i in 1:length(newclasses)){
    new_data <- subset(data, Class==newclasses[i])
    priors[i] <- nrow(new_data)/nrow(data)
    means[i,] <- apply(new_data[,2:ncol(data)], 2, mean)
    covars[[i]] <- cov(new_data[2:ncol(data)])
  }
  return(list(priors=priors, means=means, covars=covars))
```

```
}



# Luokittelupistemäärän laskemista varten lasketaan luokan k
# kovarianssimatriisin käänteismatriisi sekä determinantti.
# Funktion parametreina ovat funktion calc_stat.R antamat
# estimaatit aineiston tunnusluvuille sekä luokkien lukumäärä N_k.
# Lasse Moisio, 2016
invdet_cov <- function(data_sum, N_k){
  covars <- data_sum$covars
  invcovmat <- list()
  detcov <- NULL
  for(k in 1:N_k){
    invcovmat[[k]] <- solve(covars[[k]])
    detcov[k] <- log(det(covars[[k]]))
  }
  return(list(invcovmat=invcovmat, detcov=detcov))
}

# Arvotaan yksilölle luokka ja piirteet, ja lasketaan yksilölle
# kunkin luokan luokittelupistemäärä.
# Pienin luokittelupistemäärä on ennustettu luokka yksilölle.
# Toistetaan sama proseduuri m:lle yksilölle.
# obj: funktion invdet_cov estimaatit
# p_features: piirteiden lukumäärä
# Funktio palauttaa estimaatin Bayes-virheelle.
# Lasse Moisio, 2016
score <- function(data_sum, obj, N_k, p_features, m){
  invcovmat <- obj$invcovmat
  detcov <- obj$detcov
  means <- data_sum$means
  priors <- data_sum$priors
  score <- NULL
  wrong <- 0
  require(mixtools)
  for(i in 1:m){
    cl <-sample(c(1:N_k), size=1, replace=TRUE, prob=data_sum$priors)
    x <- rmvnorm(1, mu=data_sum$means[cl,],sigma=data_sum$covars[[cl]])
    x <- matrix(x, ncol=1, nrow=p_features)
    for(k in 1:N_k){
      score[k] <- t(x-means[k,])%*%invcovmat[[k]]%*%
        as.matrix(x-means[k,])+detcov[k]-2*log(priors[k])
    }
    pred <- which.min(score)
    if(pred!=cl){
      wrong <- wrong + 1
    }
```

```
  }
  Err <- wrong/m
  return(Err)
}

# Bayes-virheen laskeminen

m <- 10000000 # Simuloitavien yksilöiden lukumäärä
N_k <- 2 # Luokkien lukumäärä (myös 8 ja 50)
p_features <- 2 # Piirteiden lukumäärä (myös 8)

data_sum <- calc_stat(data=newdata, newclasses=newclasses)
data_sum
invde <- invdet_cov(data_sum=data_sum, N_k=N_k)
invde
#seed <- sample(.Machine$integer.max, size=1)
seed<-1572521908 # 8-8 tapauksessa: 495870689, 50-8 tapauksessa: 1879981471

set.seed(seed)

# Estimaatti Bayes-virheelle
# p_err <- score(data_sum=data_sum, obj=invde, N_k=N_k,
#               p_features=p_features, m=m)

# Laskettu aikaisemmin: Bayes-virhe 2 luokalle ja 2 piirteelle
p_err <- 0.0527183 #8-8 tapauksessa 0.0781255, 50-8 tapauksessa 0.1857347
p_err
# Estimaatti Bayes-virheen keskivirheelle
sd_p_err <- sqrt(p_err*(1-p_err)/m)
sd_p_err


# 3. Otoksen uudelleenkäyttömenetelmät



# Arvotaan kullekin yksilölle luokka käyttäen
# estimoituja prioritodennäköisyyksiä
# Funktion parametreina ovat funktion calc_stat.R
# estimaatit, luokkien lukumäärä N_k ja otoskoko N.
# Lasse Moisio, 2016
class_sample <- function(data_sum, N_k, N){
  cl <- sample(1:N_k, size=N, replace=TRUE, prob=data_sum$priors)
  return(cl)
}

# Arvotaan kullekin yksilölle piirrevektori.
# Funktion parametreina ovat luokat, funktion data_sum.R
# antamat estimaatit, luokkien lukumäärä N_k, piirteiden
# lukumäärä p_features sekä otoskoko N.
```

```
# Funktio palauttaa piirteet matriisina (Nxp-matriisi).
# Lasse Moisio, 2016
feature_sample <- function(classes, data_sum, N_k, p_features, N){
  require(mixtools)
  features <- matrix(0, nrow=N, ncol=p_features)
  for(k in 1:N_k){
    ind <- which(classes==k)
    features[ind,] <- rmvnorm(length(ind), mu=data_sum$means[k,],
                              sigma=data_sum$covars[[k]])
  }
  return(features)
}

#K-fold ristiinvalidointi
#K= fold lkm
#k = naapurien lkm
cv_fit.tr <- function(K,k, data){
  require(caret)
  classes <- data[,1]
  trControl <- trainControl(method="cv", number = K)

  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "knn",
                      tuneGrid   = expand.grid(k = k),
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data)
  err <- 1-mean(fit$results$Accuracy)
  return(err)
}


#Toistettu ristiinvalidointi
# K = fold lkm
# k = naapurien lkm
# rep = toistojen määrä
rcv_fit.tr <- function(K, k, data, rep){
  require(caret)
  classes <- data[,1]
  trControl <- trainControl(method="repeatedcv", number = K, repeats = rep)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "knn",
                      tuneGrid   = expand.grid(k = k),
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data)
  err <- 1-mean(fit$results$Accuracy)
  return(err)
}
```

```
#632-virhe
#k = naapurien lkm
#number = uudelleenotantojen iteraatio
errboot632 <- function(data, k, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "knn",
                      tuneGrid = data.frame(k=k),
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "boot632",
                                               number = number))
  err <- 1-fit$results$Accuracy
  return (err)
}

#testivirhe
#(method="none")
errboot.te <- function(data, k, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "knn",
                      tuneGrid = data.frame(k=k),
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "none",
                                               number = number))
  testPred <- predict(fit, data)
  testic<-1-(sum(testPred==classes)/length(classes))
  return(testic)}



# K=N ristiinvalidointi
# k = naapurien lukumäärä
cv_fit.loo <- function(data, k, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "knn",
                      tuneGrid = data.frame(k=k),
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "LOOCV",
                                              number = number))
  err <- 1-fit$results$Accuracy
  return (err)
```

```
}
```

```
# Toistettu opetus- ja testiainejako
# Jakosuhde k annetaan murtolukuna (esimerkiksi k=1/2).
# rep: opetus- ja testiaineistojaon toistojen lukumäärä.
# Parametrina annetaan myös data, jonka ensimmäisessö sarakkeessa
# täytyy olla luokat.
# Jako opetus- ja testiaineistoksi suoritetaan uudelleen, mikäli
# jossakin opetusaineiston luokassa on alle p+1 yksilöä.
# Lasse Moisio, 2016
rhold_out.knn <- function(data, N, k, rep){
  require(class)
  Err <- NULL
  for(i in 1:rep){
    repeat {
      train_rows <- sample(1:N,(k)*N)
      data_train <- data[train_rows,]
      data_train$classes <- as.factor(data_train$classes)
      data_test <- data[-train_rows,]
      if(all(table(data_train$classes)>ncol(data)-1))
        break
    }

    KNN.pred <- knn(data_train[,2:ncol(data)],data_test[,2:ncol(data)],
    data_train[,1], k=7)
    Err[i] <- 1-sum(KNN.pred==data_test$classes)/nrow(data_test)
  }
  error <- mean(Err)
  return(error=error)
}

# Testivirheen/yleistämisvirheen laskeminen
# Opetusaineiston koko on N_train ja
# testiaineiston koko N_test.
# Lasse Moisio, 2016
test_err_knn <- function(data_sum, N_train, N_test, N_k, p_features, k){
  err_o <- NULL
  N <- N_train + N_test
  for(i in 1:N_sim){
    set.seed(i)
    classes <- class_sample(data_sum=data_sum, N_k=N_k, N=N)
    classes <- as.factor(classes)
    features <- feature_sample(classes=classes, data_sum=data_sum,
                               N_k=N_k, p_features=p_features, N=N)
```

```
    data <- data.frame(cbind(classes,features))
    train_rows <- 1:N_train
    data_train <- data[train_rows,]
    data_test <- data[-train_rows,]

    data_train$classes <- as.factor(data_train$classes) #!!!
    data_test$classes <- as.factor(data_test$classes)

    KNN.pred <- knn(data_train[,2:ncol(data)], data_test[,2:ncol(data)],
    data_train[,1], k=k)
    err_o[i] <- 1-sum(KNN.pred==data_test$classes)/nrow(data_test)
  }
  return(err_o=err_o)
}


data_sum <- calc_stat(data=newdata, newclasses=newclasses)

N <- 50 # Simuloitavien havaintojen lkm
# 8-8 tapauksessa 100
# 50-8 tapauksessa 10000

N_sim <- 100 # Simuloitavien aineistojen lukumäärä
N_k <- 2 # Luokkien lkm (myös 8 ja 50)
p_features <- 2 # 8 muissa tapauksissa
B<-50    # Bootstrap-otosten lkm

# Alustetaan muuttujat
err_train <- NULL; err_cv2 <- NULL; err_cv3 <- NULL;
err_cv5 <- NULL; err_cv10 <- NULL; err_cvlo <- NULL;
err_rcv5 <- NULL; err_rcv10 <- NULL; err_boot <- NULL;
err_loo_boot <- NULL; err_632 <- NULL; err_632plus <- NULL;
err_ho50 <- NULL; err_ho80 <- NULL; err_ho90 <- NULL


#Boot632+ ja errboot varten wrapperi
mypredict.knn <- function(object, newdata){
  predict.ipredknn(object, newdata, type="class")}

# Datan simulointi silmukka
# Lasse Moisio, 2016
system.time(
for(i in 1:N_sim){
  cat(i, " ")
  set.seed(i)
  classes <- class_sample(data_sum=data_sum, N_k=N_k, N=N)
  as.factor(classes) ->classes
  features <- feature_sample(classes=classes,
  data_sum=data_sum, N_k=N_k, p_features=p_features, N=N)
  sim_data <- data.frame(cbind(classes,features))
```

```
  sim_data$classes <- as.factor(sim_data$classes)

  set.seed(i)
  err_train[i] <- errboot.te(sim_data, k=7, number = 50)
  set.seed(i)
  err_632[i] <- errboot632(sim_data, k=7, number=50)
  set.seed(i); err_632plus[i] <- errorest(classes ~., data=sim_data, model=ipredknn,
                                         predict=mypredict.knn,
                                         est.para = control.errorest(nboot=50),
                                         estimator="632plus", k=7)
  set.seed(i); err_boot[i] <- errorest(classes ~., data=sim_data, model=ipredknn,
                                       predict=mypredict.knn,
                                       est.para = control.errorest(nboot=50),
                                       estimator="boot", k=7)
  set.seed(i); err_cv2[i] <- cv_fit.tr(k=7, K=2, sim_data)
  set.seed(i); err_cv3[i] <- cv_fit.tr(k=7,K=3, sim_data)
  set.seed(i); err_cv5[i] <- cv_fit.tr(k=7, K=5, sim_data)
  set.seed(i); err_cv10[i] <- cv_fit.tr(k=7, K=10, sim_data)
  set.seed(i); err_cvlo[i] <- cv_fit.loo(k=7, number =N, sim_data)
  set.seed(i); err_rcv5[i] <- rcv_fit.tr (k=7, K=5, rep=5, sim_data)
  set.seed(i); err_rcv10[i] <- rcv_fit.tr(k=7, K=10, rep=5, sim_data)

  set.seed(i); err_ho50[i] <- rhold_out.knn(data=sim_data, N=N,
                                            k=1/2, rep=50)
  set.seed(i); err_ho80[i] <- rhold_out.knn(data=sim_data, N=N,
                                            k=4/5, rep=50)
  set.seed(i); err_ho90[i] <- rhold_out.knn(data=sim_data, N=N,
                                            k=9/10, rep=50)
})

#Err^(1) kaava
err_1 <- (125*err_632-46*err_train)/79

# Aiemmin laskettu Bayes-virhe 2 luokalle ja 2 piirteelle
p_err <- 0.0527183

# Estimoidaan yleistämiisvirhe 100 eri opetusaineistolle
# opetusaineiston otoskoolla 50
err_o_50 <- test_err_knn(data_sum=data_sum, N_train=50,
                         N_test=10000, N_k=2, p_features=2, k=7)


# 8-8 tapauksessa
# err_o_500 <- test_err_knn(data_sum=data_sum, N_train=500,
#                           N_test=10000, N_k=8, p_features=8, k=7)
# 50-8 tapauksessa
# err_o_10000 <- test_err_knn(data_sum=data_sum, N_train=10000,
#                             N_test=10000, N_k=50, p_features=8, k=7)

# Estimaatit luokitteluvirheille, keskihajonnoille sekä
```

```
# keskineliösummille eri otoksen uudelleenkäyttömenetelmillä
calculate_values_for_one_statistics<-function(statistics,perror)
{
  c(round(mean(statistics), 4),round(sd(statistics),4),
  round(sd(statistics)^2 +(mean(statistics) - perror)^2, 5))
}

methods<-c("err_train","err_cv2","err_cv3","err_cv5","err_cv10",
           "err_cvlo","err_rcv5","err_rcv10","err_632",
           "err_632plus","err_boot",
           "err_ho50","err_ho80","err_ho90",
           "err_o_50", "err_1")

tulosmat_knn<-data.frame(rbind(calculate_values_for_one_statistics(err_train,p_err),
                               calculate_values_for_one_statistics(err_cv2,p_err),
                               calculate_values_for_one_statistics(err_cv3,p_err),
                               calculate_values_for_one_statistics(err_cv5,p_err),
                               calculate_values_for_one_statistics(err_cv10,p_err),
                               calculate_values_for_one_statistics(err_cvlo,p_err),
                               calculate_values_for_one_statistics(err_rcv5,p_err),
                               calculate_values_for_one_statistics(err_rcv10,p_err),
                               calculate_values_for_one_statistics(err_632,p_err),
                               calculate_values_for_one_statistics(sapply(err_632plus,
                               mean), p_err),
                               calculate_values_for_one_statistics(sapply(err_boot,
                               mean), p_err),
                               calculate_values_for_one_statistics(err_ho50,p_err),
                               calculate_values_for_one_statistics(err_ho80,p_err),
                               calculate_values_for_one_statistics(err_ho90,p_err),
                               calculate_values_for_one_statistics(err_o_50,p_err),
                               calculate_values_for_one_statistics(err_1,p_err)))
tulosmat_knn<-cbind(methods,tulosmat_knn)
names(tulosmat_knn)<-c("method","hatErr","SD","MSE")
tulosmat_knn

###################################
# Satunnaismetsä
###################################
# Koodi muuten sama kuin edellä, mutta metodi muuttuu funktioissa,
# ja predict funktio palauttaa suoraan luokat.

#ristiinvalidointi satunnaismetsälle
cv_fit_rf<- function(K, data, N){
  classes <- data[,1]
  trControl <- trainControl(method="cv", number = K)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "rf",
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data)
```

```
  err <- 1-mean(fit$results$Accuracy)
  return(err)
}
#Toistettu ristiinvalidointi satunnaismetsälle
rcv_fit_rf <- function(K, data, rep){
  require(caret)
  classes <- data[,1]
  trControl <- trainControl(method="repeatedcv", number = K, repeats = rep)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "rf",
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data)
  err <- 1-mean(fit$results$Accuracy)
  return(err)
}

#Toistettu opetus/testiaine jako
#Lasse Moisio, 2016
rhold_out_rf <- function(data, N, k, rep){
  require(MASS)
  require(randomForest)
  data[,1] <- as.factor(data[,1]) #!
  Err <- NULL
  for(i in 1:rep){
    repeat {
      train_rows <- sample(1:N,(k)*N)
      data_train <- data[train_rows,]
      data_test <- data[-train_rows,]
      if(all(table(data_train$classes)>ncol(data)-1))
        break
    }
    RF <- randomForest(data_train[,2:ncol(data)], data_train$classes)
    RF.pred <- predict(RF, data_test[,2:ncol(data)])
    Err[i] <- 1-sum(RF.pred==data_test$classes)/nrow(data_test)
  }
  error <- mean(Err)
  return(error=error)
}

#Testi/yleistämisvirhe
#Lasse Moisio, 2016
test_err <- function(data_sum, N_train, N_test, N_k, p_features){
  err_o <- NULL
  N <- N_train + N_test
  for(i in 1:N_sim){
    set.seed(i)
    classes <- class_sample(data_sum=data_sum, N_k=N_k, N=N)
    features <- feature_sample(classes=classes, data_sum=data_sum,
```

```
                              N_k=N_k, p_features=p_features, N=N)
    data <- data.frame(cbind(classes,features))
    train_rows <- 1:N_train
    data_train <- data[train_rows,]
    data_test <- data[-train_rows,]

    data_train$classes <- as.factor(data_train$classes) #!!!
    data_test$classes <- as.factor(data_test$classes)

    RF <- randomForest(data_train[,2:ncol(data)], data_train$classes)
    RF.pred <- predict(RF, data_test[,2:ncol(data)])
    err_o[i] <- 1-sum(RF.pred==data_test$classes)/nrow(data_test)
  }
  return(err_o=err_o)
}
#Bootstrap632 virhe
#number = uudelleenotantojen iteraatio
errboot632 <- function(data, number){
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "rf",
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "boot632",
                                               number = number))
  err <- 1-fit$results$Accuracy
  return (err)
  }

#Opetusvirhe
errboot.te <- function(data, number){
  classes <- data[,1]
  classes <- as.factor(classes)
  fit <- caret::train(as.factor(classes)~.,
                      method = "rf",
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "none",
                                               number = number))
  testPred <- predict(fit, data)
  testic<-1-(sum(testPred==classes)/length(classes))
  return(testic)
  }

#K=N ristiinvalidointi
cv_fit.loo <- function(data, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "rf",
```

```
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "LOOCV",
                                               number = number))
  err <- 1-fit$results$Accuracy
  return (err)

}

#Errorestiä varten
#(ei $class randomForestiin, sillä se palauttaa suoraan luokat!)
mypredict.rf <- function(object, newdata)
{predict(object, newdata = newdata)}


###################################
# Kvadraattinen luokittelija
###################################

# Koodi muuten sama kuin lähinaapurin, mutta method="qda" monissa funktioissa.

# ristiinvalidointi
cv_fit_qda<- function(K, data, N){

  classes <- data[,1]
  trControl <- trainControl(method="cv", number = K)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "qda",
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data)

  err <- 1-mean(fit$results$Accuracy)
  return(err)
}

#Toistettu ristiinvalidointi QDA:lle
# K = fold lkm
# rep = toistojen määrä
rcv_fit_qda <- function(K, data, rep){
  require(caret)
  classes <- data[,1]
  trControl <- trainControl(method="repeatedcv", number = K, repeats = rep)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "qda",
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data)
  err <- 1-mean(fit$results$Accuracy)
  return(err)
```

```
}


#Toistettu opetus- ja testiainejako
#Jakosuhde k annetaan murtolukuna (esimerkiksi k=1/2)
#Lasse Moisio, 2016
rhold_out <- function(data, N, k, rep){
  require(MASS)
  Err <- NULL
  for(i in 1:rep){
    repeat {
      train_rows <- sample(1:N,(k)*N)
      data_train <- data[train_rows,]
      data_test <- data[-train_rows,]
      if(all(table(data_train$classes)>ncol(data)-1))
        break
    }
    QDA <- qda(data_train[,2:ncol(data)], data_train$classes)
    QDA.pred <- predict(QDA, data_test[,2:ncol(data)])$class
    Err[i] <- 1-sum(QDA.pred==data_test$classes)/nrow(data_test)
    #AUC <- roc(response=data_test$classes, predictor = as.numeric(QDA.pred))
    #AUCd[i] <- AUC$auc[1]
  }
  error <- mean(Err)
  return(error=error)
}

# Testivirheen/yleistÃCmisvirheen laskeminen
# Opetusaineiston koko on N_train ja
# testiaineiston koko N_test.
# Lasse Moisio, 2016
test_err_qda <- function(data_sum, N_train, N_test, N_k, p_features){
  err_o <- NULL
  N <- N_train + N_test
  for(i in 1:N_sim){
    set.seed(i)
    classes <- class_sample(data_sum=data_sum, N_k=N_k, N=N)
    features <- feature_sample(classes=classes, data_sum=data_sum,
                               N_k=N_k, p_features=p_features, N=N)
    data <- data.frame(cbind(classes,features))
    train_rows <- 1:N_train
    data_train <- data[train_rows,]
    data_test <- data[-train_rows,]
    QDA <- qda(data_train[,2:ncol(data)], data_train$classes)
    QDA.pred <- predict(QDA, data_test[,2:ncol(data)])$class
    err_o[i] <- 1-sum(QDA.pred==data_test$classes)/nrow(data_test)
  }
  return(err_o=err_o)
}
```

```
#Bootstrap632 virhe
#number = uudelleenotantojen iteraatio
errboot632 <- function(data, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "qda",
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "boot632",
                                               number = number))
  err <- 1-fit$results$Accuracy
  return (err)
}

#Opetusvirhe
errboot.te <- function(data, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "qda",
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "none",
                                               number = number))
  testPred <- predict(fit, data)
  testic<-1-(sum(testPred==classes)/length(classes))
  return(testic)}



#Leave one out CV
cv_fit.loo <- function(data, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "qda",
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "LOOCV",
                                              number = number))
  err <- 1-fit$results$Accuracy
  return (err)

}

mypredict.qda <- function(object, newdata)
{
  predict(object, newdata = newdata)$class
}
```

```
###################################
# Lineaarinen luokittelija
###################################

# Sama kuin edellä, mutta method="lda" ja qda() funktion sijaan funktio lda().

###################################
# Neuroverkot
###################################

# Sama kuin edellä, mutta funktioissa metodit vaihtuvat

# Ristiinvalidointi K taitoksella
# valitsee automaattisesti optimi piiloyksiköiden määrän
# sekä penalisointitermin arvon
cv_fit_nn<- function(K, data, N){
  require(caret)
  classes <- data[,1]
  trControl <- trainControl(method="cv", number = K)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "nnet",
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data,
                     tuneGrid=expand.grid(size=5, decay = 0.1))


  #parasMalli <<- fit$results[which.max(fit$results$Accuracy),]
  # size = piiloyksiköiden määrä = 5
  # decay= penalisointitermi =  0.1
  err <- 1- fit$results$Accuracy[which.max(fit$results$Accuracy)]

  return(err)

}

#Toistettu ristiinvalidointi
# K = fold lkm
# rep = toistojen määrä
rcv_fit_nn <- function(K, data, rep){
  require(caret)
  classes <- data[,1]
  trControl <- trainControl(method="repeatedcv", number = K, repeats = rep)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "nnet",
                      trControl  = trControl,
                      metric     = "Accuracy",
                      data       = data,
```

```
                      tuneGrid=expand.grid(size=5,
                                           decay = 0.1))
  err <- 1-mean(fit$results$Accuracy)
  return(err)
}

#Toistuva jako testi- ja opetusaineistoksi jakosuhteella k
#Lasse Moisio, 2016
rhold_out_nn <- function(data, N, k, rep){
  require(MASS)
  require(nnet)
  virhe <- NULL
  data[,1] <- as.factor(data[,1]) #!
  Err <- NULL
  for(i in 1:rep){
    repeat {
      train_rows <- sample(1:N,(k)*N)
      data_train <- data[train_rows,]
      data_test <- data[-train_rows,]
      if(all(table(data_train$classes)>ncol(data)-1))
        break
    }
    NN <- nnet((classes)~.,
               size = 5, decay = 0.1, data=data_train)
    NN.pred <- predict(NN, newdata = data_test[,2:ncol(data_test)],
                       type="class") #luokat luokkatn:ksien sijaan
    Err[i] <- 1-sum(NN.pred==data_test$classes)/nrow(data_test)
  }
  return(mean(Err))
}

#Yleistämisvirhe
#Lasse Moisio, 2016
test_err_nn <- function(data_sum, N_train, N_test, N_k, p_features){
  err_o <- NULL
  N <- N_train + N_test
  for(i in 1:N_sim){
    set.seed(i)
    classes <- class_sample(data_sum=data_sum, N_k=N_k, N=N)
    features <- feature_sample(classes=classes, data_sum=data_sum,
                               N_k=N_k, p_features=p_features, N=N)
    data <- data.frame(cbind(classes,features))
    train_rows <- 1:N_train
    data_train <- data[train_rows,]
    data_test <- data[-train_rows,]

    data_train$classes <- as.factor(data_train$classes) #!!!
    data_test$classes <- as.factor(data_test$classes)

    NN <- nnet((classes)~.,
```

```
              size = 5, decay = 0.1, data=data_train)
    NN.pred <- predict(NN, newdata = data_test[,2:ncol(data)],type="class")
    err_o[i] <- 1-sum(NN.pred==data_test$classes)/nrow(data_test)
  }
  return(err_o=err_o)
}

#Bootstrap632 virhe
#number = uudelleenotantojen iteraatio
#K = rep
errboot632_nn <- function(data, number){
  classes <- data[,1]
  trControl <- trainControl(method="boot632", number = number)
  fit <- caret::train(as.factor(classes) ~ .,
                      method     = "nnet",
                      trControl  = trControl,
                      metric     = "Accuracy",
                      tuneGrid=expand.grid(size=5,
                                           decay = 0.1),
                      data       = data)
  err <- 1-fit$results$Accuracy
  return (err)
}

#Opetusvirhe eli errbar
#number = uudelleenotantojen lkm
errboot.te_nn <- function(data, number){
  require(caret)
  classes <- data[,1]
  classes <- as.factor(classes)
  fit <- caret::train(as.factor(classes)~.,
                      method = "nnet",
                      metric = "Accuracy",
                      data = data,

                      trControl = trainControl(method = "none",
                                               number = number),
                      tuneGrid=expand.grid(size=5,
                                           decay = 0.1))
  testPred <- predict(fit, data)
  testic<-1-(sum(testPred==classes)/length(classes))
  return(testic)}



#number = uudelleenotantojen lkm
cv_fit.loo_nn <- function(data, number){
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "nnet",
```

```
                        metric = "Accuracy",
                        data = data,
                        trControl = trainControl(method = "LOOCV",
                                                 number = number),
                        tuneGrid=expand.grid(size=5,
                                             decay = 0.1))
  err <- 1-fit$results$Accuracy
  return (err)

}

#type = "class" antaa chr, as.factor muuttaa vaadituksi factoriksi
#vertailua varten
mypredict.nn <- function(object, newdata){
  pred <- as.factor(predict(object, newdata = newdata, type="class"))
  levels(pred) <- c(1:8)
  return (pred)}


####################################
# Tukivektorikone (lineaarinen ydin)
####################################

#Ristiinvalidointi tukivektorikoneille
cv_fit_svm<- function(K, data, N){
  require(caret)
  require(LiblineaR)
  classes <- data[,1]
  trControl <- trainControl(method="cv", number = K)
  fit <- caret::train(as.factor(classes) ~ .,
                        method     = "svmLinear",
                        trControl  = trControl,
                        metric     = "Accuracy",
                        data       = data)

  err <- 1-mean(fit$results$Accuracy)
  return(err) #C=1 default


#Toistettu ristiinvalidointi
# K = fold lkm
# rep = toistojen määrä
rcv_fit_svm <- function(K, data, rep){
  require(caret)
  classes <- data[,1]
  trControl <- trainControl(method="repeatedcv", number = K, repeats = rep)
  fit <- caret::train(as.factor(classes) ~ .,
                        method     = "svmLinear",
                        trControl  = trControl,
                        metric     = "Accuracy",
```

```
                        data       = data)
  err <- 1-mean(fit$results$Accuracy)
  return(err)
}

#Toistettu opetus- ja testijako
#Lasse Moisio, 2016
rhold_out_svm <- function(data, N, k, rep){
  require(MASS)
  require(e1071)
  data[,1] <- as.factor(data[,1]) #!
  Err <- NULL
  for(i in 1:rep){
    repeat {
      train_rows <- sample(1:N,(k)*N)
      data_train <- data[train_rows,]
      data_test <- data[-train_rows,]
      if(all(table(data_train$classes)>ncol(data)-1))
        break
    }
    SVM <- svm(data_train[,2:ncol(data)], data_train$classes,
               type="C-classification",
               kernel="linear")
    SVM.pred <- predict(SVM, data_test[,2:ncol(data)])
    Err[i] <- 1-sum(SVM.pred==data_test$classes)/nrow(data_test)
  }
  error <- mean(Err)
  return(error=error)
}

# Testivirheen/yleistämisvirheen laskeminen
# Opetusaineiston koko on N_train ja
# testiaineiston koko N_test.
# Lasse Moisio, 2016
test_err_svm <- function(data_sum, N_train, N_test, N_k, p_features){
  err_o <- NULL
  N <- N_train + N_test
  for(i in 1:N_sim){
    set.seed(i)
    classes <- class_sample(data_sum=data_sum, N_k=N_k, N=N)
    features <- feature_sample(classes=classes, data_sum=data_sum,
                               N_k=N_k, p_features=p_features, N=N)
    data <- data.frame(cbind(classes,features))
    train_rows <- 1:N_train
    data_train <- data[train_rows,]
    data_test <- data[-train_rows,]

    data_train$classes <- as.factor(data_train$classes) #!!!
    data_test$classes <- as.factor(data_test$classes)
```

```
    SVM <- svm(data_train[,2:ncol(data)], data_train$classes,
               type="C-classification",
               kernel="linear")
    SVM.pred <- predict(SVM, data_test[,2:ncol(data)])
    err_o[i] <- 1-sum(SVM.pred==data_test$classes)/nrow(data_test)
  }
  return(err_o=err_o)
}

#Bootstrap632 virhe
#number = uudelleenotantojen iteraatio
errboot632 <- function(data, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "svmLinear",
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "boot632",
                                               number = number))
  err <- 1-fit$results$Accuracy
  return (err)
}

#Opetusvirhe
#number = uudelleenotantojen lkm
errboot.te <- function(data, number){
  require(caret)
  classes <- data[,1]
  classes <- as.factor(classes)
  fit <- caret::train(as.factor(classes)~.,
                      method = "svmLinear",
                      metric = "Accuracy",
                      data = data,
                      trControl = trainControl(method = "none",
                                               number = number))
  testPred <- predict(fit, data)
  testic<-1-(sum(testPred==classes)/length(classes))
  return(testic)}



#number = uudelleenotantojen lkm
cv_fit.loo <- function(data, number){
  require(caret)
  classes <- data[,1]
  fit <- caret::train(as.factor(classes)~.,
                      method = "svmLinear",
                      metric = "Accuracy",
                      data = data,
```

```
                      trControl = trainControl(method = "LOOCV",
                                               number = number))
  err <- 1-fit$results$Accuracy
  return(err)

}
mypredict.svm <- function(object, newdata){
  predict(object, newdata = newdata)} #ei $class svm tapauksessa



###################################
# Tukivektorikone (sädeperusteinen ydin)
###################################

# Sama kuin edellä, mutta "svmLinear" korvataan "svmRadial"
# ja kernel="linear" korvataan kernel="radial".
```