Päätöspuu-mallit binääriluokitteluongelmassa

Kandi
Turun yliopisto
Fysiikka
2025
Aleksi Elomäki
Tarkastaja:
dos. Johannes Niskanen

TURUN YLIOPISTO
Fysiikan laitos

Elomäki, Aleksi Päätöspuu-mallit binääriluokitteluongelmassa

Kandi, 19 s., 3 liites.
Fysiikka
Huhtikuu 2025

Koneoppimista voidaan käyttää muun muassa regressio- ja luokittelumallien luomiseen. Tällöin voidaan kouluttaa malli ennustamaan haluttu vastemuuttuja selitettävillä muuttujilla. Päätöspuupohjaiset menetelmät sopivat hyvin luokittelutehtäviin. Yleisesti päätöspuun toimintaperiaate on se, että ne jakavat havaintoja selittävien tekijöiden perusteella. Jakoa jatketaan niin kauan, että päädytään niin kutsuttuun luokittelulehteen, jossa varsinainen luokitus tapahtuu. Selittäviksi muuttujiksi sopivat regressio- ja kategoriset muuttujat. Näitä luokittelulehtiä on oltava vähintään yhtä monta kuin kategorioita, jotta järkevä ennustaminen on mahdollista.

Työssä tarkastellaan yksittäistä päätöspuuta, satunnaismetsää (Random forest) ja extreme gradient boosting (XGBoost) -menetelmää matemaattisesti sekä empiirisesti käyttäen Pythonia. Teoreettisessa osiossa käydään läpi rajoittamattoman päätöspuun matemaattinen perusta käyttäen gini-indeksiä. Yksittäiselle päätöspuulle etsitään sopiva hyperparametrin arvo, jonka tarkoitus on tehostaa suorituskykyä ja yksinkertaistaa puuta. Lisäksi tarkastellaan XGBoostin toimintaperiaatetta ja millaisia hyperparametreja XGBoost -menetelmään voidaan asettaa. Teoreettisen osan lopussa määritetään eri metriikoita, joilla arvioidaan mallien suorituskykyjä.

Empiirisessä osiossa data valittiin binääriluokitteluun sopivaksi, jossa riisinjyvän geometrian perusteella pyrittiin ennustamaan, että kuuluuko se Osmancik vai Cammeo -lajikkeeseen. Yksittäinen päätöspuumalli luotiin ilman rajoittavaa parametria, ja tämä malli tunnisti testidatasta 87,8 % oikein (Accuracy). Päätöspuumalli luotiin myös käyttäen *ccp_alpha* -parametria, jolloin malli luokitteli oikein 93 % testidatasta. Satunnaismetsä-mallissa koulutettiin sata rajoittamatonta päätöspuuta, jolloin tarkkuudeksi saatiin 92,5 %. XGBoostissa suoritettiin yksinkertainen parametrien optimointi käyttäen sklearn -kirjaston GridSearchCV -funktiota, jolloin mallin tarkkuudeksi saatiin 93 %. Kullekin mallille luotiin ROC-käyrä sekä laskettiin sekaannusmatriisien avulla myös muita luokitteluongelmissa yleisesti käytettyjä metriikoita. Tässä työssä parhaat mallit muodostivat XGBoost ja rajoitettu päätöspuu, jotka muodostivat identtiset sekaannusmatriisit.

Asiasanat: Päätöspuu, XGboost, Random forest

Sisällys

Johdanto

Tallennetun datan lisääntymisen johdosta myös koneoppimisen menetelmien kehittyminen ja käyttö on yleistynyt monella eri alalla. Koneoppimista voidaan käyttää mm. kategoristen ja regressiotyyppisten ongelmien ratkaisemiseen. Tällöin pyritään selittämään tietty vastemuuttuja selittävien muuttujien avulla. Luokittelutehtävään hyvin sopiva menetelmä on päätöspuumalli, jossa selittävät tekijät voivat olla kategorisia tai kvantatiivisia. Teoreettisessa osiossa perehdytään päätöspuun, satunnaismetsän (Random Forest) ja extreme gradient boosting (XGBoost) -menetelmien matemaattiseen perustaan luokittelussa. Lisäksi määritetään empiirisessä osassa käytettävät mallien hyvyyttä mittaavat metriikat sekä hyperparametrit.

Työn empiirisessä osassa tarkastellaan päätöspuun muodostamista hyödyntäen Pythonin scikit-learn sekä XGBoost -kirjastoja binääriluokitteluongelman mallintamisessa. Päätöspuu muodostetaan ilman puun monimutkaisuutta rajoittavaa parametria sekä rajoittavan parametrin kanssa. Yksittäisen päätöspuun rajoittavana tekijänä käytetään cost-complexity pruning -parametria. Lisäksi tarkastellaan, miten satunnaismetsä ja XGBoost suoriutuvat binääriluokitteluongelmasta. Satunnaismetsä luodaan käyttäen sataa rajoittamatonta päätöspuuta. XGBoost -menetelmässä parametrien optimointiin käytetään suoraviivaista ruudukkohaku (Grid search) -menetelmää sekä muodostetaan sata erilaista päätöspuuta. Tähän empiiriseen kokeiluun valittiin data, joka pyrkii tunnistamaan riisilajikkeet Cammeo ja Osmancik käyttämällä seitsemää erilaista riisinjyvästä havaittua ominaisuutta. Työssä arvioidaan eri mallien hyvyyttä käyttäen luokitustehtävään sopivia mittareita (Accuracy, recall, F1 score, jne.), jotta voidaan tehdä johtopäätöksiä mallien eroavaisuuksista. Lopputuloksena extreme gradient boosting -menetelmä ja rajoitettu päätöspuu suoriutuivat luokittelusta yhtä hyvin. Rajoittamaton päätöspuu suoriutui tehtävästä selvästi huonoiten. Työssä ei ole käytetty tekoälyä.

1 Tausta

Koneoppimisessa yleisesti pyritään selvittämään havaittujen arvojen, eli selittävien muuttujien $\mathbf{x}_i$ avulla selitettävä muuttuja y_i. Koneoppiminen voidaan jakaa ohjattuun ja ohjaamattomaan oppimiseen, joissa molemmissa vastemuuttujat voivat olla kvantitatiivisia tai kategorisia. Ohjatussa oppimisessa vastemuuttujat y_i :t ovat hyvin määriteltyjä. Tällöin malli koulutetaan käyttäen joukkoa selittäviä ja vastaavia selitettäviä muuttujia, jota kutsutaan koulutusdataksi. Mallin suorituskykyä pystytään testaamaan mallin sellaisilla selitettävillä muuttujilla, joita ei ole käytetty mallin koulutuksessa. Ohjaamattomassa oppimisessa malli luodaan pohjautuen vain selitettäviin muuttujiin, joten ongelman luonne on yleensä kuvailla, miten havainnot ovat ryhmittyneet. Luokittelu ja regressio ovat kaksi hyvin yleistä ongelmatyyppiä koneoppimisessa. Luokitteluongelmassa vastemuuttuja on kategorinen ja regressiossa vastemuuttuja on jatkuva-arvoinen. Tyypillisesti luokittelutehtävässä ollaan kiinnostuneita väärien luokitusten suhteesta kaikkiin luokituksiin, eli virhesuhteesta.[1] Tämä työ keskittyy binääriluokitteluongelmaan, jossa vastemuuttuja koostuu kahdesta eri kategoriasta. Työssä käytetään päätöspuupohjaisia menetelmiä, jotka kuuluvat ohjattuun oppimiseen.

1.1 Päätöspuut

Koneoppimismalli, joka jakaa selittävien muuttujien perusteella selitettävät muuttujat, kutsutaan päätöspuumalliksi. Rakenteeltaan päätöspuu on sellainen, että ylimpänä on kantasolmu (Root node), joka jakaantuu kahteen eri suuntaan. Solmuksi (Split node) kutsutaan sellaista puun osaa, jossa puu jakaantuu kahteen eri suuntaan, mutta ei ole kantasolmu. Puun haarautuminen loppuu luokittelulehteen (Terminal node), jossa varsinainen luokitus tapahtuu. Kuvassa 1 on esitetty päätöspuun periaatteellinen rakenne, jossa selitettäviä muuttujia on kolme: "Lämmitä lisää", "Heitä löylyä"ja "Nauti". Tässä ensimmäinen jako tehdään lämpötilan mukaan

(Kantasolmu). Mikäli lämpötila on alle 80 °C, ajaudutaan suoraan luokkaan "Lämmitä lisää". Jos lämpötila on suurempi tai yhtäsuuri kuin 80 °C, sitten käytetään toista selittävää muuttujaa, eli suhteellinen ilmankosteus. Tämä ohjaa päätöksen kahteen eri luokittelulehteen sen mukaan, että onko suhteellinen ilmankosteus yli vai alle 50 %. Havaintosarjat ikään kuin valutetaan päätöspuun läpi, jolloin päädytään johonkin luokittelulehteen.

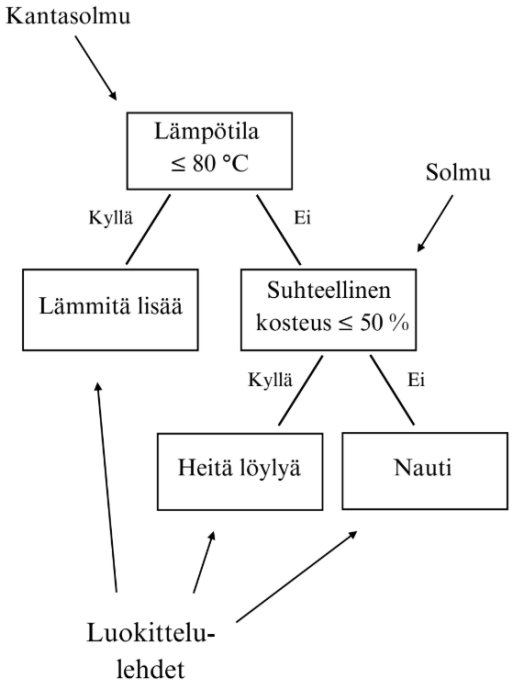
Luokitteluongelmissa päätöspuiden solmukohtien näytteiden jakoa voidaan arvioida erilaisilla mittareilla (mm. Gini, entropia). Tässä työssä käytettiin gini-indeksiä (Gini index), jonka ensisijainen tarkoitus on kuvata yksittäisen solmun tai lehden epäpuhtautta. Kunkin puun osan gini-arvo pystytään määrittämään seuraavalla kaavalla:

$$G_i = 1 - \sum_{k=1}^{K} \hat{p}_{ik}^2 = \sum_{k=1}^{K} \hat{p}_{ik}(1 - \hat{p}_{ik}), \tag{1}$$

jossa $\hat{p}_{ik}$ on havaintojen valossa laskettu todennäköisyys sille, että i:nnessä lehdessä saadaan tietty luokitus, eli kyseiseen luokkaan k kuuluvien ja kaikkien havaintojen suhdeluku. Kaavan (1) jälkimmäisestä yhtäsuuruudesta nähdään, että pienimmät gini-indeksit saavutetaan, kun $\hat{p}_{ik}$ lähestyy ykköstä, tai nollaa. Gini-indeksi saa arvoja väliltä [0,1]. Arvon ollessa nolla, tällöin kaikki lehden näytteet on luokiteltu samaan luokkaan. Gini-indeksi on suurempi kuin nolla, jos näytteet ovat sekoittuneita eri luokkiin. Päätöspuun solmukohtien järjestys perustuu vasemman ja oikean lehden gini-indeksien painotettuun keskiarvoon, eli kokonaisgini-indeksiin:

$$J(k,t_k) = \frac{m_{left}}{m_{total}} G_{left} + \frac{m_{right}}{m_{total}} G_{right}, \tag{2}$$

jossa $G_{left/right}$ on oikean/vasemman haaran gini-indeksi, $m_{left/right}$ on vasemman/oikean havaintojen lukumäärä ja m_{total} on kaikkien havaintojen lukumäärä. Kokonaisgini-



Kuva 1. Päätöspuun rakenne.

indeksin tarkoitus on asettaa päätöspuun solmujen järjestys (ylhäältä alas) käyttäen selittäviä muuttujia. Solmukohdat pyritään asettamaan siten, että ylimpänä on solmu, joka tuottaa pienimmän kokonaisgini-indeksin. Tällöin päätöspuu yrittää kokoajan löytää sellaisen jakopisteen, joka selittää vastemuuttujan käyttäen mahdollisimman vähän selittäviä muuttujia, ja siksi tätä kutsutaan "ahneeksi algoritmiksi". Käytännössä on harvinaista saavuttaa päätöspuu, jossa luokittelehdet ovat täysin puhtaita ($G_i = 0$). Kun ensimmäinen jako on tehty, etsitään vastaavalla tavalla seuraava solmukohta. Mikäli rajoittavia parametreja, kuten maksimisyvyyttä, ei ole asetettu, jakamista jatketaan, kunnes ei löydy jakopistettä, joka pienentäisi kokonaisgini-indeksiä. Toisin sanoen päätöspuu muodostetaan minimoimalla kokonaisgini-indeksi. Selittävän tekijän k ollessa jatkuva-arvoinen, etsitään pienimmän kokonaisgini-indeksin omaava raja-arvo t_k, jonka avulla muodostetaan päätöspuun solmukohta (Esim. $k \leq t_k$).[2]

1.1.1 Rajoitettu päätöspuu

Päätöspuun rakentaminen ilman rajoitteita voi muodostaa erittäin monimutkaisen puun, ja lopulta johtaa mallin ylisovittamiseen. Rajoitusmenetelmiä on useita, mutta tässä työssä on käytetty kustannus-monimutkaisuus karsinta -menetelmää (Eng. cost-complexity pruning). Karsimisella tarkoitetaan puun osaa, joka jätetään pois päätöspuusta. Ideana on, että jokaisesta lehdestä muodostuu mallin virheeseen erillinen sakko. Rajoitus voidaan kuvata kaavalla:

$$\epsilon_\alpha(T) = \epsilon(T) + \alpha\ |\tilde{T}|, \tag{3}$$

jossa $\epsilon_\alpha(T)$ edustaa koko päätöspuun kustannus-monimutkaisuus -suuretta, $\epsilon(T)$ puun tekemää virhettä (Esim. kokonaisgini-indeksi), α on hyperparametri sekä $|\tilde{T}|$ edustaa päätöspuun luokituslehtien lukumäärää. Kaavasta (3) voidaan päätellä, jos α on nolla, $\epsilon_\alpha(T)$ on yhtä suuri kuin päätöspuun virhe. Tällöin rajoittavaa tekijää päätöspuun luomiselle ei ole, vaan malliin otetaan mukaan myös ne päätöspuun solmut, joiden merkitys on mallinnuksen kannalta vähäinen tai olematon. Vastaavasti, kun hyperparametrin arvoa kasvatetaan, kustannus-monimutkaisuus -suure kasvaa, mikä on epäedullista monimutkaiselle mallille. Yksittäisen solmun t kustannus-monimutkaisuus voidaan kuvata ($|\tilde{T}|= 1$):

$$\epsilon_\alpha(t) = \epsilon(t) + \alpha\ . \tag{4}$$

Yleisesti voidaan todeta, että $\epsilon_\alpha(t) > \epsilon_\alpha(T_t)$, jossa T_t on alipuu ja t edustaa alipuun kantasolmua. Tämä on intuitiivista ajatella, että yksinkertaisempi puu tuottaa suuremman virheen, kun $\alpha = 0$. Kun α kasvatetaan, tietyssä pisteessä pätee yhtäsuuruus $\epsilon_\alpha(t) = \epsilon_\alpha(T_t)$. Kun sijoitetaan kaavat (3) ja (4) epäyhtälöön, saadaan ehto:

$$\alpha(1-|\tilde{T}_t|) > \epsilon(T_t) - \epsilon(t). \tag{5}$$

Tästä voidaan määrittää optimaalisen α:n yläraja:

$$\alpha < \frac{\epsilon(t) - \epsilon(T_t)}{(|\tilde{T}_t|-1)} = \alpha_{eff}. \tag{6}$$

Solmu, jonka α_{eff} on pienin, saavuttaa α:n kasvaessa ensimmäisenä yhtäsuuruuden $\epsilon_\alpha(t) = \epsilon_\alpha(T_t)$, jolloin se karsitaan. Tällöin saadaan uusi päätöspuu T', josta jälleen etsitään pienin α_{eff} uuden karsinnan tekemiseksi. Toisin sanoen tehtävänä on etsiä rajoitettu päätöspuu T', joka minimoi virhesuureen $\epsilon_\alpha(T')$. [3]

1.2 Joukkomallit

Yksittäisen puun sijasta voidaan käyttää useampaa mallia, joiden lopputulokset sulautetaan yhdeksi ratkaisuksi. Näitä menetelmiä kutsutaan yleisesti joukkomalleiksi (Eng. Ensemble method), joiden päällimmäinen tarkoitus on lisätä saavutettua tarkkuutta. Joukkomallit voivat käyttää samaa tai eri selittävien tekijöiden avaruutta. Tilanne, jossa voisi käyttää eri selittävien tekijöiden avaruutta, voisi olla esimerkiksi henkilön tunnistaminen. Erilliset datajoukot saataisiin usean eri sensorin mittauksista, jotka voisivat olla peräisin vaikkapa silmien verkkokalvojen ja kasvojen piirteiden skannauksista. Tällöin koulutetaan mallit, jotka tekevät omat luokittelunsa pohjautuen kunkin sensorin mittauksiin. Käytettäessä samaa selittävien muuttujien avaruutta, yhden mallin data voi sisältää vain osan S' kaikista selittävistä muuttujista S. Tällöin kullekin osajoukolle $S' \subset S$ voidaan muodostaa oma malli, joka tuottaa itsenäisen luokitteluvasteensa niiden muuttujien avulla, jotka sille on valittu. Jos selittäviä muuttujia on esimerkiksi seitsemän $S = \{1,2,..,7\}$, voisi eräs datajoukko olla $S' = \{1,2,3\}$, jonka avulla koulutetaan itsenäinen malli. Luokittelutehtävässä lopullinen lopputulos voidaan määrittää enemmistön perusteella, eli eri mallien luokitukset lasketaan ja lopullinen luokitus on se, joka on saanut eniten ääniä. Virhetermin pienemistä voidaan havainnoida binomijakauman avulla. Jos valinta tehdään binääriluokituksessa, tarvitaan vähintään $n/2$ ääntä, jossa n on luokitusten yhteismäärä:

$$p_{virhe} = \sum_{i=\frac{n+1}{2}}^{n} \binom{n}{i} \theta^i (1-\theta)^{n-i}, \tag{7}$$

Jos yksittäinen malli on väärässä todennäköisyydellä $\theta = 25$ % ja $n = 20$, lopullinen luokitteluvirheen todennäköisyys p_{virhe} on vain 1,39 %. [4]

1.2.1 Satunnaismetsä

Eräs menetelmä, joka luo useita päätöspuita sen sijaan, että etsii tiettyä kriteeriä käyttäen parhaimman puun, kutsutaan satunnaismetsäksi. Menetelmä luo satunnaisia datajoukkoja uudelleennäytteistyksellä (Bootstrap), jossa kullekin alijoukolle valitaan selittävien muuttujien lukumäärän p neliöjuuren $m = \sqrt{p}$ verran selitettäviä muuttujia. Jokaiselle joukolle luodaan oma päätöspuu valitun kriteerin perusteella (esim. Gini-indeksi). Perimmäisenä tarkoituksena on luoda päätöspuita, jotka eivät ole korreloituneita keskenään, mikä tekee niistä monimuotoisempia [5]. Tämä käytännössä vaihtaa suuremman vinouman pienempään varianssiin (Eng. Bias-variance tradeoff), mikä lähtökohtaisesti tuottaa laadukkaamman mallin. Suorittaessa ennustuksia, jokainen puu tuottaa oman luokittelunsa, ja lopullinen luokitus valitaan enemmistön perusteella. [2]

1.2.2 XGBoost

Additiivisen mallin tehostaminen (Eng. boosting) tarkoittaa, että tehdään heikosta mallista parempi. Yleisesti malleja kehitetään derivoitavien tappiofunktioiden avulla, jolloin kyseessä voi olla muun muassa regressio-ongelma, moni- tai binääriluokitteluongelma. Esimerkiksi regressiossa voidaan valita tappiofunktioksi keskineliövirhe ja luokittelussa logaritminen häviö (Eng. log loss). Kun puita lisätään yksitellen malliin, jokaisen puun lisäämisen yhteydessä otetaan askel negatiivisen gradientin suuntaan, eli minimoidaan tappiofunktiota. Tämän kaltaista menetelmää kutsutaan

funktionaaliseksi gradienttiaskeleeksi. Eräs tällainen menetelmä on XGBoost (Extreme gradient boosting), joka lisäksi käyttää useita hyperparametreja. Gradienttiaskeleet johtavat hyvin nopeasti mallin ylisovittamiseen, joten malliin optimoidaan hyperparametreja, joilla hallitaan ylisovittamista ja nopeutetaan laskentaa. Näiden parametrien optimointi on keskeinen osa XGBoost-mallin kouluttamisprosessia. XGBoostiin voidaan asettaa mm. puun maksimisyvyys ja sakkotermi, kuten yksittäisessä rajoitetussa päätöspuussa. Gradienttiaskeleihin pystytään myös asettamaan painoarvo (Learning rate), joka säätelee yksittäisen askeleen vaikutusta lopputulokseen (Learning rate). Yleisesti ottaen XGBoost on nopea ja tuottaa tehokkaita malleja.[6]

XGBoost aloittaa puun rakentamisen yksittäisestä solmusta, lisäten solmukohtia yksi kerrallaan. Konkreettisesti on mahdotonta käydä kaikki puun mahdolliset vaihtoehdot läpi ja valita niistä paras vaihtoehto. Chenin [7] mukaan jaettujen solmukohtien havainnoista $I = I_L \cup I_R$, voidaan muodostaa kaava, jonka avulla voidaan arvioida, kuinka paljon häviö vähenee jaon myötä:

$$\mathcal{L}_{jako} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} + \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma_h. \tag{8}$$

Kaavassa (8) g_i edustaa tappiofunktion derivaattaa, ja h_i toista derivaattaa sekä λ ja γ_h ovat hyperparametreja. Kaavasta nähdään, että kasvattamalla hyperparametreja λ ja γ_h, päätöspuista tulee yksinkertaisempia.

1.3 Mallin hyvyyden arviointi

Sekaannusmatriisi (Eng. Confusion matrix) kuvaa, että kuinka hyvin luokittelumalli suorittaa tehtävänsä esittämällä mallin oikeat ja väärät luokitukset matriisimuodossa. Sekaannusmatriiseista saadaan määritettyä useita eri metriikoita, joilla voidaan arvioida mallin suorituskykyä. Binäärisessä luokituksessa sekaannusmatriisi on kooltaan 2x2. Olkoon tämä matriisi $\mathbf{A}$, ja vastemuuttujien luokat ovat positiivinen ja

negatiivinen. Tällöin matriisin alkio a_{11}, edustaa lukumääriä, joissa malli ennustaa todelliset positiiviset luokat oikein (TP) sekä alkio a_{12} edustaa vääriä positiivisia luokituksia (FP). Alkioon a_{21} on laskettu väärät negatiiviset luokitukset (FN) ja a_{22}:ssa on oikeat negatiiviset luokitukset (TN). Näiden avulla voidaan määrittää taulukossa 1 esitetyt metriikat. Binääriluokituksessa on luonnollista luokitella sille luokalle, joka saa yli puolet äänistä, ja tätä rajaa kutsutaan kynnysarvoksi.

Mallin suorituskykyä kynnysarvon tai muun hyperparametrin suhteen voidaan tarkastella ROC-käyrän (Receiver Operating Characteristic curve) avulla. Päätöspuun tapauksessa ROC-käyrä lasketaan käyttäen kaikkia mahdollisia kynnysarvoja. ROC-käyrän x-akselilla esitetään False Positive Rate, joka kuvaa mallin tekemien väärien positiivisten ennusteiden osuutta kaikista todellisista negatiivisista tapauksista. Y-akselilla on True Positive Rate, joka kuvaa mallin oikein tunnistamien positiivisten tapausten osuutta kaikista todellisista positiivisista tapauksista. Esimerkiksi, jos kynnysarvoksi valitaan nolla, tällöin malli luokittelee kaikki positiivisiksi, ja saadaan xy-koordinaatistoon yksittäinen piste (1,1). ROC-käyrän alle jäävää aluetta kutsutaan AUC-arvoksi (Area under curve). AUC kuvaa, kuinka hyvin malli pystyy erottamaan positiiviset ja negatiiviset luokat kaikilla mahdollisilla kynnysarvoilla.

Metriikka	**Kaava**
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
Specificity	$\frac{TN}{TN+FP}$
F1 Score	$\frac{2}{\text{Precision}^{-1}+\text{Recall}^{-1}}$

Taulukko 1. Eri mittareiden määritelmät. TP edustaa oikein luokiteltuja positiivisia tapauksia, TN sisältää oikein luokiteltuja negatiivisia tapauksia, FP kuvaa väärin luokiteltuja positiivisia tapauksia, ja FN:ään kuuluvat väärin luokitellut negatiiviset tapaukset.

Taulukossa 1 on esitetty eri metriikoiden kaavat. Kaavoista pystyy havaitsemaan, että *accuracy* kuvaa kaikkien oikein ennustettujen tapausten osuutta kaikista tapauksista. *Precision* kuvaa mallin oikein luokiteltujen positiivisten tapausten osuutta kaikista mallin positiivisista luokituksista. *Recall* edustaa oikein ennustettujen positiivisten luokitusten osuutta kaikista todellisista positiivisista tapauksista. *Specificity* kuvaa, kuinka suuren osuuden malli tunnisti oikein kaikista todellisista negatiivisista tapauksista. *F1 score* on määritetty *recallin* ja *precisionin* harmonisena keskiarvona. [4]

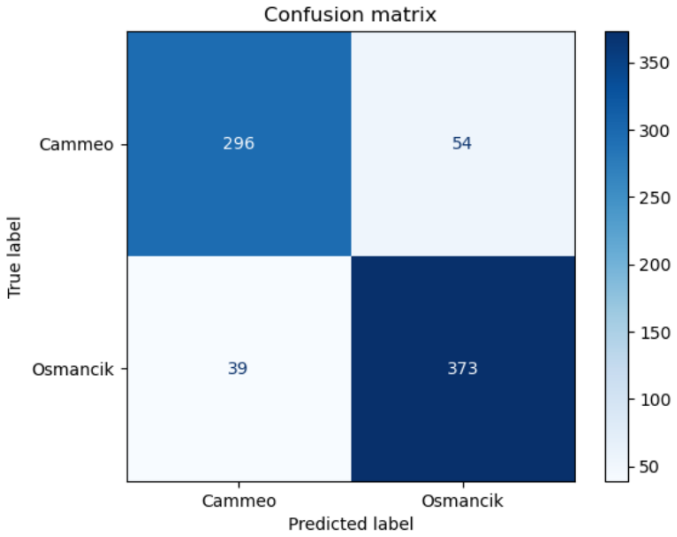
2 Empiirinen toteutus

Työssä käytettiin dataa, jossa on seitsemän erilaista selittävää muuttujaa, joiden avulla pyrittiin ennustamaan riisilajiketta. Kyseessä on binääriluokitteluongelma, sillä näiden tietojen avulla on tavoitteena tunnistaa, onko kyseessä Cammeo- vai Osmancik-riisi. Taulukossa 2 on esitetty selittävät muuttujat lyhyine selitteineen. Datassa on yhteensä 3810 eri havaintoa kullekin eri muuttujalle. [8, 9]

Taulukko 2. Käytettävät selittävät muuttujat

Nro.	Selittävä tekijä	Selite
1	Perimeter	Ympäryysmitta pikseleinä
2	Area	Pikselien lukumäärän riisinjyvän rajojen sisällä
3	Major Axis Length	Pääakselin pituus pikseleissä
4	Minor Axis Length	Sivuakselin pituus pikseleissä
5	Eccentricity	Eksentrisyys
6	Convex Area	Pienimmän koveran alueen pikselien lukumäärä
7	Extent	Pinta-alan ja piirin suhde

Työn koodi on tehty Pythonilla (Versio 3.12.7) käyttäen numpy (Versio 1.26.4) [10],



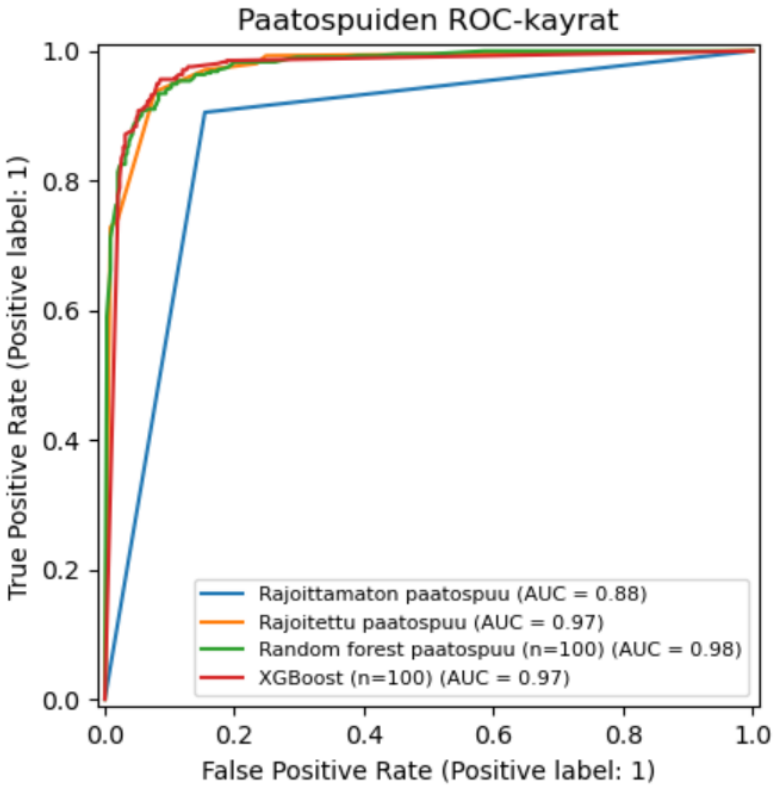
Kuva 2. Rajoittamattoman päätöspuun sekaannusmatriisi.

matplotlib (Versio 3.9.2)[11] sekä scikit-learn (Versio 1.5.1) -kirjastoja [12]. Datasta erotettiin ennen koulutusta 762 datapistettä riippumattomaksi testijoukoksi (20%), jolla lopullinen suorituskyky evaluoitiin kussakin tapauksessa. Tätä testijoukkoa ei käytetty missään koulutuksen vaiheessa, vaan se koostuu mallille ennen näkemättömästä datasta.

2.1 Rajoittamaton päätöspuu

Data jaettiin koulutus- ja testidataan, jossa 80 prosenttia oli koulutukseen ja 20 prosenttia testaukseen. Alustava päätöspuu muodostettiin ilman rajoituksia sklearning -kirjaston DecisionTreeClassifier -funktiolla, jossa kriteeriksi valittiin kokonaisgini-indeksi. Tällöin päätöspuusta tuli erittäin suuri, eli puun syvyydeksi tuli 23 eri tasoa.

Kuvassa 3 on esitetty edellä mainitun päätöspuun ROC-käyrä sekä lisäksi laskettu tämän AUC-arvo, joka on 0,88. Käyrä koostuu käytännössä kahdesta eri suorasta, joiden taitteessa malli saavuttaa parhaan tasapainon oikeiden ja väärien positiivisten tulosten välillä. Tämä on piste, jossa false positive rate on noin 0,15 ja



Kuva 3. Kaikkien mallien ROC-käyrät sekä AUC-arvot. Y-akselilla kuvataan mallin tunnistamien Cammeo-luokistusten suhde kaikista todellisista Cammeo-luokista. X-akselilla on esitetty väärien Osmancik-luokitusten suhde kaikista todellisista Osmancik-luokista.

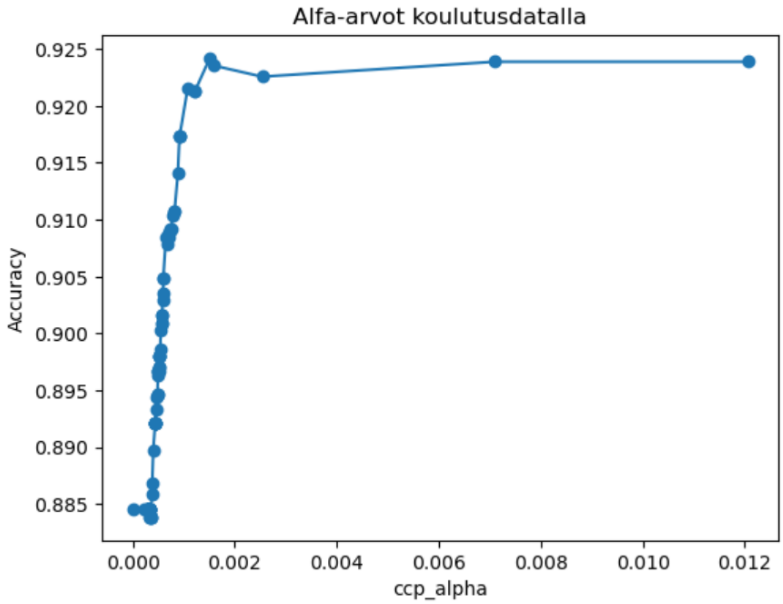
Taulukko 3. Kaikkien mallien lasketut arvot.

Malli	Accuracy	Precision	Recall	Specificity	F1 score	AUC
Rajoittamaton päätöspuu	0.878	0.884	0.846	0.905	0.864	0.88
Rajoitettu päätöspuu	0.930	0.926	0.923	0.937	0.924	0.97
Random forest	0.925	0.922	0.914	0.934	0.918	0.98
XGBoost	0.930	0.926	0.923	0.937	0.924	0.97

true positive rate on 0,90. Tällöin malli luokitteli Cammeoniksi 90 % oikein kaikista todellisista Cammeo-luokista ja 15 % luokiteltiin väärin kaikista todellisista Osmanikeista. Kuvassa 2 on esitetty päätöspuun sekaannusmatriisi, jossa oikeita Cammeo-luokituksia oli tehty 296 kappaletta ja vääriä 39 kappaletta. Lisäksi oikeita Osmancik luokituksia oli tehty 373 ja vääriä 54 kappaletta. Näiden arvojen avulla pystyttiin määrittämään taulukossa 3 lasketut arvot. Oikeita luokitteluita malli teki 87,9 % (*Accuracy*). Mallin kaikista Cammeo luokituksista olivat 88,4 % oikein (*Precision*). Kaikista todellisista Cammeo-lajikkeista malli tunnisti 84,6 % osuuden (*Recall/sensitivity*). Kaikista Osmancik-lajikkeista malli tunnisti 90,5 % (*Specificity*). *F1 scoren* (86,4 %) arvosta pystyttiin päättelemään, että luokittelujen välillä malli on hyvin tasapainoinen. Edellä laskettujen arvojen valossa, pystyttiin arvioimaan mallin suoriutuvan hyvin luokittelusta.

2.2 Rajoitettu päätöspuu

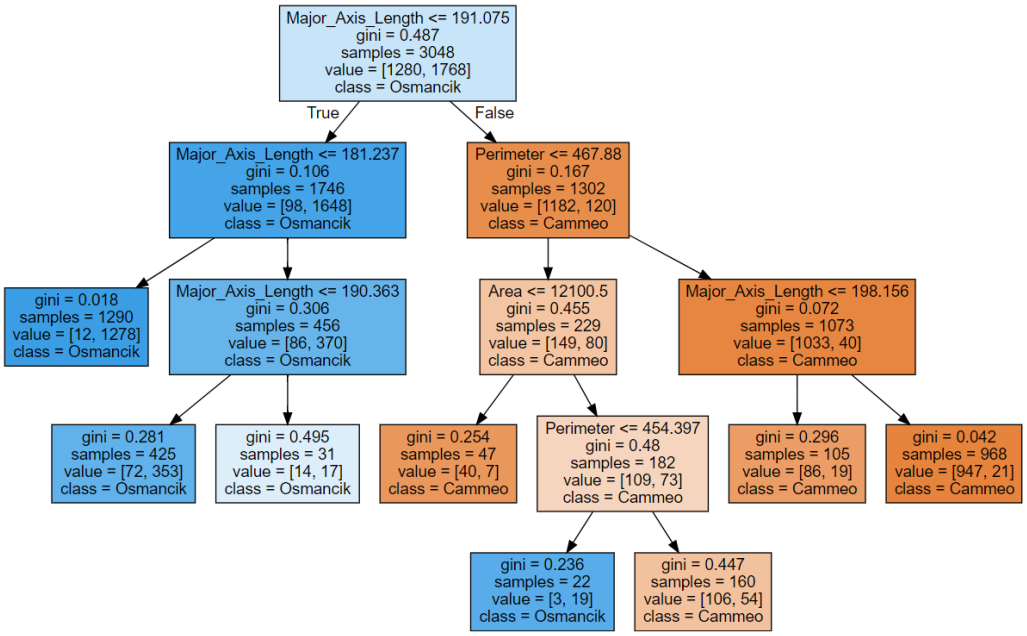
Päätöspuun rajoittavana tekijänä pyritään löytämään sellainen α :n arvo, joka yksinkertaistaa mallia, eli sen syvyyttä ja luonnollisesti pyritään samalla myös siihen, että mallin suorituskyky paranee. Menetelmänä käytettiin cost-complexity pruning -menetelmää. Eri α:n arvot määritettiin käyttäen Pythonin scikit-learn -kirjaston cost_complexity_pruning_path -funktiota, joka palauttaa tehokkaat α:n arvot (α_{eff}). Koulutusdata jaettiin viiteen osaan, jossa yksi toimi validointiaineistona ja loput koulutusaineistona. Validointiaineistoa vaihtaen koulutettiin ja testattiin päätöspuu viisi kertaa, tietyllä α_{eff} arvolla (ristiinvalidointi). Lopuksi laskettiin kullekin mallille saatujen tarkkuuksien keskiarvo, jolloin saatiin arvio päätöspuun suorituskyvystä tietyllä α_{eff} -arvolla. Tämä toimenpide toistettiin kaikilla α_{eff} -arvoilla. Visuaalisesti tätä havainnollistettiin päätöspuiden tarkkuutena *ccp_alpha* parametrin funk-



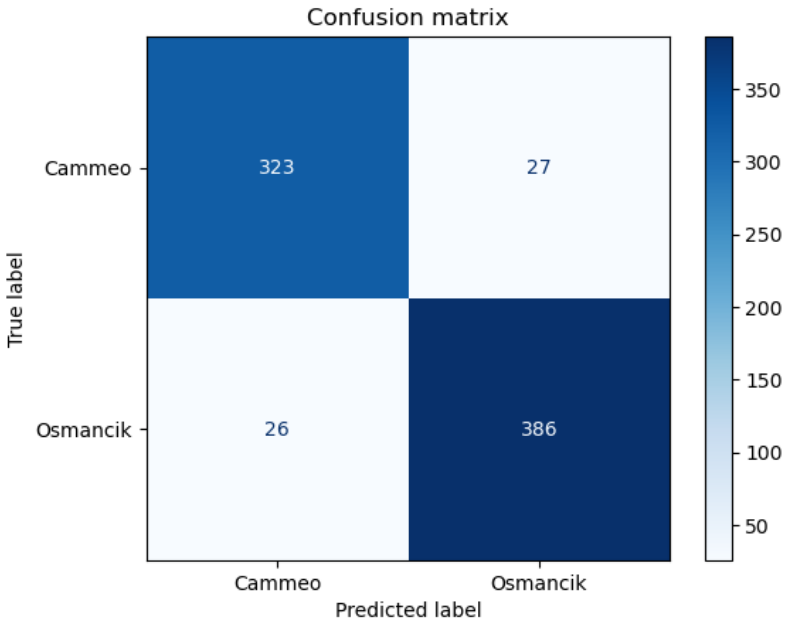
Kuva 4. Tarkkuudet α:n funktiona

tiona, joka on esitetty kuvassa 4. Kuvaajasta nähdään, että tarkkuus saa suurimman arvon, kun *ccp_alpha* on silmämääräisesti noin 0,0015.

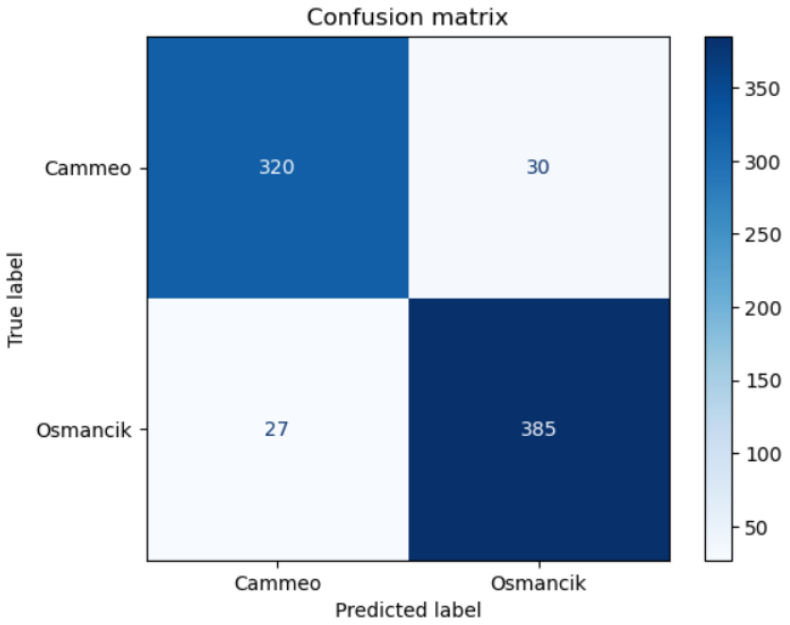
Tämän jälkeen luotiin uudestaan päätöspuu, sillä erolla, että asetettiin *ccp_alpha* -parametri. Tällöin päätöspuun maksimisyvyys tippui neljään, joka on esitetty kuvassa 5. Kaikkiin päätöspuun solmukohtiin on laskettu kynnysarvot, koska kaikki havainnot ovat jatkuva-arvoisia. Huomionarvoista on se, että kaksi solmukohtaa jakaantuu kahteen luokittelulehteen, jotka luokittelevat havainnot samaan luokkaan. Näin ollen nämä solmut voisi surutta vaihtaa yhteen luokittelulehteen. Eli rajoittava parametri *ccp_alpha* ei täysin ole karsinut puuta minimiin. Rajoitetun mallin sekaannusmatriisista (Kuva 6) sekä taulukon 3 arvoista huomattiin, että kokonaistarkkuus lisääntyi. Rajoittamattomaan puuhun verrattuna *precision* ja *recall*-arvot nousivat, joten rajoitettu päätöspuu vaikuttaisi suoriutuvan luokittelusta paremmin. *F1 scoren* kasvu myös kielii siitä, että malli on erinomaisessa tasapainossa eri riisilajien tunnistamisessa. Lisäksi ROC-käyrästä nähdään, että AUC-arvo on noussut 0,88 :sta 0,97 :ään. Rajoitus muodosti pehmeämmän ROC-käyrän verrattuna rajoittamattomaan malliin.



Kuva 5. Rajoitettu päätöspuu



Kuva 6. Rajoitetun päätöspuun sekaannusmatriisi



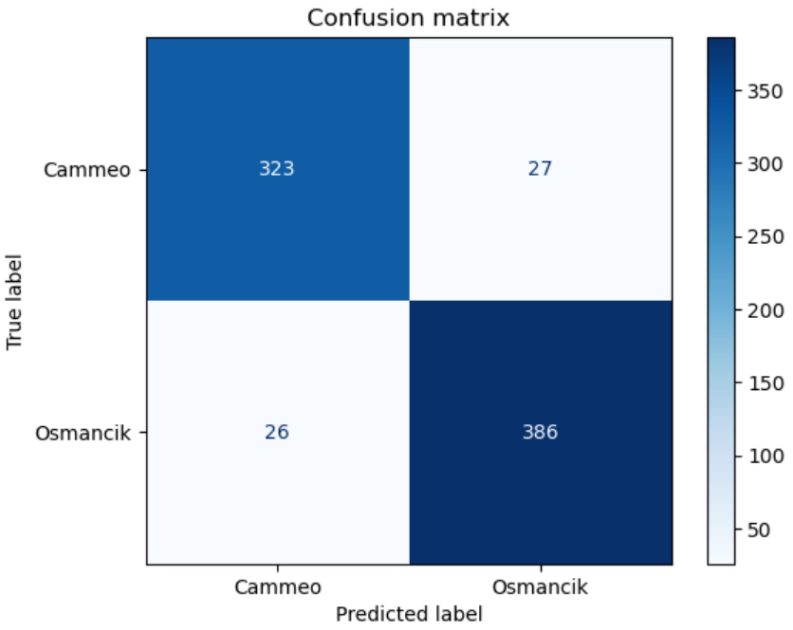
Kuva 7. Satunnaismetsän sekaannusmatriisi.

2.3 Satunnaismetsä

Kun muodostettiin malli käyttäen satunnaismetsä -menetelmää, jossa luotiin sata erilaista päätöspuuta, saatiin sekaannusmatriisiksi hyvin samankaltainen, kuin rajoitetun päätöspuun tapauksessa (Kuva 7). Satunnaismetsässä ei otettu käyttöön päätöspuuta rajoittavaa parametria. Sekaannusmatriisin perusteella saatiin tarkkuudeksi hieman pienempi kuin yksittäisen rajoitetun päätöspuun tapauksessa (Taulukko 3). Satunnaismetsä-menetelmässä laskivat myös *precision*, *recall*, *specificity* sekä *F1-score* -arvot muutaman sadasosan verran. Satunnaismetsän ROC-käyrän havaittiin tuottavan suurimman AUC-arvon.

2.4 XGBoost

Tämän työn XGBoost -mallissa käytettiin Grid search -optimointimenetelmää, jolla haettiin karkeasti parhaat γ_h:n ja λ:n arvot. Kyseinen menetelmä on hyvin suora-



Kuva 8. XGBoost-mallin sekaannusmatriisi.

viivainen parametrien optimointimenetelmä, jossa syötetään parametreille eri kandidaattiarvoja, joista scikit-learn -kirjaston GridSearchCV -funktio valitsee parhaan kombinaation. Kuten random forest -menetelmässä, myös XGBoostissa käytettiin sataa päätöspuuta luomaan lopullinen malli. Kuvassa 8 on esitetty XGBoost -mallin tuottama sekaannusmatriisi, josta huomataan tämän olevan identtinen rajoittavan päätöspuun sekaannusmatriisin kanssa. Tämän vuoksi myös taulukossa 3 lasketut arvot ovat samat. XGBoostin AUC-arvo oli hieman pienempi kuin satunnaismetsän (Kuva 3), mutta tällä ei juurikaan ole käytännön merkitystä.

3 Yhteenveto

Kaikkien tässä työssä käytettyjen mittareiden valossa, rajoitettu päätöspuu, satunnaismetsä sekä XGBoost -menetelmät pärjäsivät merkittävästi paremmin, kuin rajoittamaton päätöspuu. Satunnaismetsän muut arvot (*Accuracy*, *recall*, jne.) laskivat sadasosien verran, mutta AUC-arvo kasvoi sadasosan verran. Kuvasta 2 nähdään, että random forest ja XGBoost pärjäävät tietyllä alueella rajoitettua päätöspuuta paremmin, muuten kaikkien ROC-käyrät ovat koko matkalta lähes päällekkäin. Metriikoiden ja tässä työssä käytetyn datan perusteella XGBoost ja rajoitettu päätöspuu tuottivat parhaimmat mallit. Kuitenkin satunnaismetsä tuotti hyvin varteenotettavan mallin, koska tässä ei käytetty mitään parametrien optimointimenetelmää tai päätöspuuta rajoittavaa parametria. Toisaalta, XGBoostin parametrien optimointimenetelmä oli hyvin naivi, johon panostamalla saadaan todennäköisesti tuotettua entistäkin parempi malli.

Viitteet

[1] T. Hastie, R. Tibshirani ja J. Friedman, *Elements of Statistical Learning: Data Mining, Inference, and Prediction*, second edition ed. (Springer, 2009).

[2] A. Geron, *Hands-on machine learning with Scikit- Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems*, 3rd edition. ed. (O'Reilly, 2022).

[3] L. Breiman, *Classification and regression trees*, *Wadsworth statistics/probability series* (Wadsworth International GroupBelmont, CA, 1984).

[4] A. R. A. R. Webb ja K. D. Copsey, *Statistical pattern recognition*, 3rd ed. ed. (WileyChichester, 2011).

[5] G. James, D. Witten, T. Hastie ja R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*, second edition. ed. (Springer, 2021).

[6] J. Brownlee, *XGboost with Python* (Machine Learning Mastery, 2016).

[7] T. Chen ja C. Guestrin, kirjassa *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *KDD '16* (Association for Computing MachineryNew York, NY, USA, 2016), p. 785–794.

[8] I. Cinar ja M. Koklu, International Journal of Intelligent Systems and Applications in Engineering **7**, 188–194 (2019).

[9] Rice (Cammeo and Osmancik), UCI Machine Learning Repository, 2019, dOI: https://doi.org/10.24432/C5MW4Z.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke ja T. E. Oliphant, Nature **585**, 357 (2020).

[11] J. D. Hunter, Computing in Science & Engineering **9**, 90 (2007).

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot ja E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).