Tampereen ammattikorkeakoulu

Liiketoiminnan ennustettavuuden parantaminen koneoppimisen avulla

Simo Rautiainen

OPINNÄYTETYÖ
Lokakuu 2021

Tieto- ja viestintätekniikka
Sulautetut järjestelmät ja elektroniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Sulautetut järjestelmät ja elektroniikka

Rautiainen Simo:
Liiketoiminnan ennustettavuuden parantaminen koneoppimisen avulla

Opinnäytetyö 56 sivua, joista liitteitä 0 sivua
Lokakuu 2021

Opinnäytetyö on tehty Ambientia Oy:lle. Työn tavoitteena on löytää koneoppimisen avulla ennustava funktio, joka pyrkii mallintamaan Ambientia Oy:n liiketoimintadataa mahdollisimman hyvin. Ennustava funktio toimisi jatkokehitystä varten pohjana liiketoimintaa parantaville sovelluksille. Liiketoimintaa parantava sovellus voisi olla suosittelija, joka ennustavan funktion avulla suosittelisi käyttäjälle todennäköisimmän vaihtoehdon.

Työn ensimmäisessä kappaleessa käydään työn tavoite tarkemmin läpi. Toisessa kappaleessa selitetään teoria eri koneoppimisalgoritmien takana. Kolmannessa kappaleessa tehdään koneoppimismallit ja neljännessä kappaleessa mitataan ne. Viimeisessä kappaleessa käydään läpi työn tulokset lyhyesti ja jatkokehitys mahdollisuudet.

Työssä saatiin tehtyä ennustavat algoritmit koneoppimisen avulla. Parhaaksi malliksi saatiin päätöspuu, jonka hyperparametrit ovat optimoitu. Jos malleista halutaan tarkempia, tulee dataotoksia olla enemmän.

Asiasanat: ennustaminen koneoppiminen ohjelmointi algoritmi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT engineering
Embedded systems and electronics

Simo Rautiainen:
Improving business forecasting with machine learning

Bachelor's thesis 56 pages, appendices 0 pages
October 2021

Thesis is made for Ambientia Ltd. Goal of the thesis is to find a predictive function with machine learning that aims to model Ambientia Ltd's business data as good as possible. Predictive function would be a base for business improving applications in the future development. This business improving application could be a recommender that uses the predictive function to recommend most likely option to a user.

Thesis's first section covers the goal of the thesis in more detail. Theory behind the machine learning algorithms is explained in the second section. In third section of the thesis the machine learning models are made and in fourth models are measured. Last section goes through the results of the thesis and explores future development options.

Predictive algorithms were made with machine learning in the thesis. Best model was decision tree algorithm with optimized hyperparameters. If more precise models are needed, must there be more samples in the data.

Key words: predicting machine learning programming algorithm

SISÄLLYS

ERITYISSANASTO

Piirre – Tarkoittaa datassa yhtä kolumnia.
Koneoppimismalli – Säädetty koneoppimisalgoritmi
Hyperparametri – Koneoppimismallia tehdessä säädettävä parametri
Otos – Yksi rivi taulukossa
Ylisovitus – Kun malli ennustaa suoraan opetusdatalla
Epoch – Yksi neuroverkon opetussykli
TNR – Oikeasti negatiivisten suhde
TPR – Oikeasti positiivisten suhde
FPR – Väärien positiivisten suhde
ROC – Receiver operating characteristic

1 JOHDANTO

Opinnäytetyössä tutkitaan neuroverkkojen hyödyntämistä yrityksen liiketoiminnan ennustettavuuden parantamisessa. Opinnäytetyön tulokseksi pyritään löytämään ja toteuttamaan koneoppimismalli, jolla voidaan parantaa työn tilaajana toimivan Ambientia Oy:n liiketoiminnan ennustettavuutta.

Työn tavoitteena on löytää paras koneoppimismalli, joka pystyisi ennustamaan mahdollisimman korkealla tarkkuudella liiketoiminnan ennustettavuuteen tarpeellisia muuttujia. Jos opinnäytetyön avulla saadulla sovelluksella saadaan tarpeeksi tarkkoja tuloksia, laitetaan sovellus ajoon ja tarjoamaan arvoa Ambientia Oy:lle.

Työssä käytettävä data on luottamuksellista, joten sitä ei esitellä yksityiskohtaisesti työn missään vaiheessa. Käydään kuitenkin läpi datan muokkaamista oikeaan muotoon, jotta dataa voidaan hyödyntää käytettävien koneoppimismallien kanssa mahdollisimman hyvin.

Teoriassa käydään läpi eri koneoppimismalleja ja niiden mittaukseen huomioitavia asioita. Toteutuksessa tehdään koodiesimerkein teoriassa käydyt koneoppimismallit, mitataan mallit ja lopuksi päätetään paras malli.

Työ sopii hyvin koneoppimisesta kiinnostuvalle tekniikan alan ihmiselle ja voi antaa uusia oivalluksia koneoppimismallien suunnittelusta tai datan muokkauksesta. Työssä ei syvennytä algoritmien taustalla olevaan matematiikkaan, vaan keskitytään toteuttamaan malli käyttäen olemassa olevia avoimen lähdekoodin kirjastoja.

2 Teoria

Koneoppiminen on tekoälyn alakäsite, johon kuuluu sellaiset tilastolliset menetelmät, jotka hyödyntävät datasta opittuja suhteita. Algoritmi löytää itse datasta suhteet, eikä niitä tarvitse itse ohjelmoida. (Aisolab 2017)

Tarkennettuna, koneoppiminen tarjoaa algoritmeja, jotka pystyvät ratkaisemaan luokittelu- ja regressiotehtäviä automatisoimalla ennustamiseen tarvittavat toimenpiteet edellisten datapisteiden perusteella. Koneoppimisen tavoite ei ole kuitenkaan vaan tuottaa algoritmeja, jotka tuottavat tarkkoja ennustuksia, vaan sitä voidaan myös hyödyntää ennustamisrakenteen ymmärtämiseen. Esimerkiksi, mitkä mittapisteet ovat tärkeitä ja minkälaisella painolla ne vaikuttavat algoritmin ennustamiseen. (Louppe 2014, 2)

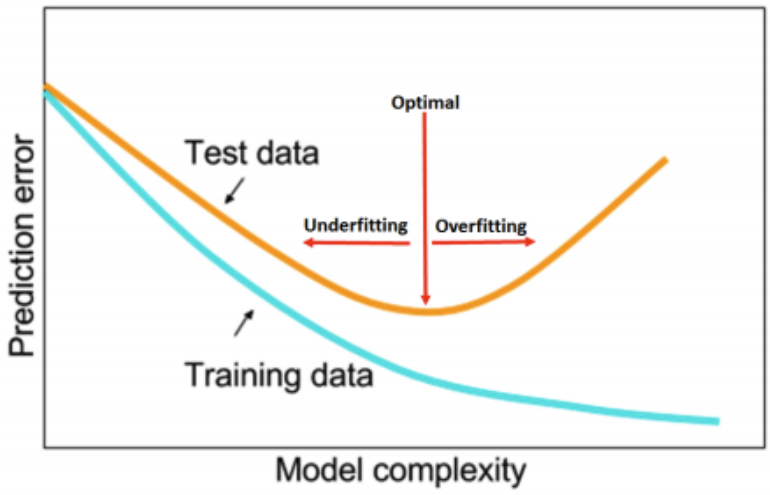
Ohjattu oppiminen on systemaattisen tavan etsimistä, jolla voidaan tehdä ennustuksia annettujen mittausten perusteella. Ohjatussa oppimisessa tehdään malli, joka ennustaa yhtä piirrettä muiden piirteiden avulla. (Louppe 2014, 2)

On olemassa myös ei-ohjattu oppiminen, jossa dataa ei ole jaettu piirteisiin, vaan järjestelmälle annetaan sisääntulona raakaa dataa. Ei-ohjattua oppimista voidaan hyödyntää, jos halutaan löytää käytetystä datasta sen rakennetta. (Mishra 2017)

Vahvistusoppimisessa taas käsitellään dynaamista ympäristöä, jossa pyritään saavuttamaan tietty päämäärä. Vahvistusoppimista on esimerkiksi auton ajaminen tai shakin pelaaminen. Vahvistusoppiminen toimii takaisinkytkennällä, joka hyödyntää rankaisemista ja palkitsemista yrittäen saada parhaimman mahdollisen ulostulon. (Mishra 2017)

Koneoppimistehtävät, joissa yritetään approksimoida ennustava funktio tulo- ja lähtöpiirteillä, voidaan jakaa luokittelu- ja regressiotehtäviin. Luokittelutehtävässä lähtöpiirteessä voi olla kaksi tai useampi eri luokkaa. Regressiotehtävässä lähtöpiirre on jatkuva muuttuja, esimerkiksi kokonais- tai liukuluku. Teoriassa keskitytään luokittelualgoritmeihin, koska opinnäytetyössä ratkaistava tehtävä on luokittelutehtävä. (Brownlee 2019)

Koneoppimisalgoritmeja on olemassa todella yksinkertaisia ja todella monimutkaisia. Useaa koneoppimisalgoritmia voidaan kuitenkin säätää hyperparametreilla, jotta niistä saataisiin monimutkaisempia. Kuviossa 1 on Kumar A. piirtämä kuvaaja (2020) ennustuksen virhe mallin monimutkaisuuden funktiona (Kumar 2020).



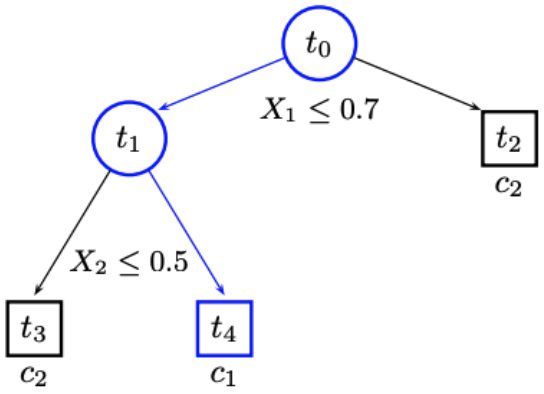
KUVIO 1. Ennustuksen virhe mallin monimutkaisuuden funktiona. (Kumar 2020)

Kuten kuviosta 1 voidaan huomata, että opetusdatan ennustuksen virhe paranee mitä monimutkaisempi malli on. Kuitenkin optimaalisen kohdan jälkeen ennustuksen virhe testidataa vasten huononee. Olisi ideaalia, jos optimaalinen kohta mallin monimutkaisuudessa pystyttäisiin löytämään. Tämän takia on tärkeää kokeilla erilaisia järkeviä parametrien yhdistelmiä eri malleilla, jotta kuvion 1 optimaalinen kohta voitaisiin löytää.

2.1 Päätöspuu (Decision tree)

Päätöspuilla voi mallintaa monimutkaisia yhteyksiä tulojen ja lähtöjen välillä ilman mitään ennakkotietoa datasta. Ihmisten on myös helppo selvittää, miten päätöspuu tekee ennustuksensa, vaikka ei olisikaan tilastollinen henkilö. (Louppe 2014, 26)

Kuviossa 2 on Louppe Gillesin piirtämä kuva päätöspuusta.

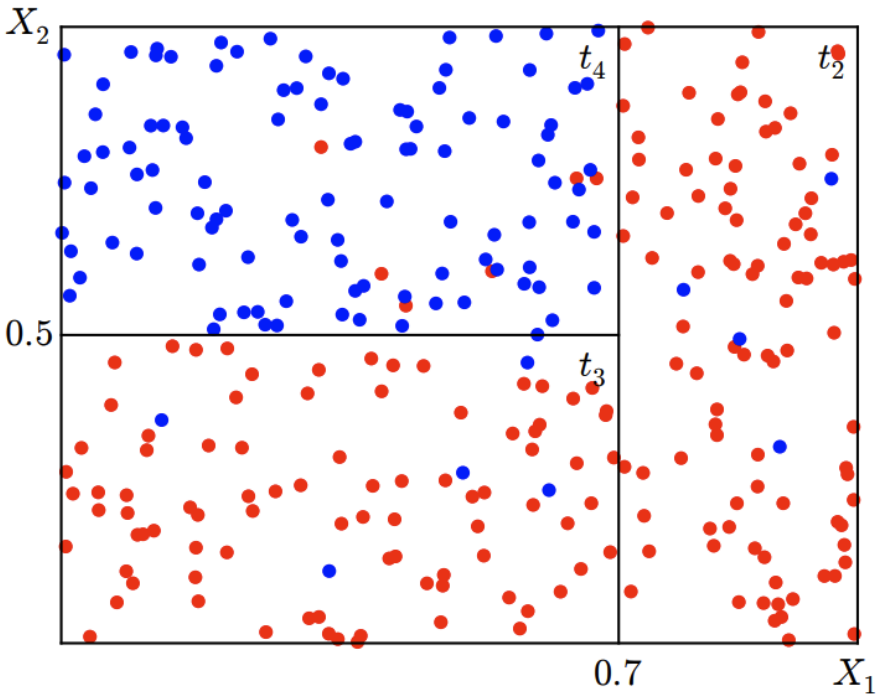


KUVIO 2. Yksinkertainen päätöspuu (Louppe 2014, 28)

Kuten kuviosta 2 näkee, päätöspuussa on 5 solmua joista 2 jakaantuu eteenpäin. Solmut on nimetty kuvassa t_x tyylisesti, jossa *x* on jokin kokonaisluku ja lehtisolmut on merkitty C_x tyylisesti, jossa x on myös joku kokonaisluku.

Päätöspuu kuvataan yleisesti siten, että jakautuvassa solmukohdassa toteutuva väite jakautuu vasemmalle ja ei-toteutuva oikealle. Tällä menettelyllä henkilölle entuudestaan tuntematon päätöspuu on luettavissa helpommin. Kuvion 2 t_0 kulkeutuu vasemmalle solmuun t_1, jos X_1 on pienempi tai yhtä suuri kuin 0,7 ja muussa tapauksessa se menee oikealle lehtisolmuun t_2. t_1 solmusta kulkeudutaan t_3 lehtisolmuun, jos X_2 on pienempi tai yhtä suuri kuin 0,5 ja muussa tapauksessa t_4 lehtisolmuun. t_3 ja t_2 lehtisolmut ovat päätöspuun lähtöluokkia C_2 ja t_4 lehtisolmu on mallin C_1 lähtöluokka.

Kuvion 2 päätöspuuta voidaan selittää vielä paremmin Louppe Gillesin piirtämällä kuviolla, joka on esitetty kuviossa 3. Kuviossa 3 on kuvion 2 päätöspuu piirretty koordinaatistoon, jossa kuvion 2 C_1 luokka on sinisen väriset ympyrät ja C_2 punaisen väriset ympyrät.



KUVIO 3. Päätöspuu piirrettynä koordinaatistoon (Louppe 2014, 29)

Kuviosta 3 voidaan hyvin visualisoida kuvion 2 päätöspuu. X_2 on y-akselilla ja X_1 on x-askelilla. Päätöspuulla voidaan rajata siniset ympyrät ulos punaisista ympyröistä jakamalla datapisteet kuvion 2 mukaisiin solmuihin.

2.1.1 Päätöspuun hyperparameterit

Päätöspuulle voidaan antaa hyperparametrinä puun maksimi syvyys, eli kuinka monta jakaantuvaa solmua puussa on (Scikit-learn -kirjaston dokumentaatio. n.d.).

Päätöspuulle voidaan myös antaa hyperparametrina solmun minimimäärä otoksia, jotta solmu voidaan jakaa (Scikit-learn -kirjaston dokumentaatio. n.d.). Esimerkiksi jos solmun minimi määrä otoksia on 5 niin jokaisesta solmusta, jolle jää alle 5 otoksia, tulee suoraan lehtisolmu.

Viimeinen työssä käytetty hyperparametri päätöspuulle on lehtisolmun minimimäärä otoksia, eli lehtisolmuun on kuuluttava vähintään hyperparametrin verran otoksia, jotta sen ylemmän tason solmun jakaminen menee läpi (Scikit-learn-kirjaston dokumentaatio. n.d.). Esimerkiksi jos lehtisolmun minimi määrä otoksia on

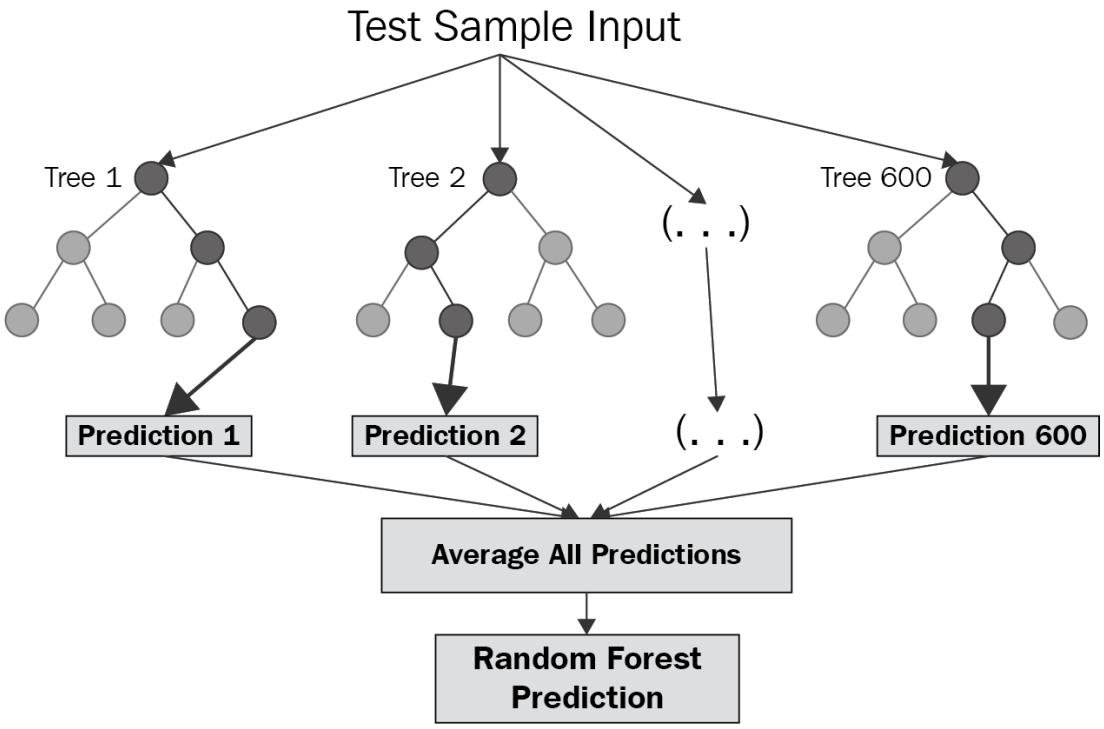
2 otosta ja solmu yritetään jakaa kahteen solmuun siten, että toiseen solmuun jää vain 1 otos ja toiseen 99 otosta, niin kyseinen jako estetään.

2.2 Satunnainen metsä ja erittäin satunnaiset puut

Vaikkakin päätöspuut tarjoavat selitettävyyttä ja niitä on helppo käyttää, ne ylisovittuvat helposti, sekä niiden rakenne muuttuu vaikkakin opetus dataa muokattaisiin vain hieman. Päätöspuut eivät pysty yleistämään mallia dataan ja ne kärsivät rajoitetusta tarkkuudesta. (Ho 2020, 1)

Satunnaiset metsät tuovat edellä mainittuun ongelmaan ratkaisun. Niitä käytetään myös ennustukseen ja niillä saadaan selvitettyä datan rakennetta. Satunnaiset metsät muodostuvat useista päätöspuista, joista jokainen tekee oman ennustuksensa ja äänestää omaa lopputulostaan. Eniten ääniä saanut lopputulos valitaan lähtöluokaksi. (Corporate Financial Institute 2021)

Chakure Afrozin (2019) piirtämä kuva satunnaisen metsän rakenteesta on esitetty kuviossa 4.



KUVIO 4. Satunnaisen metsän rakenne (Chakure 2019)

Kuvion 4 mukaisesti jokainen päätöspuu ottaa satunnaisen otoksen tulodatasta, jonka pitäisi estää ylisovittumista opetusdataa vasten.

Kun satunnainen metsä -algoritmi jakaa tulo-otokset satunnaisiin osiin päätöspuille, erittäin satunnaisten puiden -algoritmi käyttää päätöspuillensa kaikkia tulo-otoksia. Se mikä tekee erittäin satunnaisen puiden puista satunnaista, on se, että se jakaa solmujen leikkaus kohdat täysin satunnaisesti, eikä laske päätöspuiden tapaan optimaalista solmun leikkaus kohtaa. Tämän pitäisi pienentää ennustuksien varianssia ja biasta. (Pierre & Damien & Louis 2005)

2.2.1 Satunnaisen metsän ja erittäin satunnaisten puiden hyperparametrit

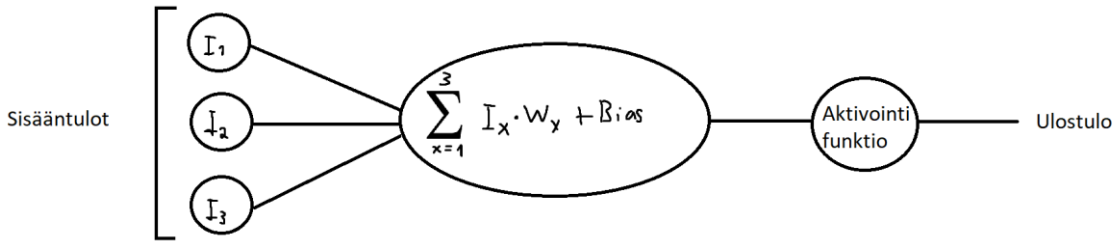
Satunnaisille metsille voidaan antaa hyperparametrina samat hyperparametrit, kun Päätöspuille, paitsi myös lisäksi määrän päätöspuita, jotka tekevät äänestyksen parhaasta vaihtoehdosta (Scikit-learn -kirjaston dokumentaatio. n.d.).

Erittäin satunnaisten puiden hyperparametrit ovat täysin samat kuin satunnaisille metsille (Scikit-learn -kirjaston dokumentaatio. n.d.).

2.3 Neuroverkot

Neuroverkot ovat koneoppimismalleja, jotka mallintavat ihmisen aivojen rakennetta ja toimintaa. Tekoälyneuroni prosessoi sen tuloarvoja laskemalla niille painotetun summan ja tuottaa lähtösignaalin suoraan tai voidaan valita, että ulostulo arvo tuotetaan, jos se ylittää jonkun kynnysarvon. Neuroniin voidaan lisätä myös arvo, jota sanotaan bias-arvoksi. Neuronin ulostulo voi mennä vielä seuraavalle tasolle neuroneja tai se voi olla suoraan mallin ennuste. (Aisolab 2017)

Kuviossa 5 on esitetty yhden neuronin toimintaperiaate neuroverkossa.



KUVIO 5. Yhden neuronin toimintaperiaate yksinkertaistettuna

Kuviosta 5 nähdään yhden neuronin toimintaperiaate. Summan laskemiseen käytetään jokaista tuloa painotettuna. I_x on tulo ja W_x on sen paino. Summan ulostulo menee aktivointi funktiolle, joka muuttaa sille annetun arvon tietyn funktion mukaisesti. Esimerkiksi Sigmoid -aktivointifunktio antaa arvon 0-1 välillä.

Yksinkertaisessa luokittelevassa neuroverkossa on tulotaso, jonka neuronit ovat esimerkiksi kuvan pikselit. Sitten on jokin määrä piilotettuja tasoja, ja lopulta on ulostulo taso, jonka neuronien määrä on sama, kun haluttujen vaihtoehtojen määrä. Edeltävän tason neuronien ulostulot menevät jokaiseen seuraavan tason neuroniin tuloina.

Neuroverkon viimeisestä tasosta menee takaisinkytkentä malliin, joka muuttaa neuronien painoja ja bias-arvoja siten, että se yrittää saada viimeisen tason ennustuksista mahdollisimman oikeita minimisoimalla häviöfunktion. Häviöfunktion tuloarvot ovat mallin ennustukset ja oikeat arvot, joiden perusteella häviöfunktio antaa lähtöön arvon. Neuronien painojen ja bias-arvojen muuttamisen oikeaan suuntaan hoitaa optimointialgoritmi ja kutstutaan opettamiseksi.

Neuroverkoissa epoch tarkoittaa yhtä opetussykliä, jossa malli opetetaan kerran datalla. Neuroverkkoja yleensä opetetaan useamman epochin verran, jolloin neuroverkko näkee saman datan useamman kerran ja yrittää löytää mahdollisimman yleistävät painot ja bias-arvot neuroneille. (DeepAi – Epoch. n.d.)

2.3.1 Neuroverkon hyperparametrit ja suunnittelu

Neuroverkossa on paljon säädettäviä hyperparametrejä. Neuroverkoille voi asettaa jokaiselle tasolle sen neuronien määrän ja aktivointifunktion. Koko neuroverkolle voi asettaa muun muassa sen häviöfunktion, optimointi, opetusvauhdin ja piilotettujen tasojen lukumäärän.

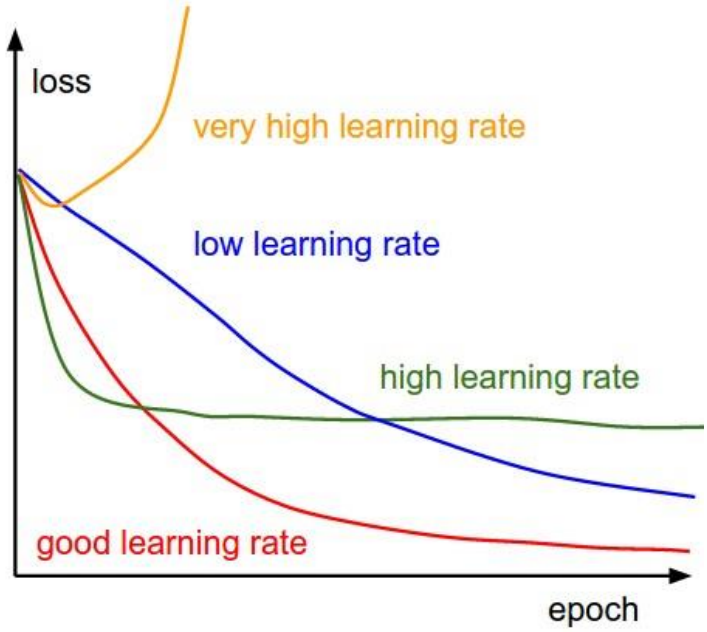
Shuklan mukaan (2019) neuroverkon neuronien määrä kannattaa olla 1-100 luokassa ja piilotettujen tasojen määrä kannattaa olla 1-5.

Neuroverkoissa voidaan jakaa tulonäytteet eriin, jolloin neuroverkon painoja ja biaksia säädetään erä tulonäytteitä kerrallaan, eikä perinteisesti joka tulonäytteellä. Tätä hyperparametriä kutsutaan erän kooksi ja siihen Shukla suosittelee (2019) väliä 2-32 (Shukla 2019).

Aktivointifunktioksi Shukla suosittelee piilotetuille tasoille SELU-aktivointifunktiota ja viimeiselle tasolle, eli ulostulotasolle Sigmoid -aktivointifunktiota, mikäli kyseessä on binäärinen luokittelutehtävä. Sigmoid -aktivointifunktio on hyvä ulostulotasolle, koska se antaa numeron 0 ja 1 välillä, jolloin voidaan jakaa binäärisessä luokittelu tehtävässä luokat helposti. Esimerkiksi luokka 1 (> 0.5) ja luokka 2 (<=0.5). (Shukla 2019)

Neuroverkon optimointialgoritmiksi Shukla suosittelee (2019) kokeilemaan montaa eri vaihtoehtoa, joita on mm. SGD, eli Stochastic gradient descent, ja Adam. Adam -optimointialgoritmi on SGD-optimointialgoritmin lisäosa (Brownlee 2017).

Neuroverkkojen opetusvauhti on tärkeä hyperparametri, koska jos sen asettaa väärin, voi neuroverkko olla täysin turha tai neuroverkko ei pääsee ikinä täyteen potentiaaliinsa. Shukla on piirtänyt (2019) kuvaajan opetusvauhdin merkityksestä mallinhäviöön, joka on kuviossa 6.



KUVIO 6. Opetusvauhdin merkitys häviöön epochien funktiona (Shukla 2019)

Kuviossa 6 on y-akselilla mallin häviö ja x-akselilla epochien lukumäärä. Mallin häviö halutaan minimoida, kun mallin opettaminen on ohi. Eli kuvion 6 mukaan hyvä opetusvauhti on sellainen, jossa epochien lisääntyessä mallin häviö pienenee punaisen käyrän mukaisesti.

Shukla mukaan (2019) hyvä opetusvauhti löydetään aloittamalla pienestä arvosta 10^{-6} ja kerrotaan sitä arvoa vakiolla, kunnes opetusvauhdin arvo on 10. Näistä opetusvauhdin arvoista valitaan se, joka tuottaa kuvion 6 punaisen opetusvauhdin mukaisen käyrän. (Shukla 2019)

SGD-optimointialgoritmissa voi valita opetusvauhdin lisäksi myös momentum -hyperparametrin, joka nopeuttaa SGD algoritmissa gradienttien menemistä oikeaan suuntaan. Momentum -hyperparametriin voi valita arvon 0 ja 1 välillä, johon Bushaev valitsee (2019) arvon 0.9, koska 0.9 on yleisesti muidenkin SGD-optimointialgoritmia käyttävien mielestä hyvä arvo. (Bushaev 2017)

2.4 Ohjelmointi

Koneoppimismallien implementoinnin nopeuttamiseksi ohjelmointikielestä olisi hyvä löytyä koneoppimista ja datankäsittelyä varten tehtyjä kirjastoja. Tämän

opinnäytetyön toteutusta varten aiemmin esitellyt algoritmit tulisi löytyä valitusta ohjelmointikielestä.

Python-ohjelmointikieli on suosittu valinta, koska sitä on nopea kirjoittaa ja siihen on tehty avoimen lähdekoodin kirjastoja paljon juuri datan käsittelyä ja koneoppimista varten. Pythonista löytyy Pandas ja Numpy -kirjastot datankäsittelyä varten. Koneoppimista varten Pythonista löytyy Keras, Tensorflow ja Scikit learn -kirjastot. Kuvaajien piirtämistä varten Python tarjoaa Matplotlib-kirjaston. Python sopii hyvin koneoppimistehtävien ratkaisujen ideointiin, kehittämiseen ja testaukseen. (Bodepudi 2021)

2.4.1 Numpy

Numpy on Python kirjasto, joka tarjoaa n-ulotteisiin taulukoihin ohjelmointia ja laskentaa nopeuttavia rutiineja. Numpy tyypillisesti tuodaan Python koodiin nimellä *np.*(Numpy Dokumentaatio nda)

2.4.2 Pandas

Pandas sarja on indexoitu Numpy taulukko, jonka voi alustaa melkein millä tahansa datamallilla. Todella moni Numpy funktio ja metodi toimii myös Pandas sarjoille. Pandas taulukko on taas 2-ulotteinen labeloitu datarakenne. Pandas taulukkoa voi ajatella Python dictionarynä Pandas sarjoina. (Pandas Dokumentaatio n.d.)

2.4.3 Scikit-learn

Scikit-learn on ilmainen koneoppimiskirjasto Python-ohjelmistokielelle. Scikit-learn tarjoaa algoritmit muun muassa työssä käytettyihin koneoppimisalgoritmeihin, eli satunnaisiin metsiin, erittäin satunnaisiin puihin ja päätöspuihin. Scikit-learn tajoaa myös datan muokkaamista varten luokkia ja metodeja, jotka eivät ole sisäänrakennettuna Pandas tai Numpy -kirjastoihin. (Pal 2018)

2.4.4 Keras ja Tensorflow

Tensorflow 2 on avoimen lähdekoodin koneoppimiseen tarkoitettu alusta. Keras on korkeamman tason rajapinta Tensorflow 2:lle. Käyttämällä Kerasia suoraan Tensorflown sijasta, voidaan nopeuttaa eri mallien kokeilua ja testaamista. (Keras Dokumentaatio n.d.)

2.4.5 Matplotlib

Matplotlib on kattava kirjasto Python-ohjelmointikielelle, jolla voidaan tehdä staattisia, animoituja ja interaktiivisia visualisointeja (Matplotlib Dokumentaatio n.d.). Työssä käytetään Matplotlib-kirjastoa kaikkien kuvaajien piirtämiseen, mutta itse piirtämiseen käytettyä koodia ei käydä läpi.

2.5 Datan käsittely

Data on yleensä väärässä muodossa koneoppimismalleille ja data täytyy muokata malleille sopivaksi. Numeeriset ja kategorialliset -piirteet voidaan muokata siten, että mallit tekisivät mahdollisimman oikeellisia ennustuksia käytetyn datan perusteella. Valitsemalla oikeat piirteet ja jakamalla data useampaan eri osaan voidaan parantaa mallien tarkkuutta ja ehkäistä ylisovittumista.

2.5.1 Numeeriset piirteet

Datasetissä voi olla numeerisia piirteitä, jolla on eri erisuuruiset vaihteluvälit, ja tällöin koneoppimisalgoritmi voi luoda biaksen osaan piirteistä ja ennustus on tällöin virheellinen. Piirteet voidaan standardisoida tai normalisoida biaksen tapahtumisen estämiseksi. Standardisointi tarkoittaa, että arvot muutetaan piirteen normaalijakauman mukaisesti 0-välille, ja toimii jos, piirre seuraa Gaussin käyrää. Normalisoinnissa muutetaan piirteiden arvot skaalaamalla ne 0-1 välille, ja toimii jos piirre ei seuraa gaussin käyrää. Jokainen datasetti ei välttämättä vaadi numeeristen arvojen normalisointia tai standardisointia, mutta se on tarpeellinen tehdä, jos piirteillä on eri suuruisia vaihteluvälejä. (Lakshmanan 2019)

Datassa voi olla myös tyhjiä numeerisia arvoja, joita suurin osa koneoppimisalgoritmeista ei osaa tulkita. Tyhjät arvot voidaan korvata piirteen mediaanilla tai aritmeettisella keskiarvolla, mutta tällöin ei huomioida piirteiden välisiä riippuvuuksia ja malli voi antaa virheellisiä ennustuksia. (Kumar 2020. 7)

Rivit, joissa on tyhjiä numeerisia arvoja, voidaan myös poistaa kokonaan. Rivien poistaminen pienentää datan otoksien määrää, ja mitä pienempi määrä on otoksia, sitä hankalampaa koneoppimisalgoritmilla on löytää piirteiden välisiä yhteyksiä. (Kumar 2020. 7)

Tyhjät arvot voidaan myös ennustaa algoritmeilla, mutta se menee hankalaksi, kun on usea piirre missä on tyhjiä numeerisia arvoja.

2.5.2 Kategorialliset piirteet

Kategoriallisissa piirteissä voidaan helposti käsitellä tyhjät kentät asettamalla tyhjä kenttä yhdeksi vaihtoehdoksi piirteeseen.

Kategorialliset piirteet voidaan käsitellä usealla eri tavalla. Artikkelissa Brownlee J. suosittelee käyttämään ordinal -enkoodausta tai One-hot-enkoodausta (Brownlee 2020). Ordinal -enkoodaus muuttaa muuttujan vaihtoedot numeerisiksi taulukon 1 mukaisesti.

TAULUKKO 1. Ordinal -enkoodaus Väri -piirteelle

Indeksi	Väri	Ordinal -enkoodaus
1	Sininen	1
2	Punainen	2
3	Keltainen	3
4	Sininen	1
5	Sininen	1

Taulukossa 1 jokaiselle piirteen Väri vaihtoehdolle on annettu numero. Ordinal -enkoodaus voi huonontaa mallia tai mallista voidaan saada odottamattomia tuloksia, koska siinä malli olettaa kasvavan järjestyksen kategorialliselle piirteille

(Brownlee 2020). One-hot-enkoodaus korjaa tämän muuttamalla kaikki mahdolliset arvot binääriseksi taulukon 2 mukaisesti.

TAULUKKO 2. One-hot-enkoodaus Väri -piirteelle

	Väri	Väri - sininen	Väri - punainen	Väri - keltainen
1	Sininen	1	0	0
2	Punainen	0	1	0
3	Keltainen	0	0	1
4	Sininen	1	0	0
5	Sininen	1	0	0

Taulukossa 2 One-hot-enkoodaus on lisännyt taulukkoon 3 lisää piirrettä, jotka selittävät Väri -piirteen arvot yhdellä ja nollalla. One-hot-enkoodaus lisää piirteiden määrää ja jakaa piirteen merkityksen useaan eri piirteeseen, mutta One-hot-enkoodauksen pitäisi johtaa mallia harhaan vähemmän kuin ordinal enkoodauksen (Brownlee 2020).

2.5.3 Piirteiden valinta

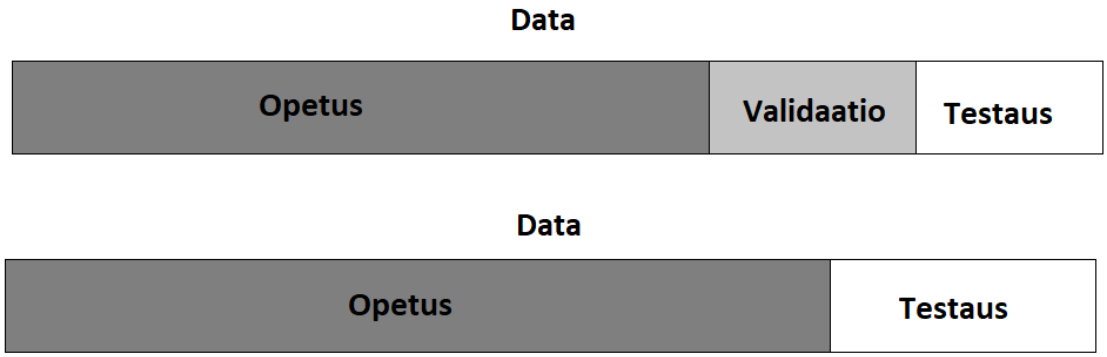
Datassa voi olla paljon piirteitä, joista osa voi olla täysin turhia ja mallille harhaanjohtavia. Turhien piirteiden poisto pienentää ylisovittumista, parantaa mallin tarkkuutta ja pienentää opetukseen kulunutta aikaa. Merkitsevimmät piirteet voidaan valita eri tavoin, joista suosituimmat algoritmit ovat SelectKBest ja puihin perustuvat luokittelijat. (Shaikh 2018)

2.5.4 Datan jako eri osuuksiin

Data jaetaan yleensä vain opetus ja testaus datasetteihin. Opetus datasetillä opetetaan koneoppimismalli ja testaus datasetillä testataan mallin tarkkuus, jotta saadaan tarkkuus datalle, jota malli ei ole ikinä nähnyt. Data voidaan jakaa vielä validaatio datasettiin, jolla voidaan testata mallia jo opetusvaiheessa. Validaatiosetin käytöllä voidaan estää ylisovittumista, koska nähdään jo opetusvaiheessa,

jos opetusdataa vasten tarkkuus paranee, mutta validaatiodataa vasten tarkkuus heikkenee. (Brownlee 2017)

Kuviossa 7 on visualisoitu datan jakaminen eri datasetteihin.



KUVIO 7. Datan jako opetus, validaatio ja testaus datasetteihin

Kuten kuviosta 7 nähdään, jos data jaetaan myös validaatio datasettiin, niin opetukselle ja testaukselle jää vähemmän otoksia. Mitä vähemmän otoksia on opetukselle ja testaukselle, sitä suurempi mahdollisuus on, että malli ei löydä kaikkia yhteyksiä piirteiden välillä.

2.6 Metriikat

Mallien hyvyys pitää pystyä määrittämään jotenkin. Tämän takia on kehitetty erilaisia metriikoita, joilla mallin hyvyys pystytään määrittämään. Keskitytään kuitenkin ainoastaan luokitteluun tarkoitettuihin metriikoihin.

2.6.1 Confusion -matriisi

Confusion -matriisilla voidaan helposti selittää luokittelijan kokonaissuoriutuminen. Confusion -matriisi kahdelle luokalle, positiivinen ja negatiivinen, on esitetty taulukossa 3. (Mishra A. 2018)

Taulukko 3. Confusion -matriisi

Oikeat		Ennustetut	
	Positiiviset	Oikeasti positiiviset (TP)	Väärin negatiiviset (FN)
	Negatiiviset	Väärin positiiviset (FP)	Oikeasti negatiiviset (TN)
		Positiiviset	Negatiiviset

Taulukossa 3 on sarakkeilla oikeat arvot ja riveillä ennustetut arvot. Oikeasti positiiviset (TP) on mallin ennustamia positiivisia, jotka ovat oikeasti myös positiivisia. Väärin positiiviset (FP) on mallin ennustamia positiivisia, jotka ovat oikeasti negatiivisia. Väärin negatiiviset (FN) on mallin ennustamia negatiivisia, jotka ovat oikeasti positiivisia. Oikeasti negatiiviset (TN) on mallin ennustamia negatiivisia, jotka ovat oikeasti negatiivisia. Taulukossa 3 vihreä väri kuvaa oikein mennyttä ennustusta ja punainen väärin mennyttä ennustusta.

2.6.2 Tarkkuus (Accuracy)

Tarkkuus on suhde oikein menneiden ennustusten ja kaikkien ennustusten välillä. Taulukon 3 termeillä kaava menee seuraavasti.

$$Tarkkuus = \frac{TP + TN}{TP + TN + FP + FN}$$

Tarkkuus metriikkana on siitä huono, että se ei toimi luokittelumallissa, jossa yhtä luokkaa on huomattavasti enemmän kuin toista. Ajatellaan, että koko datasetistä

98 % on luokkaa A ja 2 % luokkaa B, niin malli voi suoraan ennustaa kaikki luokkaan A ja mallin tarkkuus on tällöin 98 %. Ja kun A luokan määrää pienennetään 60 % ja B luokan määrää nostetaan 40 %, niin mallin tarkkuus on vain 60 %. (Mishra 2018)

2.6.3 Oikeasti positiivisten suhde (TPR), oikeasti negatiivisten suhde (TNR) ja väärien positiivisten suhde (FPR)

Oikeasti positiivisten suhde (true positive rate, TPR), tai puhutaan myös sensitivity-arvosta tai recall-arvosta, vastaa oikeiden positiivisten ennustuksien suhdetta kaikkiin oikeasti positiivisiin datapisteisiin. (Mishra 2018)

Recall-arvo kertoo, kuinka suuri osa oikeasti positiivista ennustuksista ennustettiin oikein. Taulukon 3 termeillä recall-arvo lasketaan seuraavasti.

$$TPR = Recall = \frac{TP}{FN + TP}$$

Oikeasti negatiivisten suhde (true negative rate, TNR), tai puhutaan myös specifity-arvosta, vastaa oikeiden negatiivisten ennustuksien suhdetta kaikkiin oikeasti negatiivisiin datapisteisiin (Mishra 2018). Taulukon 3 termeillä TNR lasketaan seuraavasti.

$$TNR = \frac{TN}{FP + TN}$$

Oikeasti negatiivisten suhteella *TPR* ja oikeasti positiivisten suhteella *TNR* saadaan parempi käsitys mallin hyvyydestä kuin pelkällä tarkkuudella. Jos malli yrittää ennustaa kaikki datapisteet samaan luokkaan epätasapainoisessa datassa, toinen näistä metriikoista on 0 ja toinen 1 ja se huomataan heti.

Oikeasti positiivisten suhde *TNR* voidaan kääntää väärien positiivisten suhteeseen *FPR* seuraavalla kaavalla.

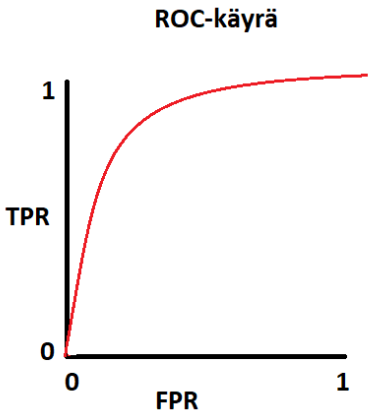
$$FPR = 1 - TNR$$

Väärien negatiivisten suhdetta käytetään ROC-käyrän arvojen laskemiseen.

2.6.4 ROC-käyrän pinta-ala

ROC-käyrä (receiver operating characteristic curve) on todennäköisyyskäyrä, joka kertoo mallin kyvystä erottaa oikeat positiiviset oikeista negatiivisista. ROC-käyrä on oikeasti positiivisten suhde (TPR) väärien positiivisten suhteen funktiona (FPR). Koska testauksen jälkeen *TPR* ja *FPR* -arvot ovat vain yhdet luvut, pitää FPR paloitella tietyn mittaisiin kynnyksiin, jolloin voidaan laskea TPR-arvo tietyllä FPR kynnyksellä. (Narkhede 2018).

Esimerkki ROC-käyrästä on kuviossa 8.



KUVIO 8. ROC-käyrä

Kuviosta 8 punaisella on piirretty ROC-käyrä, x-akselilla on eri *FPR*-arvot ja y-akselilla *TPR*-arvot. Paras mahdollinen ROC-käyrä saataisiin, jos jokaisella *FPR*-arvolla TPR-arvo olisi 1 (Narkhede 2018).

Aluetta ROC-käyrän ja x-akselin välissä alueella [0,0] ja [1,1] kutsutaan ROC-käyrän pinta-alaksi. Pinta-ala voidaan laskea muillekin käyrille, kuin pelkästään ROC-käyrälle, mutta ROC-käyrä on yleisin. ROC-käyrän pinta-ala kertoo, kuinka hyvin malli pystyy erottamaan dataotokset luokkien välillä. Mitä korkeampi käyrän alueen arvo, eli AUC-arvo on, sitä parempi malli on ennustamaan taulukon 3 luokilla positiiviset positiivisiksi ja negatiiviset negatiivisiksi. (Narkhede 2018)

2.6.5 F1-arvo

F1-arvo on harmoninen aritmeettinen keskiarvo precision-arvon ja recall-arvon välillä. Recall-arvo on sama kuin aiemmin käyty *TPR*-arvo, ja precision-arvo on oikein ennustettujen positiivisten suhde kaikkien ennustettujen positiivisten välillä. (Mishra 2018)

Precision-arvo on suhde, joka kertoo kuinka suuri osa positiivisista ennustuksista, on oikein arvattu. Kaava precision-arvolle on seuraava.

$$Precision = \frac{TP}{TP + FP}$$

F1-arvolle kaava on seuraava.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

Yleensä precision-arvon kasvaessa recall-arvo pienenee ja recall-arvon kasvaessa precision-arvo pienenee (Google, Machine learning course. N.d). Tämän takia näiden yhdistelmä eli F1-arvo on kätevä, koska ei tarvitse kuin seurata yhtä arvoa.

2.6.6 Virheet

Virheen voi jakaa kolmeen eri osaan, joita ovat varianssi, bias ja vähentymätön -virheet. Vähentymätöntä virhettä ei voida pienentää mitenkään riippumatta käytetystä algoritmista. Vähentymätön virhe syntyy suoraan ongelman asettelusta, esimerkiksi käytetyistä piirteistä. (Brownlee 2016)

Bias syntyy, kun tehdään yksinkertaisia oletuksia käytetystä datasta käytetyllä algoritmilla, jotta malli olisi nopea opettaa. Yleensä lineaarisilla algoritmeilla on

suuri bias, jolloin ne ovat todella nopea opettaa, mutta heti kun tulee hankalampi ongelma vastaan, bias virhe on suuri ja tarkkuus on tällöin todella huono. Mitä pienempi bias virhe on, sitä vähemmän tehdään oletuksia käytettyyn algoritmiin. (Brownlee 2016)

Varianssi kertoo, kuinka paljon opetettu algoritmi muuttuu, jos eri opetus dataa käytetään. Ideaalissa tapauksessa algoritmi ei muutu kuin vähän, vaikka eri dataa käytettäisiinkin, tarkoittaen, että algoritmi osaa kartoittaa piirteet oikeisiin lähtöihin. Ei-lineaarisilla koneoppimisalgoritmeilla on yleensä suuri varianssi, koska ne ovat todella joustavia. Mitä pienempi varianssi virhe on, sitä vähemmän on väliä mitä osaa datasta käytetään opetusdatana ja tällöin algoritmin on helppo tehdä oikea kartoitus tulo- ja lähtöpiirteiden välillä. (Brownlee 2016)

Mallin bias-virhettä voidaan laskea tekemällä mallista monimutkaisempi, mutta samalla varianssivirhe huonontuu ja sama jos varianssivirhettä yritetään laskea, varianssivirhe nousee. On olemassa kuitenkin kohta missä varianssivirhe ja bias-virhe ovat pienimmillään ja se on kohta mikä halutaan löytää koneoppimismallilla, jotta kokonaisvirhe saadaan minimoitua.

Bias ja varianssi -virheet lasketaan, neliöllisestä häviöstä tai 0-1 välille skaalatusta häviöstä (Bias-Variance Decomposition n.d.).

3 Toteutus

Työssä tehdään malli, joka ennustaa tuleeko myyntimahdollisuus olemaan voitettu tai hävitty. Tästä on hyötyä Ambientia Oy:lle, sillä mallia voidaan hyödyntää myyntimahdollisuuksissa, jotka eivät ole vielä päättyneet.

Mallin kouluttamisessa on käytetty lähdedatana työn toimeksiantajan liikesalaisuuden alaisia tietoja, joita ei siksi esitetä osana tätä työtä.

Kuten teoriassa käytiin läpi, on olemassa regressio tai luokittelu ongelmia. Tämä ongelma on luokittelu ongelma, koska myyntimahdollisuudelle ennustetaan sen lopputulos kahdesta luokasta, voittaminen tai häviäminen.

Teoriassa käytiin läpi eri koneoppimismalleja ja kokeillaan niistä jokaista ongelman ratkaisuun. Mallien tarkkuutta ja muita metriikoita vertaillaan vielä lopussa, jotta voidaan valita käytetyistä malleista paras vaihtoehto. Mallit mitataan kappaleen 2.7 metriikoilla.

3.1 Aloitus

Myyntimahdollisuustietomalli yhdistetään yritystietomallin kanssa, jolloin saadaan paljon enemmän piirteitä ja näin suurempi todennäköisyys, että dataan tulee enemmän piirteitä, jotka koneoppimismalli osaa kartoittaa myyntimahdollisuuden voittamisen tai häviämisen kanssa.

Työ tehdään Python-ohjelmointikielellä ja Pythoniin koneoppimiskirjastona käytetään Scikit-learn-kirjastoa ja neuroverkkoja varten Kerasia, jossa on Tensorflown backend.

Kaikissa malleissa tarvittavat Python-kirjastot ovat tuotu Python-skriptiin kuvan 2 koodissa.

```
1  import pandas as pd
2  import numpy as np
```

KUVA 2. Työn kaikissa vaiheissa tarvittavat kirjastot

Kuvassa 2 on kirjastot Pandas ja Numpy, joille annetaan aliakset *pd* ja *np*, jotta koodia voidaan kirjoittaa nopeammin ja se on myös yleinen käytäntö.

3.2 Datan muokkaus

Toteutus alkaa sillä, että haetaan lähdedata Pandas-taulukkoon (Dataframeen). Lähdedatassa on yhdistetty yritys- ja myyntimahdollisuustietomallit. Kuvan 3 koodissa ladataan Pandas-taulukkoon yhdistetyt datat, käyttämällä itsetehtyä funktiota *get_combined_data_df*, jota ei esitellä tässä .

```
1  df = get_combined_data_df()
```

Kuva 3. Datan lataaminen Pandas-taulukkoon

Kuvassa 3 alustettu muuttuja *df* sisältää käytetyn datan Pandas-taulukkona.

3.2.1 Tyhjien kolumnien poisto

Lopullinen malli tulee ennustamaan myyntimahdollisuuksia, joiden lopputulosta ei vielä tiedetä. Ei olisi siis järkevää pitää Pandas-taulukossa piirteitä, joilla on tyhjiä arvoja jokaisessa piirteessä. Tämän takia poistetaan heti alussa piirteet, joilla on uusissa myyntimahdollisuuksissa enemmän kuin puolet kaikista otoksista tyhjiä arvoja.

Kuvan 4 koodissa suoritetaan uusien myyntimahdollisuuksien tyhjien piirteiden poisto koko datasta.

```
1  # drop columns that have 50 % or more null values in a column of the first stage of the deals
2  new_deals_df = df[df['display_order_pipeline_stage'] == 0]
3
4  non_nan_value_columns_in_new_deals = new_deals_df.dropna(axis=1, how='all',
5      thresh=new_deals_df.shape[0]/2).columns.values.tolist()
6
7  df = df[non_nan_value_columns_in_new_deals]
```

KUVA 4. Uusien myyntimahdollisuuksien tyhjien kolumnien poisto

Kuvassa 4 riveillä 4 ja 5 käytettävä Pandas-taulukon metodi *dropna* poistaa otoksia tai kolumneja taulukosta. *axis*=1 -argumentti tarkoittaa, että halutaan poistaa kolumneja eikä rivejä ja *how='all'* -argumentti valitsee vain sellaiset kolumnit jossa kaikki arvot ovat tyhjiä. *thresh=df.shape*[0]/2 -argumentilla valitaan vain sellaiset kolumnit, joissa kaikista kolumnin arvoista yli puolet on tyhjiä. Eli lopuksi saadaan poistettua sellaiset kolumnit, joilla on uusissa myyntimahdollisuuksissa yli puolet kaikista otoksista tyhjiä arvoja. Lopuksi asetetaan *df*-muuttujaan ainoastaan ne kolumnit, jotka täyttävät edellä mainitut ehdot.

3.2.2 Ennustettavan piirteen muokkaaminen

Ennustettavassa kolumnissa ”dealstage_deals” on useampi eri vaihtoehto, kuin pelkästään tieto siitä onko myyntimahdollisuus voitettu vai ei. Tämän takia poistetaan opettamista varten taulukosta turhat vaihtoehdot. Lisätään vielä ”target” -kolumni, johon asetetaan 1 jos myyntimahdollisuus on voitettu ja 0 jos myyntimahdollisuus on hävitty, eli muutetaan kolumni binääriseksi. Kuvan 5 koodissa muutetaan ennustettava piirre binääriseksi.

```
1  label = "dealstage_deals"
2  wanted_dealstages = ["closedlost", "closedwon"]
3
4  df_not_finished = df[~df[label].isin(wanted_dealstages)]
5  df = df[df[label].isin(wanted_dealstages)]
```

KUVA 5. Ennustettavan kolumnin muuttaminen binäärikseksi kolumniksi

Rivillä 4 Pandas-taulukon *isin* -metodi testaa jokaista taulukon elementtiä ja jos elementti kuuluu *isin* -metodin ensimmäiseen argumenttiin, niin palautetaan tosi, ja epätosi muussa tapauksessa. Rivillä 4 siis jätetään Pandas-taulukkoon *df* vain sellaiset rivit, jotka ovat voitettuja eli ”closedwon” tai hävittyjä eli ”closedlost”

myyntimahdollisuuksia. Tämä tehdään, koska mallien opettamista varten myyntimahdollisuuksien pitää olla päättyneitä voittoon tai häviämiseen.

Kuvan 5 koodin rivillä 5 esiintyvä *np.where* -metodi palauttaa taulukon, jossa sen ensimmäisen argumentin taulukko on korvattu toisen ja kolmannen argumentin arvoilla. Ensimmäiseksi argumentiksi annetaan yhden kolumnin taulukko sisältäen tosi ja epätosi -arvoja. Rivillä 5 yhden kolumnin taulukko on ennustettava piirre, joka on tosi vain silloin, jos arvo on ”closedlost”. Toinen argumentti on arvo, joka asetetaan, jos taulukon elementti on tosi eli koodissa 0. Kolmas argumentti on arvo, joka asetetaan, jos taulukon elementti on epätosi, eli koodissa 1.

3.2.3 Numeeristen ja kategoriallisten piirteiden käsittely

Työssä käytetään vain numeerisia ja kategoriallisia kolumneja/piirteitä ja ne jaetaan koodissakin näihin, mutta näiden haluttujen kolumnien määrittelyä ei esitellä koodissa. Numeeristen kolumnien tyhjistä arvoista voidaan huolehtia teoria kappaleen mukaisesti parilla eri tavalla. Valitaan kuitenkin työhön arvojen täyttäminen kolumnin aritmeettisella keskiarvolla, koska tämä on helppo toteuttaa. Kuvan 6 koodissa täytetään numeeristen kolumnien tyhjät kentät aritmeettisella keskiarvolla.

```
7  for column in numeric_columns:
8      df[column] = df[column].fillna(df.mean()[column])
```

KUVA 6. Numeeristen kolumnien tyhjien arvojen täyttö kolumnin keskiarvolla

Kuvan 6 koodin *numeric_columns* -muuttuja sisältää listan numeerisista kolumneista. Pandas-taulukon *fillna*-metodi täyttää tyhjät kentät sille annetulla arvolla. Arvo, jolla kolumnin tyhjät arvot täytetään, otetaan Pandas-taulukon *mean*-metodista, joka palauttaa jokaisen taulukon kolumnin aritmeettisen keskiarvon. Kuvan 7 koodissa vielä normalisoidaan numeeriset kolumnit 0-1 välille.

```
from sklearn import preprocessing
#normalize numeric columns, neural networks like that
numeric_column_values = df[numeric_columns].values

min_max_scaler = preprocessing.MinMaxScaler()
numeric_column_values_scaled = min_max_scaler.fit_transform(numeric_column_values)
df[numeric_columns] = numeric_column_values_scaled
```

KUVA 7. Numeeristen kolumnien normalisointi

Sklearn:in prepocessing -kirjaston *MinMaxScaler* -luokalla voidaan helposti normalisoida Numpy-taulukoiden muodossa oleva data. Pandas-taulukon muuttaminen Numpy-talukoksi saadaan kuvan 7 mukaisesti Pandas-taulukon attribuutista *values*. *MinMaxScaler* -luokan metodi *fit_transform* ottaa argumentiksi Numpy-taulukon ja muuttaa siinä jokaisen kolumnin arvot normalisoiduiksi arvoiksi kolumni kerrallaan.

Kategorialliset kolumnit pilkotaan uusiin kolumneihin teorian mukaisesti One-hot-enkoodauksella, sillä muutoksella, että alkuperäinen kolumni poistetaan vielä taulukosta. Pandas-kirjasto tarjoaa tähän suoraan *get_dummies*-metodin ja sitä käytetään kuvan 8 koodissa.

```
df = pd.get_dummies(df, columns = categorical_columns) # one hot encodes columns in categorical columns
```

KUVA 8. Kategoriallisten kolumnien vektorisointi One-hot-enkoodauksella

Kuvan 8 koodissa Pandas-kirjaston *get_dummies* -metodin ensimmäinen argumentti on Pandas-taulukko, jolle halutaan tehdä One-hot-enkoodaus. Koodissa *categorical_columns*-muuttuja on lista, joka sisältää kolumnien nimet, jotka ovat kategoriallisia kolumneja. Asettamalla *columns=categorical_columns* saadaan *get_*dummies -metodi tekemään One-hot-enkoodaus vain kategorisille kolumneille.

3.2.4 Datan jakaminen

Data pitää pystyä jakamaan opetukseen, testaukseen ja vapaaehtoisesti myös validointiin. Funktio, joka suorittaa tämän määritetään kuvan 9 koodissa.

```
def split_dataset(df, ratio):
    random_arr_indices = np.random.rand(len(df)) < ratio # Contains true and false values and they have (ratio * 100 %) chance of being true
    train = df[~random_arr_indices]
    test = df[random_arr_indices]
    return train, test

```

KUVA 9. Pandas-taulukon pilkkominen datasetteihin

Kuvassa 9 esiintyvä *numpy.random.rand*-metodi palauttaa Numpy-taulukon, joka sisältää sille annetun argumentin verran satunnaisia arvoja välillä 0 ja 1. Kun Numpy-taulukolle suorittaa vertailua, vertailu tehdään jokaiselle taulukon alkiolle erikseen, eli *random_arr* -muuttuja sisältää tosi ja epätosi -arvoja *ratio* -parametrin suhteella vertailun jälkeen. Tilde komplementoi kaikki Numpy-taulukon arvot. Näin saadaan opetus osuudesta suurempi, kuin testi osuudesta.

Kaikki käytetyt mallit tarvitsevat datan Numpy-taulukkona jaettuna tulo- ja lähtöpiirteisiin. Määritetään funktio kuvan 10 koodissa, joka palauttaa Numpy-taulukot riippuen siitä, minkä piirteen sille antaa *label*-parametrina.

```
def get_x_y_value_split(df, label):
    return df.drop(label, axis=1).values, df[label].values
```

KUVA 10. Pandas-taulukon jakaminen tulo ja lähtö taulukoihin

Pandas-taulukon metodilla *drop* voidaan tiputtaa kolumni antamalla sille argumentiksi kolumnin nimi ja asettamalla *axis* -argumentin arvoksi 1. Labeliksi annetaan funktiolle sitä kutsuttaessa ”target”, koska se on asetettu ennustettavan piirteen nimeksi kuvan 5 koodissa.

3.3 Neuroverkon ohjemointi

Neuroverkoissa tarvittavat kirjastot on tuotu Python-skriptiin kuvan 11 koodissa.

```
import tensorflow as tf
from tensorflow.keras import layers
```

KUVA 11. Neuroverkoissa käytetyt kirjastot

Kuten kuvasta 11 nähdään, *layers* luokka tuodaan erikseen, vaikka Tensorflow on jo tuotu. Tämä lisää koodin luettavuutta.

Neuroverkkoja varten ohjelmakoodissa data jaetaan opettamiseen, testaukseen ja validointiin. Jako tehdään käyttämällä kuvassa 9 määritettyä *split_dataset*-funktiota. Funktiota käytetään kuvan 12 koodissa jakamaan data opetukseen, testaukseen ja validointi datasetteihin.

```
1  train, test = split_dataset(df, 0.2)
2  train, val = split_dataset(train, 0.2)
```

KUVA 12. Datasetin jako opetus, validointi ja testi osuuksiin.

Kuten kuvasta 12 nähdään, datasetti jaetaan kahdesti 20 % osuuksiin, eli testi ja validointi -datasetit ovat lähes yhtä suuret.

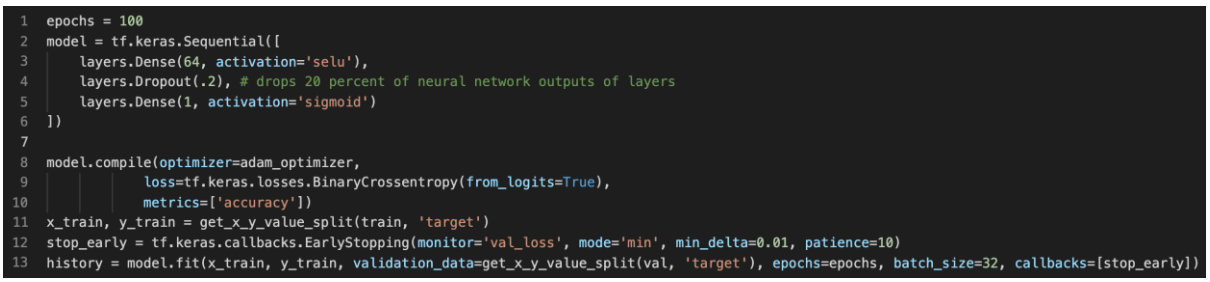
Työssä käytetyt optimointialgoritmit on alustettu kuvan 13 koodissa.

```
1  #optimizers
2  stochastic_gradient_descent_optimizer = tf.keras.optimizers.SGD(learning_rate=1e-2, momentum=0.9)
3  adam_optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-4)
```

KUVA 13. Työssä käytetyt optimointialgoritmit

Kuvassa 13 on alustettu SGD ja Adam -optimointialgoritmit. Jos halutaan SGD-optimointialgoritmin oppivan sopivalla tahdilla, on 0,01 hyvä valinta opetustahdiksi ja Adamille taas 0,0001. Teoriassa käydyssä kappaleessa neuroverkkojen momentum -hyperparametriksi päädyttiin SGD-optimointialgoritmille valitsemaan asiantuntijoiden suosittelema kuvan 13 mukainen arvo 0,9.

Itse neuroverkkomallin määrittely ja sen opettaminen on kuvan 14 koodissa.



```
1  epochs = 100
2  model = tf.keras.Sequential([
3      layers.Dense(64, activation='selu'),
4      layers.Dropout(.2), # drops 20 percent of neural network outputs of layers
5      layers.Dense(1, activation='sigmoid')
6  ])
7
8  model.compile(optimizer=adam_optimizer,
9                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
10               metrics=['accuracy'])
11 x_train, y_train = get_x_y_value_split(train, 'target')
12 stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', min_delta=0.01, patience=10)
13 history = model.fit(x_train, y_train, validation_data=get_x_y_value_split(val, 'target'), epochs=epochs, batch_size=32, callbacks=[stop_early])
```

KUVA 14. Neuroverkkomallin määrittely ja opetus

Normaalin neuroverkon, jossa tasot ovat peräkkäin, saa tehtyä kerasissa *Sequential* -luokalla. Kuvassa 14 rivillä 3 valitaan piilossa olevien tasojen aktivaatiofunktioksi SELU, koska se oli todettu parhaimmaksi teoriassa. Viimeisen tason aktivaatio funktioksi valitaan rivillä 5 Sigmoid, koska se on paras binääriseen luokitteluun. Tiputetaan 20 % neuroneista rivillä 4 vielä ennen viimeistä tasoa, joka ehkäisee mallin ylisovittumista.

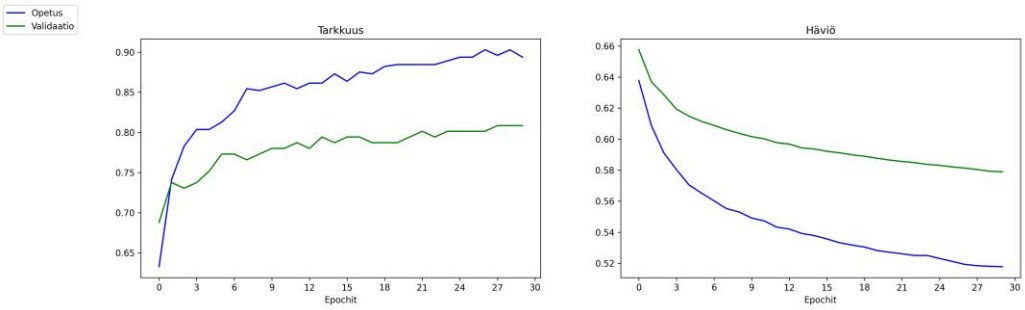
Compile-vaiheessa riveillä 8-10 kuvan 14 koodissa valitaan optimointialgoritmiksi Adam, häviöfunktioksi BinaryCrossentropy ja metriikaksi vain perinteinen tarkkuus. *BinaryCrossentropy*-luokan argumentti *from_logits* muuttaa Binarycrossentropy -häviöfunktion ulostulon 0 ja 1 väliltä palauttamaan negatiivisesta ääretömästä äärettömään.

Kuvan 14 koodissa rivillä 10 saadaan opetus ja validaatio -vaiheiden tarkkuuden ja häviön arvot jokaisella epochilla tallennettua argumentilla *metrics=[’accuracy’]*.

Kuvan 14 koodissa rivillä 12 määritellään *stop_early*-muuttuja, jota käytetään takaisinkutsuna opettamisen lopettamiseen tietyllä epochilla, joka täyttää tietyt ehdot. *tf.keras.callbacks.EarlyStopping*-luokalla saadaan tehtyä tämä takaisinkutsu, joka lopettaa mallin opettamisen kun joku ehto täyttyy. Ehto määritellään luokan rakentajan parametreillä. 12 rivillä määritellään *monitor=’val_loss’* -argumentilla *early_stopping*-objekti seuraamaan häviötä validaatiodataa vasten. *mode=’min’*, *min_delta*=0.01 ja *patience*=10 -argumentit yhdessä asettavat itse ehdon opettamisen lopettamiselle, eli kun 10 epochia on kulunut ja validaatiodatan häviö ei ole tippunut 0.01 verran, lopetetaan opettaminen. Aikaisin opettamisen lopettamisen pitäisi myös estää ylisovittumista, koska malli ei tällöin kerkeä mitenkään oppimaan opetusdataa ulkoa.

Kuvan 14 rivillä 13 itse malli opetetaan ja ensimmäiset 2 argumenttia *fit* -metodiin ovat kaikki malliin tulopiirteiden arvot eli 11 rivillä jaettu *x_train* -muuttuja ja ennustettavan piirteen arvot eli *y_train*-muuttuja. Vapaaehtoinen parametri *validation_data* on kahden pituinen lista, jossa ensimmäinen alkio sisältää tulopiirteet ja toinen alkio sisältää ennustettavan piirteen. Kaikki nämä jaot tehdään kuvan 10 koodissa määritellyllä funktiolla. *epochs=epochs*-argumentti asettaa epochien lukumääräksi 100, mutta epochien lukumäärällähän ei ole oikeastaan väliä mallissa, koska *stop_early*-takaisinkutsu lopettaa opettamisen, kun validaatiodataa vasten häviö ei pienene. *batch_size=32* eli erän kooksi asetetaan 32, jota suositeltiinkin kokeilemaan teoriassa. Lopuksi asetetaan vielä *callbacks* parametriin argumentiksi rivillä 12 määritelty *stop_early* -objekti, eli opettaminen lopetetaan, epochilla kun ehto täyttyy.

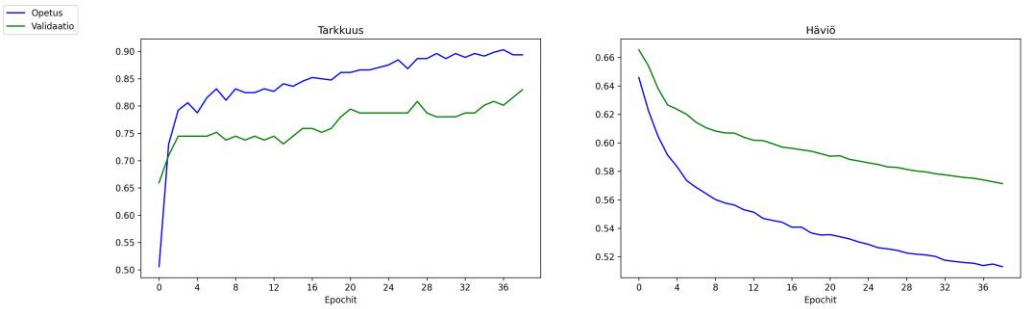
Opetetaan vielä kuvassa 14 määritelty malli molemmilla optimointialgoritmeilla. Kuviossa 9 on käytetty Adam -optimointialgoritmia mallin opettamisessa.



KUVIO 9. Neuroverkon tarkkuus ja häviö Adam -optimointialgoritmilla

Kuviosta 9 näkee, että vasemmalla on mallin tarkkuus ja oikealla on häviö. Molempiin kaavioihin on piirretty mallin opetusvaiheen käyrät, opetus sinisellä ja validointi vihreällä.

Kuviossa 10 on käytetty SGD-optimointialgoritmia mallin opettamisessa.



KUVIO 10. Neuroverkon tarkkuus ja häviö SGD-optimointialgoritmilla

Kuviosta 10 on esitettynä vasemmalla on mallin tarkkuus ja oikealla on häviö. Molempiin kaavioihin on piirretty mallin opetusvaiheen käyrät, opetus sinisellä ja validointi vihreällä.

Ajetaan sovitetut mallit testi datan läpi, jotta saadaan mallien tarkkuus testidataa vasten. Testidatan tarkkuuden tulostamiseen tehty koodi on kuvan 15 koodissa.

```
1  x_test, y_test = get_x_y_value_split(test, 'target')
2  loss, acc = model.evaluate(x_test, y_test, verbose=0)
3  print(f"Accuracy {acc}")
```

KUVA 15. Tarkkuuden tulostus testidataa vasten

Kuvassa 15 toisella rivillä annetaan mallin *evaluate* -metodille kahdessa ensimmäisessä argumentissa mallille tulopiirteiden arvot *x_test* -taulukkona ja ennustettavien piirteiden arvot *y_test* -taulukkona. Kuvan 14 koodissa rivillä 10 annetaan metriikaksi ”accuracy”, jolloin *evaluate* -metodi palauttaa vain mallin tarkkuuden ja häviön testidataa vasten. Metodi ensin ennustaa *x_test* -taulukolle ennustettavan piirteen arvot ja vertaa niitä *y_test* -taulukon arvoihin, joka palauttaa mallin tarkkuuden ja häviön.

Mallin tarkkuudeksi testidataa vasten saadaan Adam -optimointialgoritmilla

$$Tarkkuus_{Adam} = 0{,}774193525$$

Ja mallin tarkkuudeksi saadaan SGD-optimointialgoritmilla

$$Tarkkuus_{SGD} = 0{,}7649769783$$

Yhden kerran ajamisen jälkeen näyttäisi siltä, että Adam -optimointialgoritmi on tarkempi kuin SGD-optimointialgoritmi.

3.4 Päätöspuihin perustuvat koneoppimisalgoritmit

Päätöspuihin perustuviin koneoppimisalgoritmeihin tarvittavat kirjastot on tuotu Python-skriptiin kuvan 16 koodissa.

```
1  from sklearn import metrics
2  from sklearn.tree import DecisionTreeClassifier
3  from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
```

KUVA 16. Päätöspuihin perustuvien koneoppimisalgoritmien kirjastojen tuonti

Päätöspuihin pohjautuvissa koneoppimisalgoritmeissa käytetään vaan opetus ja testi -taulukoita. Käytetään kuvan 9 koodissa määritettyä funktiota datan jakamiseen *train* ja *test* -osuuksiin ja kuvan 10 koodissa määritettyä funktiota *train* ja *test* -Pandas-taulukoiden jakamiseen tulo ja lähtö -Numpy-taulukoihin. Jako tapahtuu kuvan 18 koodissa.

```
1  train, test = split_dataset(df, .3)
2  X_train, y_train = get_x_y_value_split(train, label)
3  X_test, y_test = get_x_y_value_split(test, label)
```

KUVA 18. Taulukoiden jakaminen opetus ja testi -taulukoihin ja niiden muuttaminen Numpy-taulukoiksi

Kuvan 18 ensimmäisellä rivillä jaetaan data opetusta varten *train* -taulukkoon ja testausta varten *test* -taulukkoon 30 % suhteella, koska puihin perustuvat koneoppimisalgoritmit eivät pysty hyödyntämään validointi dataa, kuten neuroverkot pystyvät. *X_train* -taulukko sisältää piirteiden arvot, joilla yritetään ennustaa *y_train* -taulukon arvoja mallin opettamisessa. Testauksessa käytetään *X_test* ja *y_test* -taulukoita.

3.4.1 Päätöspuun ohjelmointi

Päätöspuun mallin luonti, opettaminen ja testi datan ”target” -piirteen ennustus tapahtuu kuvan 19 koodissa.

```
3  clf = DecisionTreeClassifier(max_depth=5)
4
5  # Train Decision Tree Classifer
6  clf = clf.fit(X_train,y_train)
7
8  #Predict the response for test dataset
9  y_pred = clf.predict(X_test)
10
11 print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
12
```

KUVA 19. Päätöspuun luonti, opettaminen ja testi datan ennustaminen

Kuvassa 19 kolmannella rivillä argumentti *max_depth=5* asettaa päätöspuun maksimisyvyyden viiteen. 6 rivillä opetetaan malli ja 9 rivillä ennustetaan testi datasetin ”target” -piirteen arvot. Rivillä 11 käytetään Scikit-learn-kirjaston *metrics* -luokan *accuracy_score* -metodia, jolle annetaan argumentteina kaksi taulukkoa, joista metodi laskee tarkkuuden. Rivillä 11 annetaan ennustetut arvot ja oikeat arvot, joiden pohjalta *accuracy_score* -metodi laskee mallille tarkkuuden.

Kuvan 19 koodi antaa päätöspuulle tarkkuudeksi

$$Tarkkuus_{DT} = 0{,}8186813.$$

3.4.2 Satunnaisen metsän ohjelmonti

Satunnaisen metsän mallin luonti, opettaminen ja testidatan ’target’ -piirteen ennustus tapahtuu kuvan 20 koodissa.

```
clf = RandomForestClassifier(n_estimators=100, max_depth=None)
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

```

KUVA 20. Satunnaisen metsän mallin luonti, opettaminen ja testi datan ennustus

Kuvan 20 koodissa kolmannella rivillä asetetaan argumentin *n_estimators=100*, joka asettaa asettaa päätöspuiden määrän, jotka suorittavat äänestämisen todennäköisimmästä lähtöluokasta. Kolmannella rivillä myös asetetaan mallin *max_depth=None* -argumentti asettaa puiden maksimi syvyyden. *None* tarkoittaa, että puilla ei ole maksimi syvyyttä.

Koodi tulostaa satunnaisen metsän tarkkuudeksi seuraavan arvon.

$$Tarkkuus_{RF} = 0{,}7916666$$

3.4.3 Erittäin satunnaisten puiden ohjelmointi

Erittäin satunnaisten puiden mallin luonti, opettaminen ja testi datan ennustus tapahtuu kuvan 21 koodissa.

```
clf = ExtraTreesClassifier(n_estimators=100, max_depth=None, min_samples_split=2)
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

```

KUVA 21. Erittäin satunnaisten puiden mallin luonti, opettaminen ja testi datan ennustus

Argumentit *n_estimators*=100 asettaa päätöspuiden määrän ja *max_depth=None* asettaa puiden maksimi syvyyden äärettömäksi ja *min_samples_split=2* asettaa päätöspuulle solmun jakamisen minimi määrän tarvittavia otoksia, jotta solmu voidaan luoda.

Koodi tulostaa erittäin satunnaisille puille tarkkuuden

$$Tarkkuus_{RF} = 0{,}77604166.$$

3.5 Optimoidut mallit

Optimointikoodia ei esitellä sen tarkemmin, mutta optimoinnissa hyödynnetään Scikit-learn -kirjaston *model_selection.RandomizedSearchCV* ja *model_selection.GridSearchCV* -luokkia.

3.5.1 Päätöspuun hyperparametrien optimointi

Päätöspuulle yritetään optimoida kolmea eri hyperparametria. *min_samples_split* -hyperparametri määrää päätöspuun solmuille minimimäärän otoksia, jotta solmu voidaan jakaa. *min_samples_leaf* -hyperparametri määrää päätöspuun lehtisolmuille minimäärän otoksia, jotta siihen tehty jakaminen ylemmältä tasolta on laillinen. *max_depth* -hyperparametri määrää päätöspuun syvyyden, eli kuinka monta solmua päätöspuussa on.

Kun päätöspuun hyperparametrit optimoidaan, saadaan kuvan 22 mukaiset arvot

```
{'min_samples_split': 10, 'min_samples_leaf': 4, 'max_depth': 2}
```

KUVA 22. Päätöspuun optimoidut hyperparametrit

Kuten 22 kuvasta nähdään, kaikki muut hyperparametrit näyttävät hyvältä paitsi päätöspuun maksimisyvyydeksi saadaan vain 2.

3.5.2 Satunnaisen metsän hyperparametrien optimointi

Satunnaisessa metsässä on samat hyperparametrit kuin päätöspuussa, johon on lisättynä metsän päätöspuiden lukumäärä, jotka suorittavat äänestämisen parhaasta ulostulosta. Hyperparametrin nimi on *n_estimators.*

Optimoidun satunnaisen metsän hyperparametrit ovat kuvassa 23.

```
{'n_estimators': 650,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_depth': 945}
```

Kuva 23. Satunnaisen metsän optimoidut hyperparametrit

Kun kuvia 22 ja 23 verrataan, nähdään, että satunnaisissa metsissä on huomattavasti syvemmät päätöspuut, kuin pelkässä päätöspuussa. Toisaalta minimiotos määrät päätöspuun solmussa ovat pienemmät satunnaisessa metsässä, kuin päätöspuussa.

3.5.3 Erittäin satunnaisten puiden hyperparametrien optimointi

Erittäin satunnaisten puiden hyperparametrit ovat täysin samat, kuin satunnaisessa metsässä. Optimoiduiksi hyperparametreiksi erittäin satunnaisille puille saatiin kuvan 24 mukaiset tulokset.

```
{'n_estimators': 150,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_depth': 245}
```

Kuva 24. Erittäin satunnaisten puiden optimoidut hyperparametrit

Kun kuvaa 24 ja 23 verrataan, huomataan puiden määrän ja maksimi syvyyden olevan pienempi erittäin satunnaisissa puissa. Toisaalta molemmat minimi solmujen määrät ovat suuremmat erittäin satunnaisissa puissa kuin satunnaisessa metsässä.

3.5.4 Neuroverkolle optimoidut hyperparametrit

Neuroverkolle valitaan optimointialgoritmi kahdesta vaihtoehdosta, SGD ja Adam optimointialgoritmista, piilotettujen tasojen lukumäärä, opetustahti ja neuronien lukumäärä. Optimoidun neuroverkon hyperparametrit ovat kuvassa 25.

```
{'algo': 'sgd', 'how_many_layers': 1, 'lr': 0.1, 'neurons': 64}
```

KUVA 25. Optimoidun neuroverkon hyperparametrit

Kuten kuvasta 25 nähdään, optimoidun neuroverkon optimointialgoritmi on SGD, piilotettujen tasojen lukumäärä on 1, opetustahti on 0.1 ja neuronien lukumäärä sillä tasolla on 64.

3.6 Piirteiden valinta

Piirteitä on suuri määrä datassa, koska kategoriallisille kolumneille tehtiin One-hot-enkoodaus. Suuri piirteiden määrä pitäisi aiheuttaa malleihin kohina, joka lisää mallien virhettä. Kuten teoriassa käytiinkin läpi, piirteiden valintaa voidaan tehdä päätöspuihin perustuvilla luokittelijoilla. Työn merkitsevimpien piirteiden valinta tehdään erittäin satunnaisten puiden -mallilla, koska siinä on kaikista vähiten varianssia. Piirteiden valinta tehdään vain neuroverkoille. Scikit-learn-kirjaston opetetuista päätöspuihin perustuvista malleista saa piirteiden merkitsevyydet ulos *feature_importances_*-attribuutilla. Valitaan aina n merkitsevintä piirrettä neuroverkkoa varten.

4 Mittaukset ja tulosten käsittely

Kaikista toteutuksessa tehdyistä malleista mitataan varianssi ja bias -virheet. Varianssin ja bias -virheiden laskemiseen on käytetty apuna *mlxtend* -kirjaston *evaluate.bias_variance_*decomp -metodia. Samalla metodilla lasketaan varianssi ja bias -virheet 0-1 välille skaalatusta virheestä. Kappaleessa mitataan malleista myös ROC-käyrän pinta-ala, tarkkuus, tarkkuus opetuksessa ja F1-arvo.

Tehdään neuroverkkomalleille mittaukset kaikilla piirteillä eli 764-piirteellä, 382-piirteellä, 191-piirteellä, 95-piirteellä, 48-piirteellä ja 24-piirteellä.

4.1 Mittaukset

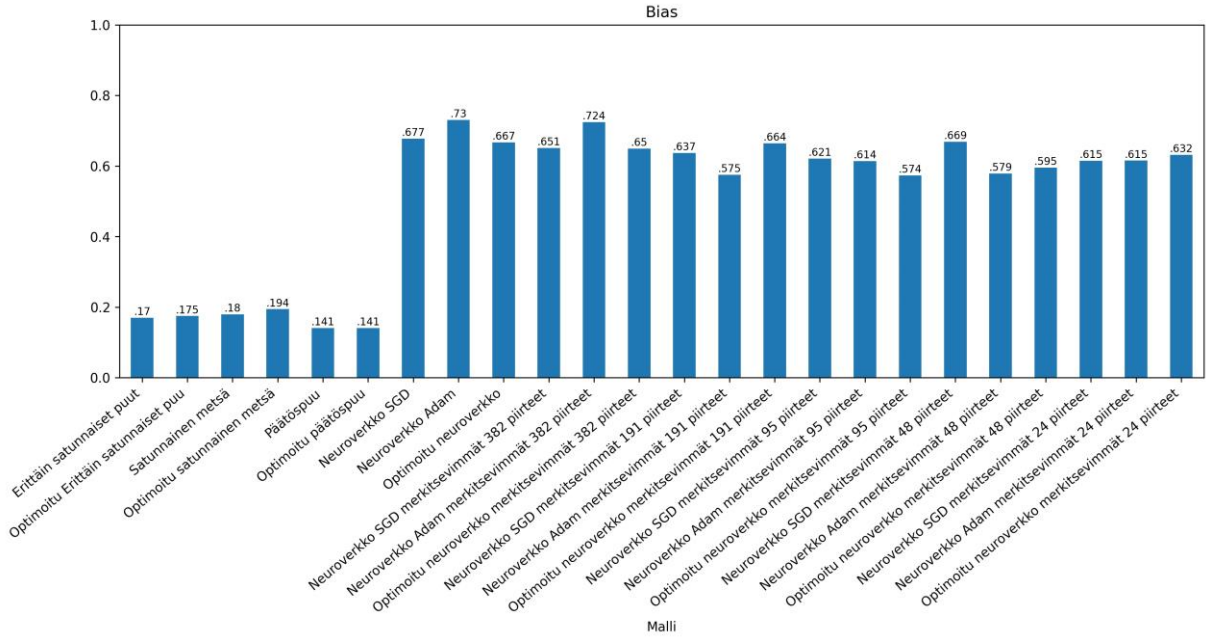
Taulukossa 5 on kaikki mittaukset mitä malleille tehtiin. Ensimmäisellä kolumnilla on mallien nimet ja ensimmäisellä rivillä on metriikkojen nimet.

TAULUKKO 4. Mallien mittaukset

Malli	ROC-käyrän pinta-ala	Tarkkuus	Bias	0-1 häviö	Varianssi	Tarkkuus opetuksessa	F1-arvo
Erittäin satunnaiset puut	0,789115898	0,815538517	0,169902913	0,195097087	0,072475728	**1**	0,8584713
Optimoitu Erittäin satunnaiset puu	0,78378893	0,820731725	0,174757282	0,190291262	0,064563107	0,971549627	0,86673097
Satunnainen metsä	0,78100624	0,8187064	0,17961165	0,188834951	0,068932039	0,999988584	0,86531466
Optimoitu satunnainen metsä	0,777598875	0,822387208	0,194174757	0,188495146	0,05461165	0,951076635	0,87071604
Päätöspuu	0,806277407	0,842660216	**0,140776699**	0,154174757	0,053495146	0,896183668	0,88296802
Optimoitu päätöspuu	0,810834873	**0,856039527**	**0,140776699**	**0,140776699**	**0**	0,857343799	**0,89571487**
Neuroverkko SGD	0,841699214	0,8009072	0,677419355	0,677419355	**0**	0,874307312	0,84596161
Neuroverkko Adam	0,820628288	0,792775159	0,730434783	0,730347826	8,70E-05	0,932140981	0,8387095
Optimoitu neuroverkko	0,836914499	0,80220594	0,666666667	0,665111111	0,001555556	0,921450126	0,84497501
Neuroverkko SGD merkitsevimmät 382 piirteet	0,837328002	0,799279715	0,650793651	0,650793651	**0**	0,86814391	0,84563536
Neuroverkko Adam merkitsevimmät 382 piirteet	0,821435654	0,798123167	0,724137931	0,724137931	**0**	0,929873623	0,84256396
Optimoitu neuroverkko merkitsevimmät 382 piirteet	0,837380171	0,802017795	0,649635036	0,64350365	0,006131387	0,916929508	0,84346217
Neuroverkko SGD merkitsevimmät 191 piirteet	0,837682213	0,799668593	0,637096774	0,637096774	**0**	0,853300318	0,84769246
Neuroverkko Adam merkitsevimmät 191 piirteet	0,851679779	0,822257473	0,57480315	0,574330709	0,000472441	0,92059586	0,86105198
Optimoitu neuroverkko merkitsevimmät 191 piirteet	0,840696093	0,811904351	0,6640625	0,6609375	0,0040625	0,902660525	0,85463883
Neuroverkko SGD merkitsevimmät 95 piirteet	0,832849519	0,804625638	0,621212121	0,621212121	**0**	0,837586792	0,8556719
Neuroverkko Adam merkitsevimmät 95 piirteet	0,84435998	0,82168167	0,613636364	0,613636364	**0**	0,898397827	0,86342253
Optimoitu neuroverkko merkitsevimmät 95 piirteet	0,842684727	0,821683767	0,573643411	0,571317829	0,002325581	0,884346035	0,86264328
Neuroverkko SGD merkitsevimmät 48 piirteet	0,816375191	0,830722886	0,668965517	0,668965517	**0**	0,841415947	0,8781604
Neuroverkko Adam merkitsevimmät 48 piirteet	0,835981262	0,84259266	0,578947368	0,578947368	**0**	0,883273048	0,882393
Optimoitu neuroverkko merkitsevimmät 48 piirteet	0,828045492	0,842980868	0,595238095	0,595238095	**0**	0,867924539	0,88490941
Neuroverkko SGD merkitsevimmät 24 piirteet	0,822518886	0,843770962	0,614864865	0,614864865	**0**	0,846105431	0,88680953
Neuroverkko Adam merkitsevimmät 24 piirteet	**0,853202655**	0,85192064	0,615384615	0,615384615	**0**	0,865838239	0,89041673
Optimoitu neuroverkko merkitsevimmät 24 piirteet	0,840322272	0,853748859	0,631578947	0,631578947	**0**	0,857366638	0,89306822

Taulukossa 4 on tummennettuna parhaimmat arvot. Taulukon 4 pohjalta piirrettiin pylväsdiagrammit jokaista metriikkaa kohti, jotta tulosten tulkinta olisi helpompaa.

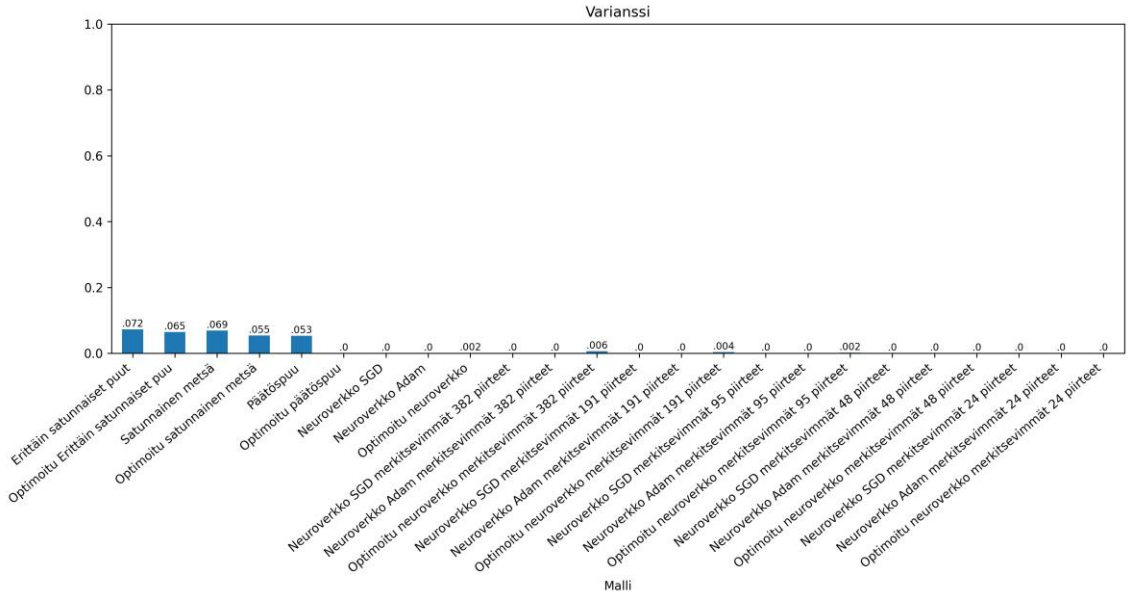
Kuviossa 11 on kaikkien tehtyjen mallien bias-virheet.



KUVIO 11. Mallien Bias-virheet

Kuten kuviosta 11 nähdään, pienimmät bias-virheet ovat päätöspuihin perustuvilla malleilla ja pienin bias-virhe saadaan päätöspuilla. Kuvion 9 mukaan päätöspuut tekevät kaikista vähiten oletuksia datasta, kun taas Adam-optimointialgoritmilla tehty neuroverkkomalli tekee eniten oletuksia datasta. Toisaalta, kun mitataan bias-virhe Adam-optimointialgoritmilla tehty neuroverkko vain merkitsevimmällä 191 tai 95 -piirteellä, saadaan pienin bias-virhe neuroverkkomalleista.

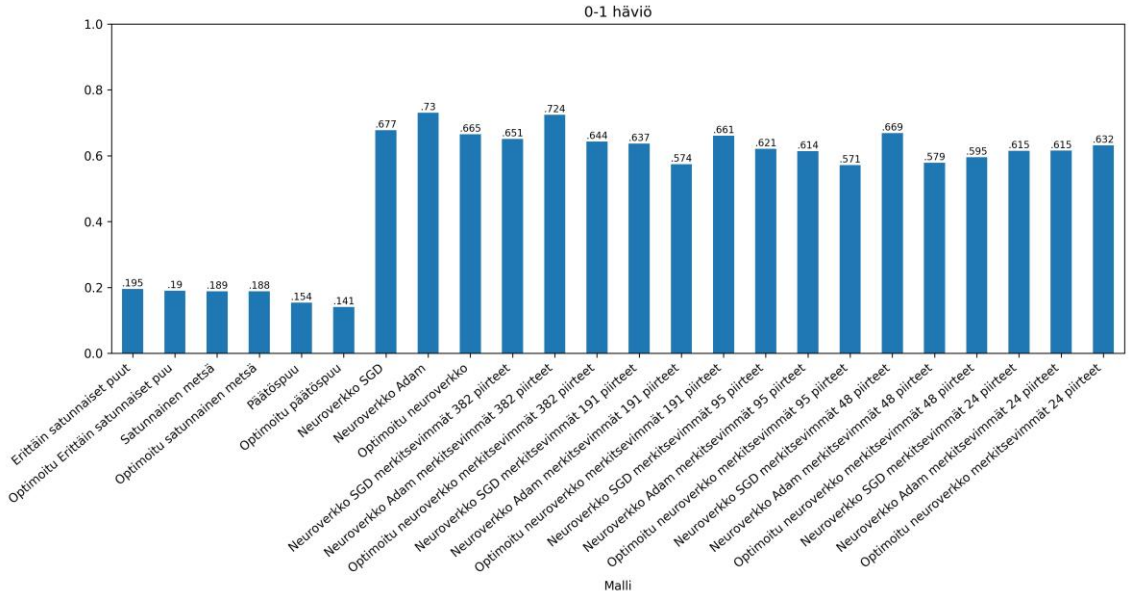
Kuviossa 12 on mallien varianssivirheet.



KUVIO 12. Mallien varianssivirheet

Kuviossa 12 suurimmat varianssivirheet saadaan päätöspuihin pohjautuvilla malleilla. Ainoastaan optimoidulla neuroverkolla on ylipäätänsä tarpeeksi iso varianssivirhe neuroverkkomalleista, jotta sen määrä näkyy kuviossa.

Kuviossa 13 on mallien 0-1 häviö.

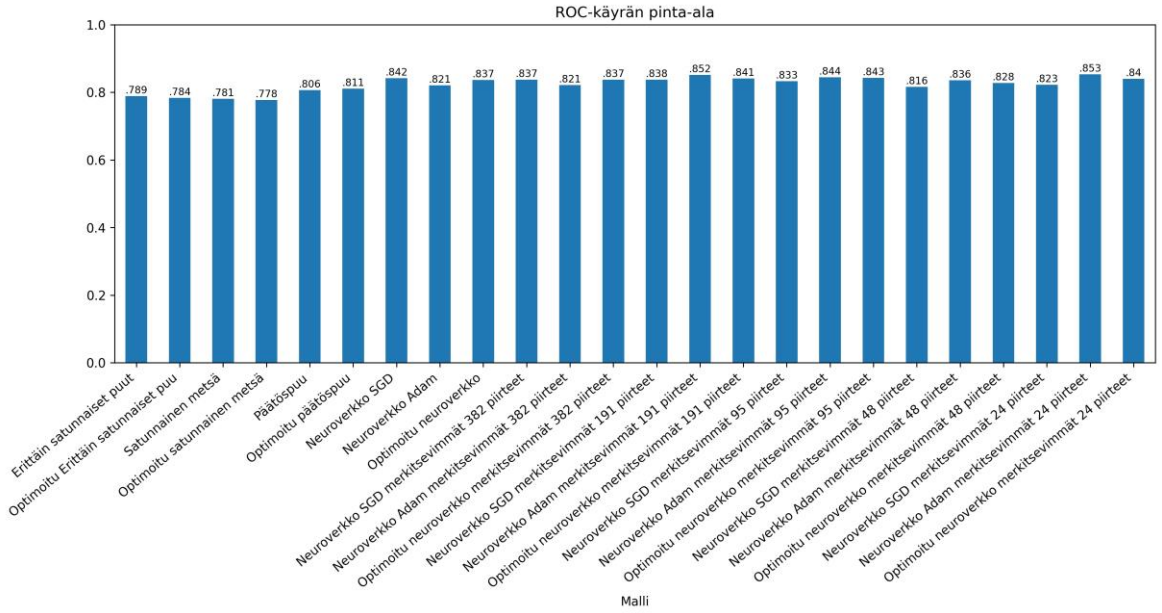


KUVIO 13. Mallien 0-1 häviö.

Kuvio 13 näyttää hyvin paljon kuviolta 11, koska suurin osa mallien virheestä tulee kuvion 11 bias-virheestä. Eli voidaan päätellä, että kaikki koneoppimismallit tekevät käytettyyn dataan oletuksia. Samat oletukset tehdään riippumatta siitä

mitä osaa datasta käytetään opettamiseen ja testaamiseen tai mitä toteutuksessa tehtyä mallia käytetään.

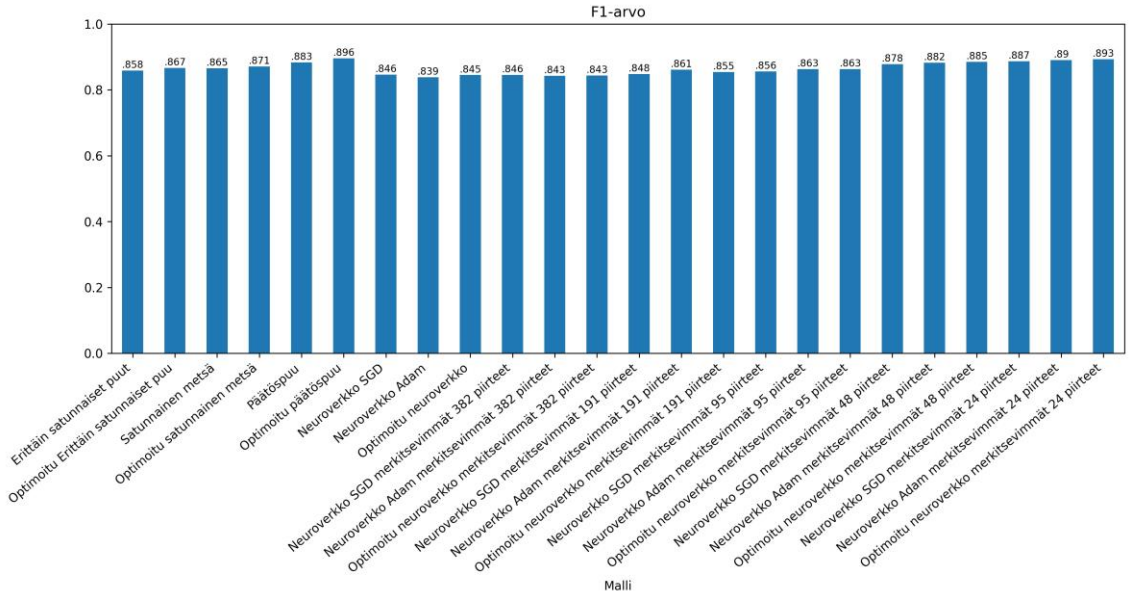
Kuviossa 14 on ROC-käyrän pinta-alan arvot jokaiselle mitatulle mallille.



KUVIO 14. Mallien ROC-käyrän pinta-ala

Kuviossa 14 paras ROC-käyrän pinta-ala saadaan neuroverkolla, jossa käytetään Adam -optimointialgoritmia ja käytetään vain merkitsevintä 24 piirrettä. Päätöspuihin pohjautuvilla malleilla ROC-käyrän arvo on pienempi, kuin neuroverkoilla. Eli päätöspuihin pohjautuvilla malleilla on enemmän hankaluuksia luokitella jompaakumpaa luokkaa, myyntimahdollisuuden voittamista tai häviämistä, kuin neuroverkoilla.

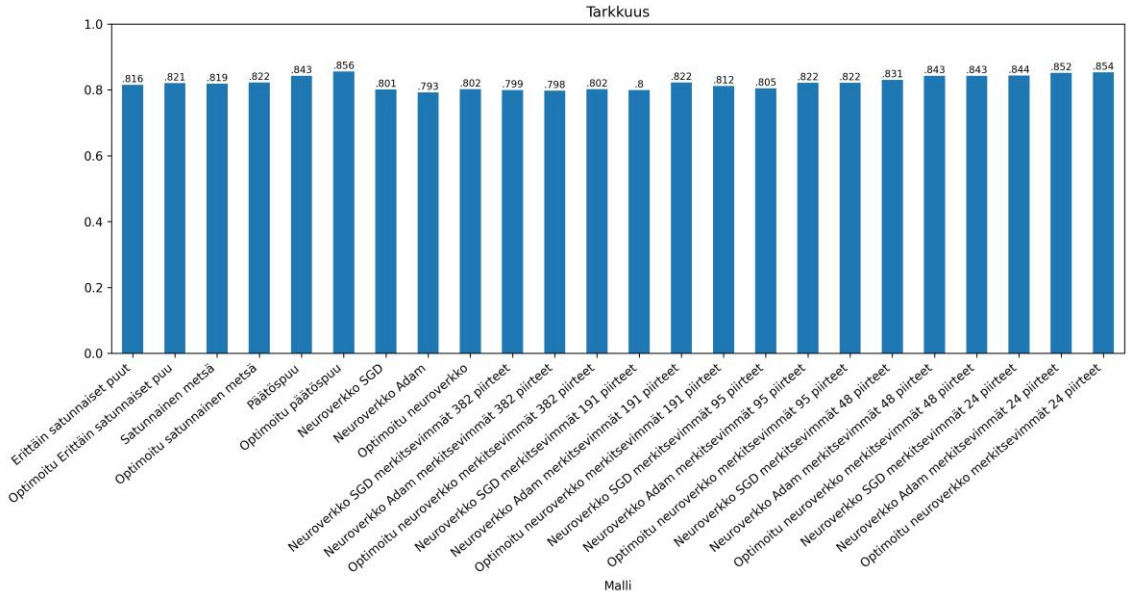
Kuviossa 15 on mallien F1-arvot.



KUVIO 15. Mallien F1-arvot

Kuviossa 15 parhaimman F1-arvon saa optimoitu päätöspuu. F1-arvot kasvavat neuroverkoissa, mitä vähemmän piirteitä mallille jätetään.

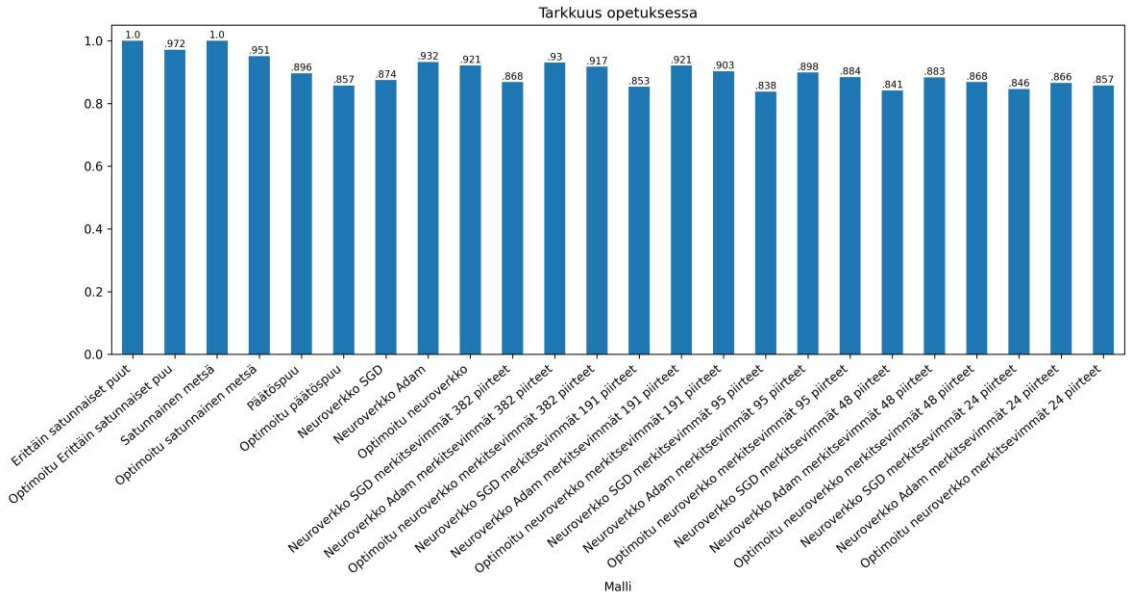
Kuviossa 16 on kaikkien mallien tarkkuudet.



KUVIO 16. Kaikkien mallien tarkkuudet

Kuviossa 16 paras tarkkuus saadaan optimoidulla päätöspuulla. Tarkkuudessa käy sama kuin F1-arvon kanssa, eli tarkkuus kasvaa mitä vähemmän piirteitä käytetään neuroverkkomallissa.

Kuviossa 17 on vielä mallin tarkkuus opetusdataa vasten. Pylväsdiagrammi piirrettiin, koska siitä nähdään helposti, jos malli ylisovittuu opetusdataa vasten.



KUVIO 17. Mallien tarkkuus opetuksen jälkeen

Kuviossa 17 nähdään, että Erittäin satunnaiset puut ja Satunnainen metsä -mallit saivat 100 % tarkkuuden opetuksessa, vaikka ne saivat huonot tarkkuudet kuviossa 16. Voidaan siis päätellä, että kyseiset algoritmit ylisovittuivat testidataan, joka aiheutti mallien huonon menestymisen muissa metriikoissa. Kuviosta 17 nähdään myös, että vaikka optimoidulla päätöspuulla oli lähes huonoin tarkkuus, mutta opetus dataa vasten oli sen tarkkuus kuitenkin kuviossa 16 paras.

4.2 Tulosten käsittely

Kuvioiden 11-33 pohjalta voi päätellä, että suurin osa jokaiseen malliin syntyvästä virheestä on bias-virhettä eli malleilla on hankala olla tekemättä oletuksia käytetystä datasta.

Toisaalta kuvioiden 14–17 pohjalta huomataan, että kaikista malleista saatiin yllättävänkin tarkkoja. Eli voidaan päätellä, että käytetyssä datassa on piirteitä tai piirrekombinaatioita, jotka korreloivat myyntimahdollisuuden voiton ja häviön kanssa.

Optimoitu päätöspuu myyntimahdollisuus dataa vasten on hyvä esimerkki lähes parhaimmasta mahdollisesta mallista. Optimoidun päätöspuun 0-1 häviö on pienin ja se on saatu juuri oikealla määrällä bias-virhettä ja varianssivirhettä. Vaikka ROC-käyrän pinta-ala optimoidulla päätöspuulla ei ole paras kaikista malleista, mutta on se kuitenkin hyvin lähellä parhaimman neuroverkon ROC-käyrän arvoa. Mallin tarkkuus voisi olla tietysti aina parempi, mutta se paranee, kun saadaan lisää dataotoksia, joilla opettaa koneoppimismalleista tarkempia.

5 Pohdinta

Parhaaksi malliksi saatiin optimoitu päätöspuu noin 86 % tarkkuudella. Optimoidun päätöspuun hyperparametrit ovat kuvassa 22. Optimoidulla päätöspuulla oli parhaimmat arvot kaikista muista metriikoista, paitsi ROC-käyrän pinta-alasta ja varianssivirheestä. ROC-käyrän pinta-ala optimoidulla päätöspuulla oli kuitenkin hyvin lähellä parhaimman mallin arvoa. Vaikka varianssivirhe oli isompi kuin neuroverkkomalleilla, saatiin pienin 0-1 häviö optimoidulla päätöspuulla. Toisaalta optimoidun päätöspuun syvyys on vain 2, eli siinä on 3 solmua ja 2 lehtisolmua eli dataan on todennäköisesti jäänyt jotain piirteitä, jotka asetetaan myyntimahdollisuuden voittamisen tai häviämisen jälkeen.

Päätöspuuta on myös helppo ymmärtää ja siitä saadaan selville, miten tiettyyn ennustukseen se pääsee, koska solmut kertovat suoraan kuvion 2 mukaisesti ehdot jokaiselle ennustukselle. Tämä on kätevää, koska jos joku työntekijä haluaa tietää, miten ennustuksesta saadaan parempi, voi hän suoraan katsoa kuvaa puun solmuista.

Kaikista malleista saadaan kuitenkin parempia, kun myyntimahdollisuuksien lukumäärä kasvaa tietokannassa. Sen takia malleja kannattaa tulevaisuudessa opettaa uudelleen ja optimoida uudestaan, koska dataotoksien määrällä on suuri ero mallien suoriutumisen kanssa.

Ennen kuin mallia voi alkaa käyttämään ennustamiseen, pitäisi päättää mitkä piirteet tai minkä verran piirteitä on löydyttävä myyntimahdollisuudelta, jotta sille kannattaa lähteä ennustamaan lopputulosta. Ei ole järkeä esimerkiksi ennustaa myyntimahdollisuutta, jolle on annettu vain sen nimi.

Vaikkakaan mallit eivät ole 100 % tarkkoja, voi niitä silti hyödyntää. Lopullisen mallin käyttötarkoitus voisi olla, että jos myyjällä on usea myyntimahdollisuus työnalla, myyjä voi keskittyä myyntimahdollisuuteen jolle malli antaa parhaimman todennäköisyyden onnistua.

Toinen käyttötarkoitus voisi olla, että jos myyjä ei tiedä mikä arvo pitäisi laittaa johonkin kohtaan myyntimahdollisuudella, voisi malli ennustaa siihen todennäköisimmän vaihtoehdon, jotta myyntimahdollisuus voitettaisiin.

Kolmas käyttötarkoitus mallille voisi olla, että jos on olemassa jokin kenttä, jonka muuttamisella malli antaisi paljon todennäköisemmän tuloksen myyntimahdollisuuden voittamiselle, voisi malli ehdottaa tätä myyjälle.

Työssä tehdyissä malleissa on kuitenkin ongelmia. Jos halutaan opettaa mallia lisää myyntimahdollisuuksien lukumäärän kasvaessa, malli pitää opettaa alusta asti uudestaan, koska numeerisille kolumneille asetettiin kolumnin aritmeettinen keskiarvo tyhjien arvojen tilalle ja uuden datan tullessa aritmeettinen keskiarvo muuttuu. Joko numeeristen kolumnien tyhjät arvot pitää muuttaa aritmeettisesta keskiarvosta johonkin muuhun tai jatkokehitystä varten tulee valita sellainen malli, jossa olisi mahdollisimman pieni varianssi. Tällöin ei ole niinkään väliä, jos malli joudutaan uudelleen opettamaan.

Työssä käytiin paljon aiheita läpi, joten työn rajaamiseksi raskas laskenta ja osa koodista jätettiin tämän työn ulkopuolelle. Opinnäytyössä esitetyistä lähteistä voi etsiä lisää tietoa aiheeseen liittyen.

LÄHTEET

IBM Machine Learning. 2020. Machine Learning. Artikkeli. Julkaistu 15.7.2020. Luettu 08.06.2021 https://www.ibm.com/cloud/learn/machine-learning

Andrew Ng. n.da. Extract Data Conference. Diaesitys. Katsottu 08.06.2021 https://www.slideshare.net/ExtractConf

Frankenfield J. 2021. Artificial Intelligence (AI). Artikkeli. Julkaistu 8.3.2021. Luettu 23.6.2021. https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp

Aisolab. 2017. INTRO TO AI #1: ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, NEURAL NETWORKS AND DEEP LEARNING: WHAT ARE THE DIFFERENCES? Artikkeli. Julkaistu 14.9.2017. Luettu 23.6.2021. https://aiso-lab.com/intro-to-ai-1-artificial-intelligence-machine-learning-neural-networks-and-deep-learning-what-are-the-differences/

Louppe G. 2014. Understanding Random Forests: From Theory to Practice. Väitöskirja. Julkaistu 6.2021. https://arxiv.org/pdf/1407.7502.pdf

Mishra S. 2017. Unsupervised Learning and Data Clustering. Artikkeli. Julkaistu 20.5.2017. Luettu 27.6.2021. https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a

Ho D. 2020. NBDT: Neural-Backed Decision Trees. Tekninen raportti. Julkaistu 26.5.2020. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-65.pdf

Corporate Financial Institute. 2021. Random Forest. Artikkeli. Luettu 29.6.2021. https://corporatefinanceinstitute.com/resources/knowledge/other/random-forest/

Chakure A. 2019. Random Forest Regression. Artikkeli. Julkaistu 29.6.2019. Luettu 29.6.2021. https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f

Pierre G. & Damien E. & Louis W. 2005. Extremely randomized trees. Artikkeli. Julkaistu 2006. Luettu 30.6.2021. https://link.springer.com/content/pdf/10.1007/s10994-006-6226-1.pdf

Abel L. & Peirson V & E. Meltem Tolunay. 2018. Dank Learning: Generating Memes Using Deep Neural Networks. Tutkimuspaperi. Julkaistu 8.6.2018. https://arxiv.org/pdf/1806.04510.pdf

Shukla L. 2019. Designing Your Neural Networks. Artikkeli. Julkaistu 23.9.2019. Luettu 19.7.2021. https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed

Brownlee J. 2017. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. Julkaistu 13.1.2017. Luettu 19.7.2021. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Kumar A. 2020. Overfitting & Underfitting Concepts & Interview Questions. Artikkeli. Julkaistu 16.12.202020. Luettu 20.7.2021. https://vitalflux.com/overfitting-underfitting-concepts-interview-questions/

Bushaev V. 2017. Stochastic Gradient Descent with momentum. Artikkeli. Julkaistu 4.12.2017. Luettu 22.7.2021. https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

Bodepudi R. 2021. Most Popular Programming Languages & Why They’re Useful in Machine Learning. Artikkeli. Päivitetty 31.5.2021. Luettu 22.7.2021. https://neptune.ai/blog/programming-languages-machine-learning

Numpy Dokumentaatio n.d.. What is Numpy? Dokumentaatio. Luettu 24.7.2021. https://numpy.org/doc/stable/user/whatisnumpy.html

Pandas Dokumentaatio n.d.. Intro to data structures. Luettu 28.7.2021 https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html

Pal Satyabrata. 2018. Scikit-learn Tutorial: Machine Learning in Python. Artikkeli. Julkaistu 25.11.2018. Luettu 2.8.2021. https://www.dataquest.io/blog/sci-kit-learn-tutorial/

Keras Dokumentaatio n.d.. About Keras. Dokumentaatio. Luettu 6.8.2021. https://keras.io/about/

Matplotlib Dokumentaatio n.d.. Matplotlib Documentation. Dokumentaatio. Luettu 8.8.2021. https://matplotlib.org/devdocs/index.html

Lakshmanan S. 2019. How, When, and Why Should You Normalize / Standardize / Rescale Your Data? Artikkeli. Julkaistu 16.5.2019. Luettu 8.8.2021. https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff

Kumar S. 2020. 7 Ways to Handle Missing Values in Machine Learning. Artikkeli. Julkaistu 24.6.2020. Luettu 9.8.2021. https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e

Bownlee J. 2020. Ordinal and One-Hot Encodings for Categorical Data. Artikkeli. Julkaistu 17.8.2020. Luettu 9.8.2021. https://machinelearningmastery.com/One-hot-encoding-for-categorical-data/

Shaikh R. 2018. Feature Selection Techniques in Machine Learning with Python. Artikkeli. Julkaistu 28.10.2018. Luettu 9.8.2021. https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e

Brownlee J. 2017. What is the Difference Between Test and Validation Datasets? Artikkeli. Julkaistu 14.8.2020. Luettu 9.8.2021. https://machinelearningmastery.com/difference-test-validation-datasets/

Mishra A. 2018. Metrics to Evaluate your Machine Learning Algorithm. Artikkeli. Julkaistu 24.2.2018. Luettu 10.8.2021. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

Narkhede S. 2018. Understanding AUC – ROC Curve. Artikkeli. Julkaistu 26.6.2018. Luettu 11.8.2021. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Brownlee J. 2016. Gentle Introduction to the Bias-Variance Trade-Off in Machine Learning. Artikkeli. Julkaistu 18.3.2016. Päivitetty 26.10.2019. Luettu 16.8.2021. https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

Bias-Variance Decomposition n.d.. Dokumentaatio. Luettu 22.8.2021. http://rasbt.github.io/mlxtend/user_guide/evaluate/bias_variance_decomp/

Brownlee J. 2019. Difference Between Classification and Regression in Machine Learning. Artikkeli. Julkaistu 11.12.2017. Päivitetty 22.5.2019. Luettu 24.8.2021. https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

Scikit-learn -kirjaston dokumentaatio. n.d. Documentation of Scikit-learn 0.21.3. Luettu 24.8.2021. https://scikit-learn.org/0.21/documentation.html

Google, Machine learning course. N.d. Classification: Precision and Recall. Luettu 25.8.2021. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

DeepAi – Epoch. n.d. What is an Epoch? Artikkeli. Luettu 27.8.2021. https://deepai.org/machine-learning-glossary-and-terms/epoch