

Markus Siitonen

ÄLYKÄS REGRESSIOTESTAUS: Koneoppimisen hyödyntäminen testien optimoinnissa

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Huhtikuu 2025

TIIVISTELMÄ

Markus Siitonen: Älykäs regressiotestaus: Koneoppimisen hyödyntäminen testien optimoinnissa
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Huhtikuu 2025

Koneoppiminen mahdollistaa sen, että tietokone pystyy tekemään päätöksiä ja havaitsemaan rakenteita datasta ilman, että kaikkia sääntöjä tarvitsee ohjelmoida etukäteen. Koneoppimista käytetään muun muassa kasvojentunnistuksessa, itseajavissa autoissa, kielimallien kehityksessä sekä lääketieteellisessä diagnostiikassa. Koneoppiminen jaetaan kolmeen pääluokkaan: ohjattuun oppimiseen, jossa malli oppii esimerkkidatan avulla, ohjaamattomaan oppimiseen, jossa malli etsii datasta itse rakenteita, ja vahvistusoppimiseen, jossa oppiminen tapahtuu palkkiojärjestelmän kautta.

Neuroverkot ovat keskeinen koneoppimisen menetelmä, joka jäljittelee ihmisaivojen toimintaa monimutkaisten tehtävien, kuten kuvantunnistuksen ja luonnollisen kielen käsittelyn, ratkaisemiseksi. Viime vuosina neuroverkkojen käyttö on laajentunut myös ohjelmistotestauksen optimointiin, jossa niitä hyödynnetään testitapausten analysoinnissa, virheiden ennakoinnissa ja testiautomaation parantamisessa.

Regressiotestaus on ohjelmistotestauksen menetelmä, jolla varmistetaan, että ohjelmistoon tehdyt muutokset eivät riko aiemmin toimineita ominaisuuksia. Tämä on erityisen tärkeää jatkuvan integraation ja ketterän kehityksen ympäristöissä, joissa ohjelmistoa päivitetään usein. Testausprosessia voidaan tehostaa optimoimalla testitapausten valintaa ja suorittamista. Testisarjan optimointi voidaan toteuttaa testitapausten priorisoinnilla (TCP), valinnalla (TCS) tai minimoinnilla (TSM) – tai näitä menetelmiä yhdistelemällä. Tässä yhteydessä nousee esiin tutkimuskysymys: miten koneoppimistekniikoita voidaan hyödyntää regressiotestauksessa ja testien optimoinnissa?

Koneoppiminen tarjoaa tehokkaita keinoja automatisoida ja optimoida regressiotestausta. Se voi tunnistaa tärkeimmät testitapaukset, vähentää tarpeettomia testejä ja parantaa testausprosessin tehokkuutta jatkuvassa integraatiossa ja ohjelmistokehityksen ylläpidossa. Tämä mahdollistaa nopeamman ja luotettavamman ohjelmistokehityksen. Useat tutkimukset ovat osoittaneet, että perinteisillä koneoppimismenetelmillä sekä edistyneillä neuroverkko- ja klusterointitekniikoilla voidaan saavuttaa merkittäviä ajallisia ja taloudellisia säästöjä.

Avainsanat: koneoppiminen, regressiotestaus, testitapausten priorisointi, testitapausten valinta, testisarjan minimointi, testiautomaatio

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

- Ei
- Kyllä

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni tutkielmanprosessin aikana seuraavia tekoälysovelluksia:

Tekoälysovellusten nimet ja versiot:

- Scopus AI
- ChatGPT GPT-4

Käyttötarkoitus:

- Scopus AI: Hyödynnetty opinnäytteen alkuvaiheessa tutkielman rajauksen ja laajuuden määrittelyyn. Sen avulla sain alustavan käsityksen tutkielmaan liittyvien käsitteiden lukumäärästä ja tärkeydestä. Scopus AI:n avulla laatimani ensimmäinen suunnitelma ja lähteet toimivat alkusysäyksenä, mutta päädyin lopulta hylkäämään ne ja tarkentamaan työn fokusta.
- ChatGPT:tä käytin niin ikään siihen, että yritin keksiä sen avulla mahdollisimman sopivia ja relevantteja käsitteitä työn sisällön kannalta. Lisäksi hyödynsin sitä joissakin lauseiden muotoilussa ja tekstin selkeyttämisessä, kun olin haasteissa kappaleiden luettavuuden kanssa. ChatGPT:tä käytin myös lähteiden tiivistelmien kääntämiseen suomeksi, mikä auttoi tekemään lähteiden relevanssia selkeämmäksi ja helpommin havainnollistettavaksi.

Osiot, joissa tekoälyä on käytetty:

Tiivistelmä, satunnaiset kohdat työn tekstinhuollon tukena

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

SISÄLLYSLUETTELO

1	Johdanto	1
2	Tutkimusmenetelmä.....	2
3	Koneoppiminen	3
3.1	Koneoppiminen lyhyesti	3
3.2	Ohjattu oppiminen	4
3.3	Ohjaamaton oppiminen	4
3.4	Vahvistusoppiminen	5
3.5	Neuroverkot	5
4	Regressiotestaus	7
4.1	Miksi regressiotestaus on olennainen osa ohjelmistokehitystä?	7
4.2	Testitapausten optimointimenetelmät	8
5	Koneoppiminen regressiotestauksessa	11
5.1	Perinteisten koneoppimismallien hyödyntäminen testitapausten valinnassa	11
5.2	Bayesin verkkoihin perustuva priorisointi	12
5.3	Neuroverkkopohjainen priorisointi	12
5.4	Klusterointipohjainen priorisointi	13
5.5	Koneoppimista hyödyntäviä testaustyökaluja	14
6	Pohdintaa	16
7	Yhteenveto	17
	Lähdeluettelo.....	18

1 Johdanto

Ohjelmistotestaus on olennainen osa ohjelmistokehitystä ja sen tavoitteena on varmistaa järjestelmän toiminnallisuus, luotettavuus ja suorituskyky. Eräs tärkeä testausvaihe on regressiotestaus, joka varmistaa, että ohjelmistoon tehdyt muutokset eivät riko aiemmin toimineita osia. Perinteiset regressiotestausmenetelmät perustuvat usein laajoihin testiajoihin, joissa suoritetaan suuri määrä testitapauksia riippumatta muutosten todellisesta vaikutuksesta. Tämä voi tehdä testauksesta tehotonta, hidasta ja resursseja kuluttavaa, erityisesti suurissa ja monimutkaisissa ohjelmistojärjestelmissä.

Viime vuosina koneoppimistekniikat ovat nousseet lupaavaksi ratkaisuksi ohjelmistotestauksen automatisointiin ja optimointiin. Koneoppimisen avulla voidaan analysoida aiempia testituloksia, tunnistaa muutosten vaikutusalueet ja ennustaa, mitkä testitapaukset ovat merkityksellisimpiä suorittaa. Älykkään regressiotestauksen ansiosta testiajat lyhenevät ja resurssien käyttöä voidaan optimoida. Koneoppimista voidaan hyödyntää myös virheiden ennustamisessa, testikattavuuden parantamisessa ja visuaalisessa laadunvarmistuksessa, mikä tekee siitä monipuolisen työkalun ohjelmistokehityksen laadunvarmistusprosessissa.

Tämän tutkielman tavoitteena on tarkastella koneoppimisen roolia regressiotestauksessa ja testien optimoinnissa. Tutkimuskysymyksenä on: *Miten koneoppimistekniikoita voidaan hyödyntää regressiotestauksessa ja testien optimoinnissa?* Tutkielmassa analysoidaan koneoppimisen mahdollisuuksia ohjelmistotestauksessa ja tarkastellaan, mitkä koneoppimisen osa-alueet soveltuvat testauksen eri vaiheisiin.

Johdannon jälkeen käydään läpi koneoppimisen perusteita, minkä jälkeen syvennytään regressiotestauksen optimointiin ja siihen, miten koneoppiminen voi parantaa testien valikointia ja suoritusjärjestystä. Lopuksi tarkastellaan käytännön sovelluksia ja esimerkkejä siitä, miten koneoppimista hyödynnetään regressiotestauksessa nykypäivänä sekä millaisia mahdollisuuksia se tarjoaa tulevaisuudessa.

2 Tutkimusmenetelmä

Tämä tutkielma on toteutettu kirjallisuuskatsauksena, jossa analysoidaan aiempia tutkimuksia koneoppimisen soveltamisesta ohjelmistotestaukseen. Aineistona käytetään vertaisarvioituja artikkeleita, konferenssijulkaisuja ja teknisiä raportteja, joita haetaan tietokannoista kuten IEEE Xplore, ACM Digital Library ja Google Scholar.

Aineiston analyysi keskittyy koneoppimisen peruskäsitteisiin, regressiotestauksen menetelmiin sekä koneoppimiseen pohjautuviin ratkaisuihin testitapausten priorisoinnissa ja optimoinnissa. Tavoitteena on tunnistaa keskeiset optimointitekniikat, niiden soveltuvuus ohjelmistotestaukseen sekä mahdolliset haasteet ja rajoitteet. Lisäksi tarkastellaan erilaisia koneoppimismenetelmiä siinä määrin kuin se on tämän työn laajuuden puitteissa perusteltua. Menetelmien tehokkuutta ja käytännön sovellettavuutta arvioidaan ohjelmistotestauksen laadun ja resurssitehokkuuden näkökulmista.

Hakuprosessissa käytettyjä hakusanoja ovat muun muassa "*machine learning*", "*regression testing*", "*test case prioritization*", "*test case optimization*" ja "*GUI regression testing*", sekä näiden yhdistelmiä. Hakutuloksia on rajattu ajallisesti vuosille 2010–2025, jotta tarkastelu kohdistuu ajankohtaisiin ja relevantteihin lähteisiin. Hakusanoja on muokattu ja täydennetty iteratiivisesti hakutulosten analyysin perusteella.

Lähteet valittiin ensisijaisesti otsikon ja tiivistelmän perusteella sekä sen mukaan, oliko julkaisu vertaisarvioitu, mikä helpotti luotettavuuden arviointia. Aineistoa kerätessäni havaitsin, että aihealue on laajempi kuin alun perin oletin ja jouduin rajaamaan alkuperäistä tutkimusideaani huomattavasti.

Löydetty aineisto jaoteltiin sen mukaan, käsittelevätkö julkaisut koneoppimista, regressiotestausta vai molempia. Tämä jaottelu muodostui lopulta myös tutkielman rakenteen perustaksi.

3 Koneoppiminen

Koneoppiminen on tietojenkäsittelyn osa-alue, jossa tietokone opetetaan tunnistamaan kuvioita ja tekemään päätöksiä aiemman kokemuksen perusteella ilman erikseen ohjelmoituja sääntöjä. Koneoppiminen perustuu algoritmeihin, jotka mukautuvat ja optimoivat koulutusdatan avulla. Sen sovelluksia ovat esimerkiksi kasvojentunnistus, suosittelujärjestelmät ja automaattinen vianetsintä. Koneoppiminen on keskeinen osa tekoälyä ja sitä hyödynnetään laajasti eri aloilla, kuten lääketieteessä, vähittäiskaupassa, finanssialalla ja ohjelmistokehityksessä. [1]

3.1 Koneoppiminen lyhyesti

Tekoälyn (AI) ja koneoppimisen välinen ero ei ole aivan selvä. Joissain tapauksissa niitä pidetään jopa synonyymina, ja tekoäly saatetaan nähdä kansankielisenä terminä, jota käytetään erityisesti ei-asiantuntijoille suunnatussa keskustelussa [2]. Yleinen käsitys kuitenkin tuntuu olevan, että tekoälyn tavoitteena on luoda järjestelmiä, jotka kykenevät ihmisen kaltaiseen älykkyyteen, kun taas koneoppiminen keskittyy kehittämään algoritmeja, jotka mahdollistavat järjestelmien oppimisen datasta ja suorituskyvyn parantamisen itsenäisesti.

Koneoppimisessa käytössä on malli, eli matemaattinen rakenne tai kaava, joka oppii tunnistamaan kaavoja ja suhteita datassa. Malli voi olla ennustava, kuvaileva tai molempia. Ennustava malli tekee tulevaisuutta koskevia ennusteita, esimerkiksi arvioi, miten asiakas todennäköisesti käyttäytyy. Kuvaileva malli tuottaa tietoa olemassa olevasta datasta, kuten tunnistaa asiakasryhmiä tai ostokäyttäytymisen malleja. [1]

Koneoppiminen hyödyntää tilastotiedettä matemaattisten mallien rakentamisessa, sillä sen keskeinen tehtävä on tehdä johtopäätöksiä otoksista. Tietojenkäsittelytieteellä on tässä kaksiosainen rooli: Ensinnäkin koulutusvaiheessa tarvitaan tehokkaita algoritmeja optimointiongelmien ratkaisemiseen sekä suurten tietomäärien tallentamiseen ja käsitteelyyn. Toiseksi, kun malli on oppinut, sen esitystavan ja päättelyalgoritmin on oltava tehokkaita. Joissakin sovelluksissa oppimis- tai päättelyalgoritmin tehokkuus voi olla yhtä tärkeää kuin sen ennustustarkkuus. [1]

Koneoppiminen on yksi tietojenkäsittelytieteen kuumimmista tutkimusalueista. Digitaalisen teknologian kehittyessä syntyy yhä enemmän dataa, tietokoneiden laskentateho kasvaa, ja oppivat algoritmit kehittyvät vauhdilla. Varsinkin syväoppimisesta on tullut koneoppimisen hallitseva suuntaus viimeisen vuosikymmenen aikana. [1]

Koneoppiminen on jaettu kolmeen päälinjaan: ohjattu oppiminen, ohjaamaton oppiminen sekä vahvistusoppiminen. Näiden lisäksi puhutaan muun muassa neuroverkoista, joka on kuitenkin yleisnimi kokonaiselle koneoppimisen lähestymistavalle, eikä mikään yksittäinen menetelmä. [2]

3.2 Ohjattu oppiminen

Ohjatussa oppimisessa (supervised learning, SL) koneoppimismalli opetetaan yhdistämään selittäjän tai selittäjien (X) vaikutus vastemuuttujaan (Y), kun X:n ja Y:n arvot on määritelty ennalta koulutusdatassa. Esimerkiksi käytetyn auton hinnan ennustaminen perustuu auton ominaisuuksiin, kuten merkkiin, vuosimalliin ja ajettuihin kilometreihin. Malli oppii hahmottamaan tilastollisen tai matemaattisen yhteyden selittäjien (X) ja hinnan (Y) välille, jolloin se pystyy ennustamaan hintaa uusille autoille. [1]

Ohjattu oppiminen jakautuu kahteen eri ongelmatyyppiin: regressioon ja luokitteluun. Regressiomallissa vastemuuttujan arvo on jatkuva luku, kuten auton hinta tai ilman lämpötila. Luokittelussa vastemuuttuja on kategorinen, esimerkiksi ennustetaan jonkin elektronisen osan rikkoutumisen todennäköisyyttä (toimii = 0, rikki = 1). Kahden kategorian tapauksessa kyseessä on binääriluokittelu, jossa yleisesti käytetty menetelmä on logistinen regressio. Useamman kategorian tapauksessa luokittelu on moniluokkaista, jolloin siihen voidaan käyttää esimerkiksi moniluokkaista logistista regressiota tai muita koneoppimismenetelmiä. [1]

Yhdistelmämallit ovat edistyneitä koneoppimistekniikoita, joissa koulutetaan useita koneoppimismalleja ja yhdistetään niiden ennusteet suorituskyvyn parantamiseksi. Näitä menetelmiä käytetään laajasti luokittelussa, regressiossa ja poikkeamien havaitsemisessa. [3]

3.3 Ohjaamaton oppiminen

Ohjaamaton oppiminen (unsupervised learning, UL) eroaa ohjatusta siinä, että käytössä on ainoastaan selittäjä(t) (X), eikä vastemuuttujan (Y) arvoja ole määritelty. Tavoitteena on löytää aineistosta rakenteita, kuten usein toistuvia kuvioita tai luonnollisia ryhmittymiä. Yksi tapa tähän on tiheyden estimointi, jota käytetään esimerkiksi klusteroinnissa. Lisäksi ohjaamatonta oppimista hyödynnetään muun muassa ulottuvuuksien vähentämisessä ja poikkeavuuksien tunnistuksessa. [2]

Klusteroinnissa pyritään löytämään samankaltaisten syötteiden ryhmiä, eli klustereita. Esimerkiksi yritystä voi kiinnostaa selvittää asiakasdatan perusteella millainen on heidän asiakasprofiiliensa jakauma ja minkälaisia asiakkaita esiintyy useimmin. Toisin sanoen klusterointimalli ryhmittelee asiakkaat heidän ominaisuuksien perusteella, jolloin yritys voi kehittää kohdennettuja strategioita. Merkitsevästi muista poikkeavia asiakkaita nimitetään poikkeamiksi (outlier). [1]

Ohjaamatonta oppimista voidaan hyödyntää ohjatun oppimisen tukena, jos ohjaamaton oppiminen tuottaa tarvittavat luokittelutiedot. Tätä hyödynnetään itseohjatussa oppimisessa (self-supervised learning), jossa malli luo itse tarvitsemansa opetusdatan. Esimerkiksi tietokonenäössä ohjaamaton oppiminen voi ryhmitellä valokuvia luokkiin, joita

voidaan käyttää ohjatun oppimisen opetusaineistona. Jos menetelmä toimii hyvin, tulokset voivat vastata todellisia luokkia, joilla on selkeä merkitys ihmisen näkökulmasta (esimerkiksi 'koirat', 'kissat' ja 'linnut'). [2]

3.4 Vahvistusoppiminen

Vahvistusoppimisessa (reinforcement learning, RL) tavoitteena on oppia toimintapolitiikka, eli oikeiden toimintojen sarja, joka johtaa haluttuun lopputulokseen. Yksittäinen toimi on hyödyllinen vain, jos se on osa toimintapolitiikkaa, joka maksimoi pitkän aikavälin palkkiot. Hyvä esimerkki vahvistusoppimisesta ovat shakin tekoälysovellukset. Shakin säännöt ovat yksinkertaiset, mutta pelin strateginen monimutkaisuus tekee siitä haastavan oppimisongelman [1]. Vahvistusoppimiseen perustuvat tekoälyt, kuten AlphaZero, ovat oppineet pelaamaan shakkia ilman ihmisten esimerkkidataa, kehittämällä itse omia strategioitaan kokeilemalla ja saamalla palautetta pelituloksista.

Vahvistusoppiminen voidaan jakaa ala-aiheisiin, kuten itseohjattu oppiminen ja uuteen ongelmaan mukautuminen (domain adaptation & transfer learning). Itseohjatussa oppimisessa yksi tietolähde, kuten kosketussensori, voi ohjata oppimista, kun esimerkiksi robotti oppii välttelemään törmäyksiä. Ongelmaan mukautumisessa selvitetään, miten aiemmin opittu menetelmä siirretään uuteen tilanteeseen (lähes) ilman uusia opetus-esimerkkejä. [2]

Vahvistusoppimisessa tekoälyagentti oppii toimintatapansa vuorovaikutuksessa ympäristönsä kanssa. Tekoälyagentti havainnoi ympäristön tilaa, kuten jalkapallopelissä pelaajien ja pallon sijainteja, jonka perusteella se valitsee sopivan toiminnon. Jokainen toiminto tuottaa palkinnon ja agentin tavoitteena on maksimoida pitkän aikavälin palkintojen summa, kuten maalit, torjunnat tai otteluvoitot. Agentit voivat myös toimia yhteistyössä jonkin yhteisen tavoitteen saavuttamiseksi, esimerkiksi kokonainen robottijoukkue, joka pelaa jalkapalloa. [1]

3.5 Neuroverkot

Keinotekoiset neuroverkot (neural networks tai artificial neural networks), jotka olivat suosittuja jo 1980- ja 1990-luvuilla, jäivät tuolloin rajoitetun laskentatehon ja vähäisen datan vuoksi vähemmälle huomiolle. Nykyään, kun dataa on runsaasti ja prosessorit ovat nopeampia ja edullisempia, on pystytty kehittämään monikerroksisia syviä neuroverkkoja, jotka ovat osoittautuneet erittäin tehokkaiksi monilla alueilla. Lisäksi avoimen lähdekoodin ohjelmistot ovat tehneet syväoppimisen soveltamisesta helpompaa. [4]

Neuroverkot ovat laskennallisia malleja, jotka jäljittelevät ihmisaivojen toimintaa. Ne on suunniteltu tunnistamaan kuvioita, tekemään päätöksiä ja oppimaan datasta samalla tavalla kuin biologiset neuronit. Neuroverkot koostuvat kerroksista, joissa on toisiinsa

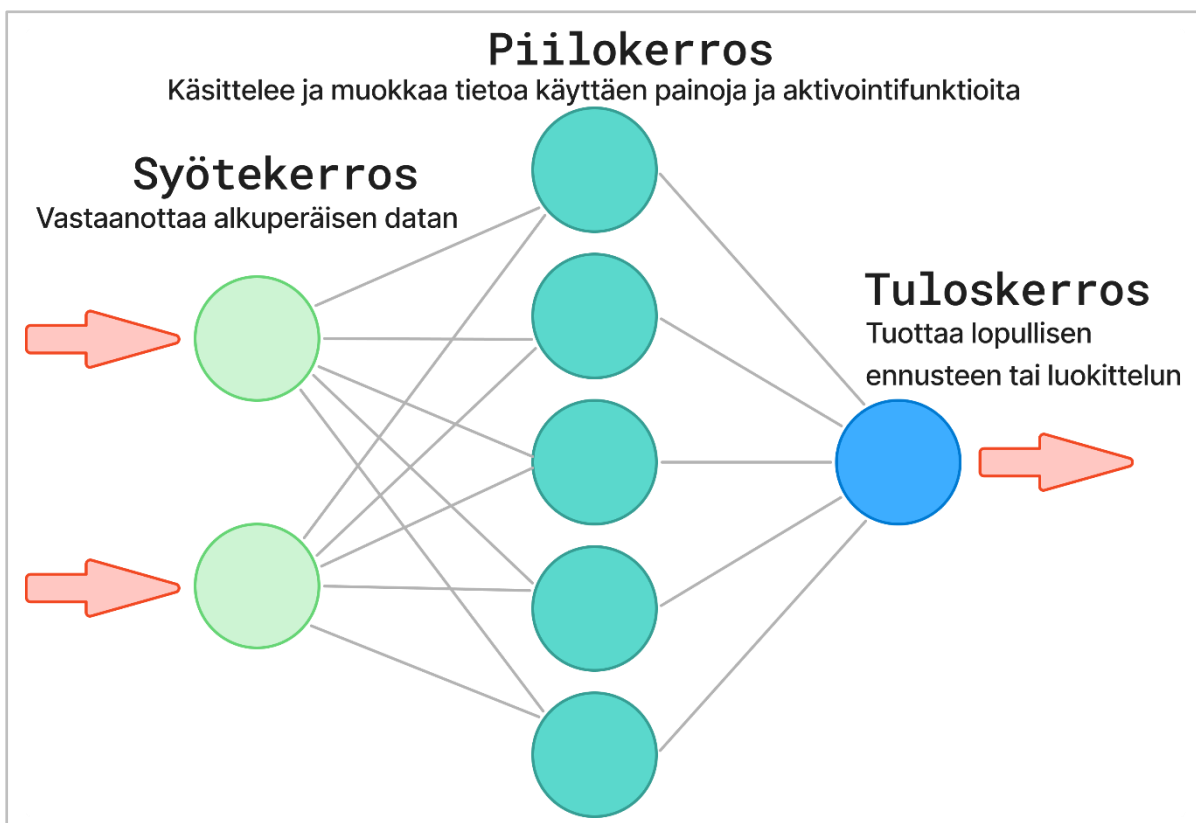
yhdistettyjä neuroneita. Neuroverkko sisältää syötekerroksen (input layer), yhden tai useampia piilokerroksia (hidden layer) ja tuloskerroksen (output layer) [4]. **Kuva 1** on esitetty yhden piilokerroksen neuroverkon rakenne.

Neuroverkon tietojenkäsittelyelementit koostuvat neuroneista, jotka on kytketty toisiinsa yhteyksillä. Jokaisella yhteydellä on painokerroin, joka mukautuu oppimisen edessä, mikä auttaa verkkoa tekemään tarkempia ennusteita. Tämä prosessi toteutetaan usein algoritmeilla, kuten takaisinlevitys (backpropagation). [4]

Nykyisin syvät neuroverkot ovat keskeisessä roolissa muun muassa kasvojentunnistuksessa, kuvatekstien tuottamisessa ja itseajavissa autoissa. Niiden kyky oppia monimutkaisia malleja suurista tietomääristä on mahdollistanut merkittäviä edistysaskelia muun muassa luonnollisen kielen käsittelyssä, lääketieteellisessä diagnostiikassa ja reaaliaikaisessa kuvantunnistuksessa. [5][6][7]

Esimerkiksi Go-pelissä neuroverkko on oppinut voittamaan ihmispelaajat pelissä, jonka aiemmin uskottiin olevan tekoälylle liian haastava. AlphaGo-ohjelma ei ainoastaan päihittänyt maailman parhaimpia pelaajia, vaan myös kehitti täysin uudenlaisia strategioita, joita ihmiset eivät olleet aiemmin käyttäneet. [8]

Vastaavasti syvät neuroverkot ovat mullistaneet muita pelejä, kuten shakin ja pokerin, sekä mahdollistaneet kehittyneitä sovelluksia, kuten puheen reaaliaikaisen kääntämisen ja taiteellisten kuvien generoinnin. Tekoälyn jatkuva kehitys viittaa siihen, että neuroverkkojen sovellusalueet tulevat laajenemaan entisestään eri teollisuudenaloilla. [9]



Kuva 1. Yhden piilokerroksen neuroverkko.

4 Regressiotestaus

Tieteellisissä yhteyksissä käytetään usein anglismia ”regressio”, jolla voi olla eri merkityksiä kontekstista riippuen. Ohjelmistotestauksessa regressiolla ei kuitenkaan viitata järjestelmän taantumiseen tai heikkenemiseen, vaan siihen, että aiemmin toimineet ominaisuudet voivat lakata toimimasta uusien muutosten tai päivitysten seurauksena. Vertailun vuoksi voidaan ajatella regressioanalyysiä, jossa tarkastellaan, miten tietyt muuttujat vaikuttavat lopputulokseen. Samalla tavalla regressiotestauksessa selvitetään, miten ohjelmistoon tehdyt muutokset vaikuttavat jo testattuihin toiminnallisuuksiin. Koska regressiotestausta tehdään pääasiassa ohjelmiston ylläpitovaiheessa, sen tehostaminen voi merkittävästi vähentää ylläpitokustannuksia [10].

4.1 Miksi regressiotestaus on olennainen osa ohjelmistokehitystä?

Regressiotestauksesta puhuttaessa tarkoitetaan sitä, kuinka tehdyt muutokset voivat vaikuttaa ohjelmiston aikaisempiin osiin, vaikka muutokset eivät suoranaisesti liittyneet näihin osiin. Toisin sanoen regressiotestauksessa tehdään oletus, että jokin on voinut muuttua suhteessa aiempaan koodiin. Siksi vanhat testit tulee ajaa uudelleen, jotta varmistetaan, ettei mikään ole mennyt rikki. [11]

Ohjelmiston koon ja monimutkaisuuden kasvaessa testisarjaan kertyy yhä enemmän testejä, mikä pidentää niiden suoritusaikaa. Erittäin suurten ohjelmistojen regressiotestaus voi olla todella aikaa vievää, pahimmillaan jopa useita päiviä [12]. Ohjelmistokehityksen lyhyempi elinkaari, kuten ketterän ohjelmistokehityksen mukainen lähestymistapa, asettaa rajoituksia ja haasteita sille, miten regressiotestaus pystytään toteuttamaan rajallisilla resursseilla. Mikropalveluarkkitehtuurin yleistyminen ohjelmistokehityksessä tuo mukanaan haasteita regressiotestaamiseen. Käytettäessä kolmannen osapuolen komponentteja, niihin tehtyjen muutosten vaikutuksia ei voida ennustaa, koska näiden komponenttien sisäinen rakenne ei ole käyttäjien tiedossa. [13]

Regressiotestauksessa, ja ohjelmistotestauksessa ylipäätään, puhutaan *white box*- ja *black box* -testauksesta. White box -testauksessa analysoidaan ohjelman koodin logiikkaa ja rakennetta, ja sitä käytetään sekä testauksen kattavuuden seurantaan että sopivan testidatan valintaan. Black box -testauksessa ohjelmistoa käsitellään ”mustana laatikkona”, ilman, että tiedetään sen sisäisestä koodista mitään, jolloin testausta tehdään syötteiden perusteella. White box -testaus on tehokkaampaa, kun taas black box -testaaminen on edullisempää ja helpompaa suorittaa. [14]

4.2 Testitapausten optimointimenetelmät

Regressiotestaukseen on kehitetty erilaisia testitapausten optimointimenetelmiä, joiden avulla testeihin käytettyä aikaa ja resursseja voidaan yrittää minimoida. Kolme keskeistä menetelmää ovat testitapausten priorisointi (TCP), testitapausten valinta (TCS) sekä testisarjan minimointi (TSM). Eri menetelmien hyötyjä ja haittoja on esitelty **Taulukko 1**. Näiden lisäksi on olemassa myös hybridimenetelmiä, jotka hyödyntävät useampaa eri optimointimenetelmää. Esimerkiksi ensin ajetaan testisarjan minimointi, jonka jälkeen jäljelle jääneet tapaukset vielä priorisoidaan. [16]

RT-lähestymistapa	Hyödyt	Haitat
Testitapausten priorisointi (TCP)	Hyödyllinen, kun uusia testitapauksia otetaan jatkuvasti huomioon testivaiheessa.	Aikavievä prosessi, joka voi johtaa suurempaan testisarjaan.
Testitapausten valinta (TCS)	Vähentää testauksen suoritus-aikaa. Sopii hyvin suurille projekteille	Voi jättää joitakin virheitä havaitsematta, jos valitut testit eivät kata kaikkia skenaarioita.
Testisarjan minimointi (TSM)	Nopeampi testisuoritus-aika ja pienemmät kustannukset.	Voi johtaa testikattavuuden heikkenemiseen, jos testejä poistetaan liikaa. Ei huomioi uusia bugeja.

Taulukko 1. Regressiomenetelmien hyödyt ja haitat. [14]

Testitapausten priorisointi (*TCP - Test Case Prioritization*) on menetelmä, jonka avulla testit suoritetaan järjestyksessä, joka maksimoi virheiden löytymisen mahdollisimman varhaisessa vaiheessa. Näin voidaan minimoida ajanhukkaa ja resurssien kulutusta. Perinteisten TCP-menetelmien etuna on ollut se, että ne ovat selkeitä ja helppoja toteuttaa, mutta niistä puuttuu koneoppimiseen perustuvien lähestymistapojen joustavuus. [11] [15][16][17] [18]

Testitapausten valinta (TCS - Test Case Selection) on menetelmä, jota käytetään regressiotestauksessa päättämään, mitkä testit suoritetaan tietyssä testausistunnossa. Menetelmä valitsee testausistunnossa käytettävät testit arvioimalla sovelluksen nykytilaa, tehtyjä muutoksia ja testauksen tavoitteita. Testisarjan loput testit jätetään ajamatta. TCS auttaa vähentämään testauksen kustannuksia ja kestoja valitsemalla vain oleelliset testit, mistä on suuri hyöty varsinkin suurissa ohjelmistoprojekteissa. Menetelmän haasteena on kuitenkin se, että rajoitetun testijoukon valitseminen voi johtaa virheiden jäämiseen huomaamatta, jos kaikki riippuvuudet eivät tule huomioituiksi. [16][17]

Testisarjan minimointi (TSM - Test Suite Minimization) on menetelmä, joka parantaa ohjelmistotestauksen tehokkuutta poistamalla tarpeettomat testitapaukset säilyttäen kuitenkin vikojen havaitsemiskyvyn. TSM on hyvin samankaltainen TCS-menetelmän

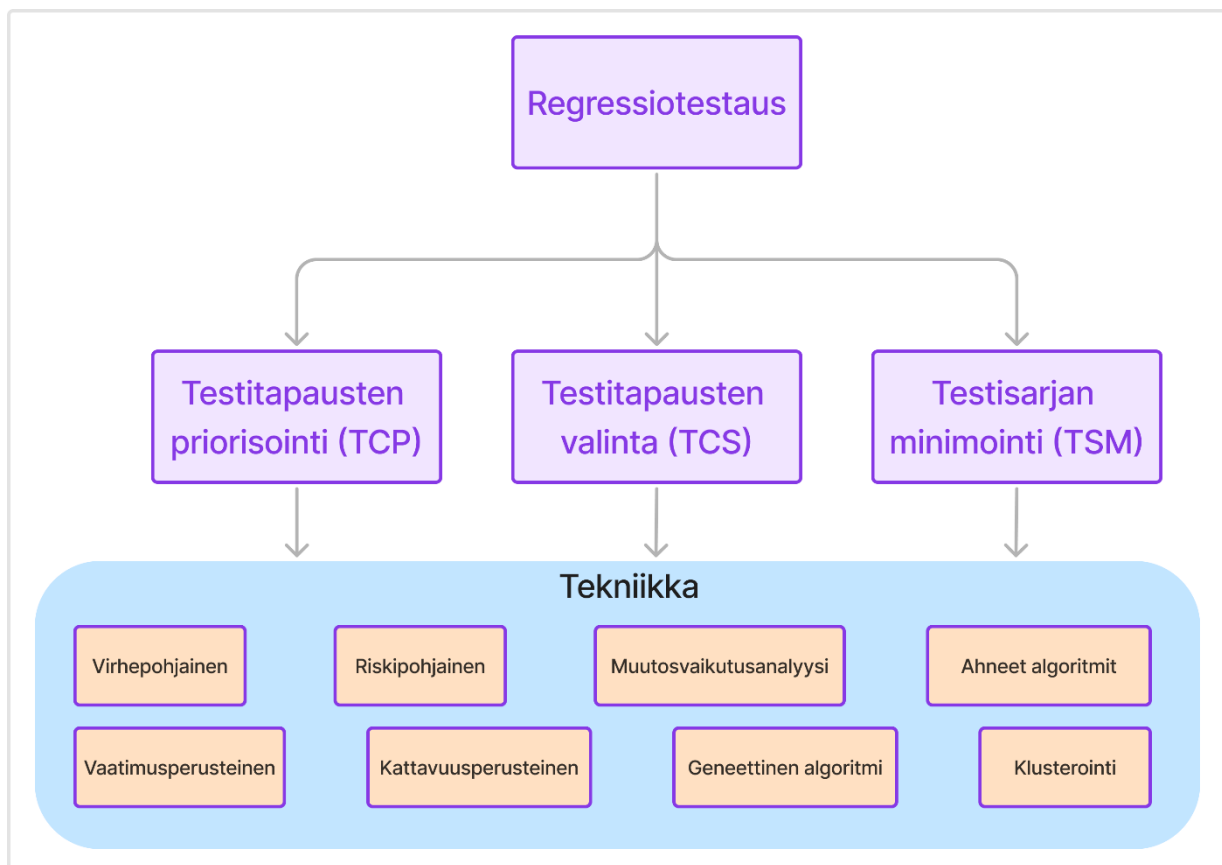
kanssa. Erona on käytännössä vain se, että tässä tapauksessa testisarjasta poistetaan kokonaan tarpeettomat testitapaukset. [16][17]

Taulukko 2 on kerätty joitain yleisimpiä regressiotestauksessa TCP-, TCS- ja TSM-menetelmiin käytettyjä tekniikoita. Suurinta osaa tekniikoista voidaan hyödyntää kaikkiin menetelmiin; TCP:n tapauksessa tekniikoita käytetään testisarjan järjestyksen optimointiin, kun taas TCS- ja TSM-menetelmissä niillä valitaan/rajataan testisarjaa. **Kuva 2** on pyritty havainnollistamaan optimointimenetelmiä ja tekniikoita.

Tekniikka	Kuvaus
Virhepohjainen <i>Black box</i>	Priorisoi/valitsee testitapaukset aiemman vikahistorian perusteella, keskittyen vikatiheisiin moduuleihin. [17]
Riskipohjainen <i>Black box</i>	Testaus painottuu korkean riskin järjestelmäkomponentteihin, joissa virheiden vaikutus ja todennäköisyys ovat suuria. [17]
Vaatusperusteinen <i>Black box</i>	Valitsee testitapaukset niiden vaatimusten perusteella, jotka ovat kriittisiä järjestelmän toiminnalle tai kattavuudelle. [17]
Kattavuusperusteinen <i>White box</i>	Testitapaukset priorisoidaan/valitaan kattavuuden perusteella (esim. koodikattavuus, vaatimuskattavuus, polkukattavuus). [17][18]
Ohjelman viipalointi <i>White box</i>	Luo ohjelmasta abstraktion, josta on poistettu testin kannalta epäoleellinen tieto ja rakenteet. Esimerkiksi, jos halutaan tarkastella, miten ohjelma voi virheellisesti muuttaa muuttujan x arvon, voimme poistaa kaikki ohjelman lauseet, jotka eivät vaikuta x:ään eivätkä ole x:n arvon vaikutuksessa. [19]
Klusterointi <i>White box/ Black box</i>	Klusterointimenetelmät ryhmittävät samankaltaiset testitapaukset niiden kattavuuden tai käyttäytymisen perusteella. Ajatuksena on valita edustava testitapaus kustakin klusterista, mikä vähentää toistoa. [20]
Muutosvaikutusanalyysi <i>White box</i>	Analysoi koodimuutosten vaikutuksen ja valitsee vain testit, jotka liittyvät niihin. [21][23]
Ahneet algoritmit <i>White box/Black box</i>	Valitsevat testit iteratiivisesti kattavuuden tai vianpaljastuspotentiaalın perusteella (TCP) tai valitsevat testit, jotka vähentävät testisarjan kokoa säilyttäen kattavuuden (TSM). [22]
Geneettinen algoritmi <i>White box/ Black box</i>	Käyttää evoluutioon perustuvia tekniikoita kehittääkseen testitapausten joukon useiden sukupolvien aikana. Käytetään testien priorisoinnin/valinnan optimointiin tiettyjen soveltuvuusperusteiden, kuten vikojen havaitsemisen maksimoimisen ja testitapausten määrän minimoimisen, perusteella. [23]

<p>Kustannustietoisuus-perusteinen (Cost-aware based) <i>White box/ Black box</i></p>	<p>Testit valitaan ja priorisoidaan suhteuttamalla testauksen hyötyjä ja kustannuksia. Tavoitteena on saavuttaa mahdollisimman hyvä kattavuus ja vikojen havaitseminen samalla minimoiden testauksen resurssikulut. Kustannustietoisuus on usein muiden testauksen optimointitekniikoiden, kuten ahneiden algoritmien ja geneettisten algoritmien, sisäänrakennettu ominaisuus. [24]</p>
<p>Bayes-verkkoon perustuva (Bayesian-network based) <i>White box/ Black box</i> <i>Bayes-verkon toimintaa selitetty luvussa 5.2.</i></p>	<p>Bayesin verkkoja käytetään mallintamaan ohjelmistokomponenttien ja vikojen välisiä todennäköisyyksiä. Tämä auttaa ennustamaan, mitkä testitapaukset todennäköisimmin havaitsevat vikoja, jolloin testaus voidaan kohdistaa riskialttiisiin alueisiin. Tekniikka tukee erityisesti riskipohjaista testausta, mutta sitä voidaan hyödyntää myös muutosvaikutusanalyysissä arvioimalla, mitkä koodimuutokset vaikuttavat järjestelmän toimintaan eniten. [25]</p>

Taulukko 2. Yleisiä optimointimentelmien hyödyntämiä tekniikoita



Kuva 2. Regressiotestauksen optimointimenetelmät ja niihin liittyvät tekniikat.

5 Koneoppiminen regressiotestauksessa

Jatkuva integraatio vaatii nopeaa palautetta kehittäjille, joten hidas testaus voi olla haitallista. Ohjelmiston kehittyessä testijoukot kasvavat ja niiden suoritus voi kestää liian kauan. Koneoppiminen voi nopeuttaa tätä prosessia analysoimalla testituloksia ja tunnistamalla testitapaukset, jotka ovat todennäköisimmin kriittisiä muutosten kannalta.

Koneoppimismallit voivat priorisoida testitapaukset niiden tärkeyden ja virheiden löytymistodennäköisyyden perusteella. Tämä vähentää testien suorittamiseen kuluvaa aikaa ja varmistaa, että tärkeimmät testit suoritetaan ensin. Lisäksi koneoppiminen voi auttaa tunnistamaan tarpeettomia testejä ja ehdottaa testitapausten yhdistämistä tai poistamista, mikä tehostaa testausprosessia entisestään.

Koneoppimista voidaan myös käyttää testitapausten automaattiseen luomiseen ja ylläpitoon. Algoritmit voivat oppia sovelluksen käyttäytymismalleja ja luoda testitapauksia, jotka kattavat laajasti erilaisia käyttötilanteita. Tämä parantaa testikattavuutta ja auttaa löytämään vaikeasti löydettäviä virheitä.

5.1 Perinteisten koneoppimismallien hyödyntäminen testitapausten valinnassa

Ahmadin ja muiden (2024) tutkimuksessa käytettiin ohjattua oppimista sekä neuroverkkoja laajassa teollisessa aineistossa testitapausten valitsemiseksi. Valintaan vaikuttivat lähdekoodimuutokset, commit-viestit ja muutetut tiedostopolut, joita käytettiin selittäjinä. Tutkimuksessa analysoitiin kunkin selittäjän sekä niiden yhdistelmien vaikutusta testitapausten valintaan. Aineisto koostui yhteensä 15 miljoonasta yksittäisestä testiajosta. Mallien koulutuksen ja testauksen lisäksi niiden suorituskykyä arvioitiin kuukauden ajan. Tavoitteena oli laskea ajansäästö ja testiajojen määrän vähentyminen verrattuna siihen, mitä kehittäjät olivat todellisuudessa suorittaneet. Tutkimuksen tulokset osoittivat, kuinka yhdistettyjen selittäjien käyttö paransi mallien tarkkuutta noin 97 %. Tämän seurauksena koneoppimismallit vähensivät testiajojen määrää 88–90 % ja testausaikaa 44–74 % testauksen eri vaiheissa. [26]

Sawantin (2024) tutkimuksessa tutkittiin eri koneoppimismalleja yhdistelevien yhdistelmämallien tehokkuutta testitapausten priorisointiin (TCP). Tutkimuksen mukaan yhdistelmämallit, parantavat TCP:n tehokkuutta merkittävästi. Ne tarjoavat skaalautuvia ja vahvoja ratkaisuja regressiotestauksen nykyaikaisiin haasteisiin, parantaen virheiden havaitsemista ja resurssien käyttöä. Tutkimuksessa suositeltiin, että tulevaisuudessa tulisi keskittyä datan keruun ja ominaisuuksien valinnan prosessien tarkentamiseen koneoppimis pohjaisen TCP:n suorituskyvyn parantamiseksi. [18]

Wangin (2012) tutkimuksessa pyrittiin ennustamaan miten ohjelman muutos vaikuttaa muihin ohjelman osiin. Tämä toteutettiin luomalla koneoppimiseen ja ohjelman viipalointiin perustuva testitapausten priorisointikehys. Ohjelma jaetaan lohkoihin, jotka ovat vaikuttaneet ohjelman muutoksiin tai joihin muutokset voivat vaikuttaa. Lohkot

priorisoidaan sen mukaan, kuinka usein ne esiintyvät testisuorituksessa. Tätä tietoa käytetään neuroverkon kouluttamiseen, jolloin se oppii priorisoimaan testitapaukset automaattisesti. Kokeet osoittivat, että menetelmällä saavutettiin hyvä suorituskyky testitapausten priorisoinnissa ja ohjelman virheet saatiin havaittua jo pienellä määrällä testitapauksia. [19]

5.2 Bayesin verkkoihin perustuva priorisointi

Bayesin verkot ovat todennäköisyyspohjaisia graafisia malleja, joissa solmut edustavat satunnaismuuttujia ja kaaret kuvaavat todennäköisyysriippuvuuksia. Verkot mahdollistavat todennäköisyyspohjaisen päättelyn, jonka avulla voidaan ennustaa testitapausten onnistumisen todennäköisyyttä havaittujen tietojen perusteella. Koneoppimisen avulla Bayesin verkkoja voidaan jatkuvasti parantaa, mikä tekee niistä tehokkaita työkaluja ohjelmistotestauksessa.

Mirarabin & Tahvildarin (2007) tutkimuksessa ehdotetaan uutta testiaineiston priorisointimenetelmää, joka yhdistää useita tietolähteitä yhdeksi kokonaisuudeksi. Lähestymistapa hyödyntää Bayesin verkkoja, joiden avulla analysoidaan ohjelmistomuutoksia, virhealttiuden indikaattoreita ja testikattavuustietoja. Menetelmän tehokkuutta testattiin kahdeksalla peräkkäisellä Java-sovellusversiolla. Tulokset osoittivat sen ylittävän perinteiset tekniikat erityisesti silloin, kun lähdekoodissa on riittävästi virheitä. Tutkimus osoitti, että lähestymistapa saavutti paremman virheiden havaitsemisasteen kuin pelkkään koodikattavuuteen perustuvat menetelmät. [27]

Edellisessä tutkimuksessa ei kuitenkaan otettu huomioon testitapausten samankaltaisuutta, mikä voi johtaa tarpeettomiin toistoihin. *Zhao ja muut (2015)* toteuttivat tutkimuksen, jossa pyrittiin lieventämään tätä ongelmaa luomalla hybridipriorisointitekniikka regressiotestaukseen. Tutkimuksessa yhdistettiin koodikattavuuteen perustuva klusterointimenetelmä ja Bayesin verkkoihin perustuva testitapausten priorisointimenetelmä, jolloin saatiin vähennettyä samankaltaisten testitapausten aiheuttamaa päällekkäisyyttä. Menetelmän tehokkuutta arvioitiin kahdella Java-projektilla, joissa oli mutaatiivirheitä sekä yhdellä Java-projektilla, jossa virheet oli lisätty käsin. Sen suorituskykyä verrattiin useisiin testausmenetelmiin ja tulokset osoittivat, että ehdotettu menetelmä on lupaava. [28]

5.3 Neuroverkkopohjainen priorisointi

Neuroverkot ovat yhä keskeisemmässä roolissa regressiotestauksen testitapausten optimoinnissa nykypäivänä. Tässä luvussa esitellään kaksi tutkimusta, joissa hyödynnetään neuroverkkoja regressiotestauksessa. Ensimmäinen tutkimus on hieman vanhempi, mutta se havainnollistaa hyvin koneoppimisen ja neuroverkkojen soveltamista tällaisiin tehtäviin. Toinen tutkimus vuodelta 2022 hyödyntää moderneja neuroverkkoratkaisuja ja erilaisia optimointimenetelmiä.

Spiekerin ja muiden (2017) tutkimuksessa esitellään vahvistusoppimiseen ja neuroverkkoihin perustuva menetelmä, joka oppii tunnistamaan virhealttiita testitapauksia aiemman suoritushistorian perusteella. Testitapaukset järjestetään suoritusajan, aiempien vikojen ja suorituskertojen mukaan. Menetelmän etuna on sen mukautuvuus muuttuviin testausympäristöihin, kuten uusien testien lisäämiseen ja vanhojen poistamiseen. Lisäksi ratkaisu on kevyt, mallivapaa ja kieliriippumaton, koska se käyttää ainoastaan testihistoriaa eikä vaadi lähdekoodin analysointia. Arviointitulokset osoittavat, että menetelmä oppii nopeasti ja mukautuu tehokkaasti kolmessa teollisessa tapaustutkimuksessa. Menetelmä löysi toimivan priorisointistrategian, joka suoriutui yhtä hyvin kuin perinteiset menetelmät noin 60 oppimisvaiheen jälkeen, ilman ennakkokoulutusta testitapausten priorisoinnissa. Menetelmää voidaan jatkokehittää lisäämällä testitapausten metatietoja ja hyödyntämällä syväoppimista tarkempien priorisointimallien rakentamiseen. [29]

Raameshin ja muiden (2022) tutkimuksessa esiteltiin modernia neuroverkkotekniikkaa hyödyntävä malli, joka minimoi testitapaukset ennen regressiotestausta. Ehdotetun menetelmän tarkkuus ja kattavuus ovat 98,5 % ja 99 %, mikä ylittää nykyiset tekniikat. Testitapausten minimoinnin ansiosta suoritus aika (19,14 sekuntia) on lyhyempi kuin muiden huipputekniikoiden. Tulokset osoittavat, että malli on tehokas virheiden tunnistamisessa Java-, Python- ja PHP-lähdekoodissa. Tulevaisuudessa mallia aiotaan laajentaa käytettäväksi muissa avoimen lähdekoodin järjestelmissä, jotka hyödyntävät erilaisia testinluontityökaluja. [30]

5.4 Klusterointipohjainen priorisointi

Klusterointipohjaisen testitapausten priorisointimenetelmän tarkoituksena on hyödyntää testin aikana saatavaa virhetietoa priorisoinnin tehostamiseksi, yhdistäen klusterointianalyysin ja adaptiiviset algoritmit. Ennen testin suoritusta testitapaukset jaetaan ryhmiin klusterointikriteerien avulla. Nämä klusterit määrittelevät, miten testitapaukset järjestetään suorituksen aikana. Kun testitapaus havaitsee virheen testin aikana, sen jälkeiset testitapaukset priorisoidaan uudelleen adaptiivisesti ottaen huomioon virheen havaitseminen ja aiemmin suoritettut testit. Menetelmää on testattu useammalla olio-ohjelmalla ja tulokset osoittavat, että klusteripohjainen adaptiivinen priorisointi ylittää perinteiset TCP-menetelmät virheiden havaitsemisessa ja testin suoritus aikojen optimoinnissa. [31][32]

Wenhao ja muut (2017) lisäsivät klusterointipohjaiseen testitapausten priorisointiin ajoitusalgoritmeja, jotka järjestivät testitapaukset suoritettavaksi paitsi niiden virheiden havaitsemiskyvyn myös odotusajan perusteella. Tämä parantaa testien tehokkuutta erityisesti tilanteissa, joissa on tärkeää havaita mahdollisimman monta erilaista virhetyyppiä mahdollisimman varhain. Ajoitusalgoritmi hyödynsi dynaamista prioriteettien säätöä, jossa jo suoritettujen testien tulokset vaikuttavat jäljellä olevien testien järjestykseen, sekä aikakatkaisumekanismia, joka varmistaa pitkään odottaneiden testien suorituksen riippumatta niiden arvioidusta prioriteetista. Tällä tavoin vältetään yksittäisiin virhetyyppeihin

keskittyminen ja parannetaan kattavuutta. Menetelmän tehokkuus arvioitiin kokeellisesti 12 C-ohjelmalla, ja se osoittautui paremmaksi, tai vähintään yhtä tehokkaaksi, kuin perinteiset menetelmät, kuten ahneet algoritmit ja muut klusterointiin perustuvat lähestymistavat, etenkin virheiden nopeassa havaitsemisessa. [33]

Hieman erilaisen lähestymiskulman klusterointiin tarjoavat *Tamagnan ja muut* (2023), jotka tutkimuksessaan hyödynsivät klusterointia regressiotestauksen automatisointiin ja optimointiin käyttäjälökien pohjalta. Web-sovellusten todellisten käyttöpolkujen perusteella muodostettiin klustereita, joista valittiin edustavat testitapaukset kattavan ja resurssitehokkaan testiaineiston luomiseksi. Klustereiden laadun arviointiin kehitettiin tilastollinen kattavuusmittari, joka perustuu käyttäytymismallien tunnistamiseen (pattern mining) ja tarjoaa aiempaa luotettavamman keinon arvioida testien edustavuutta suhteessa todelliseen käyttöön. Menetelmää sovellettiin neljään eri web-sovellukseen, ja sen avulla luodut testiaineistot ylittivät käsin laadittujen testien kattavuuden. [34]

5.5 Koneoppimista hyödyntäviä testaustyökaluja

Saatavilla on useita tekoälypohjaisia työkaluja ja kehyksiä, jotka tehostavat testausprosessia, kuten:

- **Testim.io**
Käyttää tekoälyä automatisoitujen testitapausten luomiseen ja ylläpitämiseen. Sen koneoppimisalgoritmit tunnistavat muutoksia sovelluksessa ja muokkaavat testitapauksia niiden mukaan. [35]
- **Stryker**
Käyttää tekoälyä koodin analysointiin ja testitapausten luomiseen. Tekee muutoksia, eli mutaatioita, koodiin ja tarkistaa, voivatko olemassa olevat testit havaita nämä muutokset, parantaen testitapausten tehokkuutta. [35]
- **Mabl**
Hyödyntää koneoppimista testiautomaation parantamisessa. Sisältää itsekorjautuvia testejä, jotka mukautuvat sovelluksen muutoksiin ja tarjoavat näkemyksiä testituloksista, mahdollistaen tehokkaamman testitapausten luomisen. [35]
- **Test.ai**
Käyttää koneoppimista automaattisten testitapausten luomiseen ja toiminnalliseen testaukseen. Se mukautuu sovelluksen käyttöliittymän muutoksiin, tarjoten automatisoitua testausta. [35]
- **Functionize**
Priorisoi testitapauksia vaatimuspohjaisesti koneoppimista ja luonnollisen kielen käsittelyä hyödyntäen. Tarjoaa työkaluja dynaamiseen testien luomiseen ja hallintaan. Sovelluksen muutoksiin mukautuminen on pitkälti automatisoitu. [35]
- **Appvance IQ**

Tekoäly luo testitapauksia käyttäjätoimintojen ja sovelluksen käyttäytymisen perusteella. Tarjoaa älykkään testien luomisen ja hallinnan, parantaen testikattavuutta ja tehokkuutta. [35]

- **Parasoft Selenic**

Tekoälyavusteinen UI-testauksen työkalu, joka tekee Selenium-testauksesta älykkäämpää ja luotettavampaa. Se integroituu suoraan olemassa oleviin Selenium-testeihin ilman tarvetta siirtyä uuteen alustaan, tarjoten itsekorjautuvia testejä ja automatisoituja korjaussuosituksia. Selenic nopeuttaa testien ylläpitoa ja vähentää testauksen pullonkauloja. [36]

Selenium-testaus on verkkosovellusten automaattista testausta, jossa simuloidaan käyttäjän toimintaa selaimessa eri ympäristöissä.

- **Katalon studio**

Hyödyntää tekoälyä testiautomaation optimointiin ja ylläpidon helpottamiseen. Sen AI-pohjaisiin ominaisuuksiin kuuluu älykäs elementtitunnistus, testiskenaarioiden automaattinen luonti ja testitapausten priorisointi. [37]

6 Pohdintaa

Tutkimuksen tavoitteena oli selvittää, miten koneoppimistekniikoita voidaan hyödyntää regressiotestauksessa ja testien optimoinnissa. Kirjallisuuskatsauksen perusteella selvisi, että menetelmiä on useita, mutta käytännön sovelluksista on vaikea saada kattavaa kuvaa. Tämä johtuu osittain siitä, että monet julkaisut ovat varsin uusia, eikä niiden esittämiä menetelmiä ole välttämättä vielä laajasti otettu käyttöön. Lisäksi testaustyökalujen käyttämien koneoppimistekniikoiden tarkka selvittäminen on haastavaa, sillä niiden toiminnasta on saatavilla vain rajallisesti julkista tietoa.

Koneoppimisen laajuus vaikutti merkittävästi tutkimuksen rajaukseen. Vaikka monia potentiaalisia optimointimenetelmiä olisi voinut käsitellä, niiden kattava tarkastelu olisi tehnyt tutkielmasta liian laajan. Esimerkiksi syväoppimiseen perustuvat menetelmät jätettiin tarkastelun ulkopuolelle, jotta tutkimuksen rajausta pysyisi järkevänä. Tämä osoittaa, että aihepiirin laajuus vaatii lisätutkimusta ja esimerkiksi syväoppimisen hyödyntäminen regressiotestauksessa voisi olla lupaava jatkotutkimuksen kohde.

Tutkielman myötä vahvistui käsitys siitä, että koneoppimista voidaan hyödyntää regressiotestauksessa erityisesti testien optimoinnissa ja virheiden tunnistamisessa. Kirjallisuudessa esitellyt menetelmät, kuten testitapausten priorisointi ja virheanalyysi, tarjoavat lupaavia keinoja testausprosessin tehostamiseen. Kuitenkin konkreettisten tulosten yleistettävyyttä jää osittain epävarmaksi, sillä suurta osaa tutkimuksista ei toteutettu todellisessa testausympäristössä.

Vaikka koneoppiminen voi tehostaa regressiotestausta merkittävästi, sen käyttöönotto ei ole ilmaista. Se vaatii panostuksia dataan, mallien kehittämiseen ja laskentatehoon. Suurien järjestelmien regressiotestauksessa, pitkällä aikavälillä, hyödyt voivat kuitenkin olla huomattavia. Myös datan laatu on iso tekijä koneoppimismallin onnistumisessa; huonolla datalla saadaan huono malli.

Tutkimus tarjosi myös henkilökohtaisesti uusia näkökulmia ohjelmistotestaukseen, joka ei aiemmin ollut itselleni erityisen tuttu aihealue. Valitsin aiheen, koska minua kiinnosti tekoäly ja koneoppiminen, mutta halusin samalla tutustua sen sovelluksiin uusilla alueilla. Tämä osoittautui hyödylliseksi, sillä koneoppimisen hyödyntäminen ohjelmistokehityksessä vaikuttaa olevan kasvava tutkimusalue, jossa on vielä paljon kehitettävää.

Jatkotutkimuksen kannalta olisi hyödyllistä tarkastella laajemmin, mitkä koneoppimismenetelmät ovat jo käytössä teollisuudessa ja miten ne ovat parantaneet testausprosessien tehokkuutta. Lisäksi olisi kiinnostavaa vertailla eri menetelmien suorituskykyä käytännön sovelluksissa, jotta saataisiin selville, mitkä ratkaisut tarjoavat parhaat hyödyt regressiotestauksen optimointiin.

7 Yhteenveto

Koneoppimisessa tietokone opetetaan löytämään ja hyödyntämään datasta toistuvia kuvioita, mikä mahdollistaa päätöksenteon ilman ennalta määriteltyjä sääntöjä. Se perustuu algoritmeihin, jotka mukautuvat ja optimoituvat koulutusdatan avulla. Koneoppiminen jakautuu kolmeen päälinjaan: ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen. Ohjatussa oppimisessa malli opetetaan yhdistämään selittäjien vaikutus vastemuuttujaan, kun taas ohjaamattomassa oppimisessa tavoitteena on löytää aineistosta rakenteita ilman etukäteen määriteltyjä vastemuuttujia. Vahvistusoppimisessa tavoitteena on oppia toimintapolitiikka, joka johtaa haluttuun lopputulokseen.

Neuroverkot ovat laskennallisia malleja, jotka jäljittelevät ihmisaivojen toimintaa. Ne koostuvat kerroksista, joissa on toisiinsa yhdistettyjä neuroneita. Neuroverkot ovat osoittautuneet erittäin tehokkaiksi monilla alueilla, kuten kasvojentunnistuksessa ja itseajavissa autoissa.

Regressiotestaus on ohjelmistotestauksen osa-alue, jossa tutkitaan, kuinka ohjelman muutokset vaikuttavat aiemmin testattuihin osiin. Tämä on tärkeää, koska ohjelmiston koon ja monimutkaisuuden kasvaessa testisarjaan kertyy yhä enemmän testejä, mikä pidentää niiden suorittamisaikaa. Koneoppiminen voi nopeuttaa tätä prosessia analysoimalla testituloksia ja priorisoiden testitapauksia niiden tärkeyden ja virheiden löytymistodennäköisyyden perusteella.

Koneoppimismallit voivat auttaa tunnistamaan tarpeettomia testejä ja ehdottaa testitapausten yhdistämistä tai poistamista, mikä tehostaa testausprosessia. Lisäksi koneoppimista voidaan käyttää testitapausten automaattiseen luomiseen ja ylläpitoon, parantaen testikattavuutta ja auttaen löytämään vaikeasti löydettäviä virheitä.

Tutkimuksissa on osoitettu, että koneoppimismallit voivat vähentää testiajojen määrää ja testausaikaa merkittävästi. Esimerkiksi yhdistettyjen selittäjien käyttö paransi mallien tarkkuutta noin 97 %, testiajojen määrä väheni 88–90 % sekä testausaika 44–74 % testauksen eri vaiheissa. Bayesin verkot ovat myös osoittautuneet tehokkaiksi työkaluiksi testitapausten priorisoinnissa, ja klusteripohjaiset menetelmät ovat parantaneet virheiden havaitsemista ja testin suoritusajojen optimointia.

Yhteenvetona voidaan sanoa, että koneoppiminen tarjoaa merkittäviä hyötyjä ohjelmistotestauksessa, erityisesti regressiotestauksessa, jossa se voi nopeuttaa prosessia ja parantaa testikattavuutta. Koneoppimisen hyödyntäminen testausprosessissa on tulevaisuudessa tärkeää, kun ohjelmistojen kehitys ja ylläpito jatkuvat nopealla tahdilla.

Lähdeluettelo

- [1] Alpaydin, E. (2020) Introduction to Machine Learning, Fourth Edition, *MIT Press*. <https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=6676810>.
- [2] Kämäräinen, J. *Koneoppimisen perusteet*. Otatieto, 2023.
- [3] Kolodiazhnyi, K. *Hands-on Machine Learning with C++ : Build, Train, and Deploy End-to-End Machine Learning and Deep Learning Pipelines*. 1st edition, Packt Publishing, 2020.
- [4] Ghani, F. A., Rivaie, M., Yusoff M., Puteh, M. (2022). A Review of Artificial Neural Network Applications in Variants of Optimization Algorithms, *2022 International Visualization, Informatics and Technology Conference (IVIT)*, Kuala Lumpur, Malaysia, pp. 115-123, <https://doi.org/10.1109/IVIT55443.2022.10033339>.
- [5] Fujiyoshi, H., Hirakawa, T., Yamashita, T. (2019). Deep Learning-Based Image Recognition for Autonomous Driving. *IATSS Research*, vol. 43, no. 4, pp. 244–52, <https://doi.org/10.1016/j.iatssr.2019.11.008>.
- [6] Shastry, K., Shastry, A. (2023). An Integrated Deep Learning and Natural Language Processing Approach for Continuous Remote Monitoring in Digital Health.” *Decision Analytics Journal*, vol. 8, pp. 100301-, <https://doi.org/10.1016/j.dajour.2023.100301>.
- [7] Chakraborty C., Bhattacharya, M., Pal, S., Lee, S. (2024). From Machine Learning to Deep Learning: Advances of the Recent Data-Driven Paradigm Shift in Medicine and Healthcare. *Current Research in Biotechnology*, vol. 7, 2024, pp. 100164-, <https://doi.org/10.1016/j.crbiot.2023.100164>.
- [8] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676), 354-359,359A-359L. <https://doi.org/10.1038/nature24270>.
- [9] Bertram, T., Furnkranz, J., Muller, M. (2024). Neural Network-Based Information Set Weighting for Playing Reconnaissance Blind Chess.” *IEEE Transactions on Games*, vol. 16, no. 4, pp. 960–70, <https://doi.org/10.1109/TG.2024.3425803>.
- [10] Chopra, R. (2018). Software Testing : A Self-Teaching Introduction. *Mercury Learning & Information*. <https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=30331787>.
- [11] Ansari, A., Khan, A., Khan, A., Mukadam K. (2106). Optimized Regression Test Using Test Case Prioritization. *Procedia Computer Science*, Vol.79, p.152-160. <https://doi.org/10.1016/j.procs.2016.03.020>.
- [12] Carlson, R., Hyunsook, D., Denton, A. (2011). A clustering approach to improving test case prioritization: An industrial case study. *27th IEEE International Conference on Software Maintenance (ICSM)*, 382-391. <https://www.doi.org/10.1109/ICSM.2011.6080805>.
- [13] Yoo, S., Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Softw. Test. Verif. Reliab.*, 22: 67-120. <https://doi.org/10.1002/stvr.430>.

- [14] Kaur, K., Khatri, S. Datta, R. (2014). Analysis of Various Testing Techniques. *International Journal of System Assurance Engineering and Management*, vol. 5, no. 3, pp. 276–90, <https://doi.org/10.1007/s13198-013-0157-6>.
- [15] Khan, M.A, Azim, A., Liscano, R., Smith, K., Chang, Y. Seferi G., Tauseef Q. (2024). An End-to-End Test Case Prioritization Framework using Optimized Machine Learning Models. *Proceedings - 2024 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2024*, 1-8. <https://www.doi.org/10.1109/ICSTW60967.2024.00014>.
- [16] Siddique, M A. (2023). The State-of-the-art Hybrid Approaches in Regression Testing. *International Journal of Industrial Engineering & Production Research*, vol. 34, no. 4, pp. 31–44. <http://ijiepr.iust.ac.ir/article-1-1835-en.html>.
- [17] Alkawaz, M. H., Silvarajoo A. (2019). A Survey on Test Case Prioritization and Optimization Techniques in Software Regression Testing, *IEEE 7th Conference on Systems, Process and Control (ICSPC), 2019*, pp. 59-64, <https://doi.org/10.1109/ICSPC47137.2019.9068003>.
- [18] Sawant, P.D. (2024). Test Case Prioritization for Regression Testing Using Machine Learning. *Proceedings - 6th IEEE International Conference on Artificial Intelligence Testing, AITest 2024*, 152-153. <https://www.doi.org/10.1109/AITest62860.2024.00027>.
- [19] Wang, F., Yang, S., Yang, Y., Li, T. (2012). Regression Testing Based on Neural Networks and Program Slicing Techniques. *Practical Applications of Intelligent Systems*, vol. 124, Springer Berlin Heidelberg, pp. 409–18, https://doi.org/10.1007/978-3-642-25658-5_50.
- [20] Yoo, S., Harman M., Tonella P., Susi A. (2009). Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. *In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). Association for Computing Machinery*, 201–212. <https://doi.org/10.1145/1572272.1572296>.
- [21] Ahmed, F.S., Majeed, A., Khan, T.A., Bhatti, S.N. (2022). Value-based cost-cognizant test case prioritization for regression testing. *PLoS ONE 17(5)*: e0264972. <https://doi.org/10.1371/journal.pone.0264972>.
- [22] Cormen T., Leiserson C., Rivest R., Stein C. (2022). *Introduction to Algorithms*. 4th ed., MIT Press.
- [23] Alrawashdeh, T. A., ElQirem, F., Althunibat, A., Alsoub, R. (2021). A Prioritization Approach for Regression Test Cases Based on a Revised Genetic Algorithm. *Information Technology and Control*, 50(3), 443-457. <https://doi.org/10.5755/j01.itc.50.3.27662>.
- [24] Ahmed, F.S., Majeed, A., Khan, T.A., Bhatti, S.N. (2022). Value-based cost-cognizant test case prioritization for regression testing. *PLoS ONE 17(5)*: e0264972. <https://doi.org/10.1371/journal.pone.0264972>.
- [25] Fang Z., Sun H. (2010). A Software Regression Testing Strategy Based on Bayesian Network. *International Conference on Computational Intelligence and Software Engineering*, IEEE, 2010, pp. 1–4, <https://doi.org/10.1109/CISE.2010.5676806>.

- [26] Ahmad, A., Rentas, D., Hasselqvist, D., Sandberg, P., Sandahl, K., Vulgarakis, A. (2024). Test Case Selection in Continuous Regression Testing Using Machine Learning: An Industrial Case Study. *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE pp. 33–38, <https://doi.org/10.1109/COMP-SAC61105.2024.00015>.
- [27] Mirarab, S., Tahvildari, L. (2007). A Prioritization Approach for Software Test Cases Based on Bayesian Networks. In: Dwyer, M.B., Lopes, A. (eds) *Fundamental Approaches to Software Engineering. FASE 2007. Lecture Notes in Computer Science*, vol 4422. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-71289-3_22
- [28] Zhao, X., Wang, Z., Fan, X., Wang, Z. (2015). A Clustering-Bayesian Network Based Approach for Test Case Prioritization. *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 3, IEEE, pp. 542–47, <https://doi.org/10.1109/COMP-SAC.2015.154>.
- [29] Spieker H., Gotlieb A., Marijan D., Mossige, M. (2017). Reinforcement learning for automatic test case prioritization and selection in continuous integration. *In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*, pp. 12–22, <https://doi.org/10.1145/3092703.3092709>.
- [30] Raamesh, L., Jothi, S., Radhika, S. (2022). Test Case Minimization and Prioritization for Regression Testing Using SBLA-Based Adaboost Convolutional Neural Network. *The Journal of Supercomputing*, vol. 78, no. 16, pp. 18379–403, <https://doi.org/10.1007/s11227-022-04540-1>
- [31] Wang, X., Zhang, S. (2024). Cluster-Based Adaptive Test Case Prioritization. *Information and Software Technology*, vol. 165, 2024, pp. 107339, <https://doi.org/10.1016/j.infsof.2023.107339>.
- [32] Chen, J., Zhu, L., Chen, T.Y., Towey, D., Kuo, F., Huang R., Guo, Y. (2018). Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering, *Journal of Systems and Software*, vol. 135, pp. 107-125, <https://doi.org/10.1016/j.jss.2017.09.031>.
- [33] Wenhao F., Huiqun Y., Guisheng F., Xiang J. (2017) Coverage-Based Clustering and Scheduling Approach for Test Case Prioritization, *IEICE Transactions on Information and Systems*, 2017, Volume E100.D, Issue 6, Pages 1218-1230, <https://doi.org/10.1587/transinf.2016EDP7356>.
- [34] Tamagnan, F., Vernotte, A., Bouquet, F. and Legeard, B. (2024), Generation of Regression Tests From Logs With Clustering Guided by Usage Patterns. *Softw Test Verif Reliab*, 34: e1900. <https://doi.org/10.1002/stvr.1900>
- [35] Baqar, M., Khanda, R. (2024). The Future of Software Testing: AI-Powered Test Case Generation and Validation. <http://dx.doi.org/10.48550/arXiv.2409.05808>.
- [36] Parasoft Launches Parasoft Selenic, an AI-Powered UI Testing Solution for Selenium: Parasoft Selenic Empowers Organizations to Accelerate Software Delivery While Ensuring a Reliable Customer Experience. *PR Newswire*, PR Newswire Association LLC, 2019.
- [37] Katalon (2025). Verkkosivu. Saatavissa: (Viitattu 29.03.2025) <https://katalon.com/katalon-studio>.