OULUN AMMATTIKORKEAKOULU

Tomi Niemelä

POIKKEAMIEN TUNNISTAMINEN LOKITIEDOSTOISTA

Splunk-ohjelmiston hyödyntäminen poikkeamien tunnistamiseen

POIKKEAMIEN TUNNISTAMINEN LOKITIEDOSTOISTA

Splunk-ohjelmiston hyödyntäminen poikkeamien tunnistamiseen

Tomi Niemelä
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä(t): Tomi Niemelä
Opinnäytetyön nimi: Poikkeamien tunnistaminen lokitiedostoista: Splunk-ohjelmiston hyödyntäminen poikkeamien tunnistamiseen
Työn ohjaaja(t): Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: kevät 2024 Sivumäärä: 42

Toimeksiantaja valmistaa langattomia verkkoliikennelaitteita, joiden testaamisesta syntyy päivittäin tuhansia rivejä lokitiedostoja. Toimeksiantaja halusi tutkia koneoppimisen ja automaation mahdollisuuksia suoritusaikalokitiedostojen läpikäymiseen ja mahdollisten pullonkaulojen tunnistamiseen lokitiedostoista. Opinnäytetyön tavoitteena oli binäärimuotoisten suoritusaikalokitiedostojen kerääminen jatkuvasta integraatiosta ja niiden lähettäminen Splunk-ohjelmistolle indeksoitaviksi ja edelleen analysoitaviksi.

Opinnäytetyön tarkoituksena oli selvittää, miten binäärimuotoisista lokitiedostoista saadaan tunnistettua automaation tai koneoppimisen avulla komponenttien mahdollista poikkeavaa käytöstä ja sen mahdollista kestoa. Splunk-ohjelmistoa käytettiin lokitiedostojen läpikäymiseen, ja sen Machine Learning Toolkit -laajennusosaa hyödynnettiin koneoppismenetelmien käyttöönottamiseksi. Opinnäytetyön tietoperusta koostuu pääsääntöisesti englanninkielisistä julkaisuista ja artikkeleista aiheinaan koneoppiminen ja Splunk.

Poikkeamien tunnistamiseksi binäärimuotoiset suoritusaikalokitiedostot purettiin ja parsittiin Splunk-ohjelmiston ymmärtämään csv-tiedostomuotoon testiympäristössä. Splunk-ohjelmistoa ei kuitenkaan ollut käytössä osastolla, jolle työ tehtiin, joten Splunk päätettiin asentaa paikalliseen kehitysympäristöön poikkeamien tunnistamisen kokeilemista varten.

Splunk-ohjelmisto asetettiin seuraamaan tiedostopolkua, josta csv-muotoiset tiedostot indeksoitiin tapahtumiksi, jokaisesta testikerrasta erikseen. Splunk-ohjelmistolla onnistuttiin näistä indeksoiduista tapahtumista löytämään kuormitettuja prosessoriytimiä sekä esittämään keinoja poikkeamien tunnistamiseksi suoritusaikalokitiedostoista koneoppimista hyödyntäen. Mahdollisia poikkeamia löydettiin myös ilman koneoppimisen keinoja. Mahdollisten poikkeamien kestoa ei kuitenkaan onnistuttu tässä opinnäytetyössä laskemaan, sillä suoritusaikalokitiedostoja täytyisi muokata sisältämään enemmän yksilöivää tietoa kulloinkin käsittelyssä olevasta datapaketista. Ilman tätä tietoa Splunk-ohjelmistolla ei voitu yhdistää oleellisia tietoja keskenään ja näin laskea datapaketin käyttämää kokonaisaikaa prosessien ketjussa.

Splunk vaikutti erittäin tehokkaalta työkalulta lokitiedostojen läpikäymiseen. Koneoppimisen hyödyntäminen oli helppoa, mutta toisaalta kulutti virtuaalikoneen laskentatehoa merkittävästi. Splunk-ohjelmistossa oli mahdollista ohjelmoida haut automaattisiksi, ja asettaa hälytykset poikkeaville tuloksille, joten lokitiedostojen manuaalinen läpikäyminen vähentyisi työkalua käyttämällä merkittävästi.

Asiasanat: Algoritmit, koneoppiminen, poikkeamien tunnistaminen, Splunk

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author(s): Tomi Niemelä
Title of thesis: Identifying Anomalies from Log Files: Utilizing Splunk to Identify Anomalies
Supervisor(s): Jouni Juntunen
Term and year when the thesis was submitted: Spring 2024
Number of pages: 42

The contractor manufactures wireless network communication devices. Thousands of lines of logfiles are generated daily from testing these devices, and contractor wanted to investigate the possibilities of machine learning and automation for going through the logfiles and identifying possible bottlenecks in the system from the logfiles. The aim of this thesis was to collect runtime log files in binary format from continuous integration and send them to the Splunk for indexing and further analysis.

The subject of this thesis was to find out how to identify possible abnormal behavior of components and software and their possible duration by analyzing binary log files, that were result from testing network communication devices, with the help of automation or machine learning. Splunk was utilized to go through the log files, and its Machine Learning Toolkit plugin was used to implement machine learning methods.

To identify anomalies, runtime log files in binary format were extracted and parsed into csv file format understood by Splunk software in the test environment. These formatted csv files were sent for indexing. The Splunk software was able to find loaded processor cores from these indexed events. With the Splunk, it was possible to present ways to identify possible anomalies from log files using machine learning. Possible anomalies were also found without the use of machine learning. However, the duration of possible anomalies could not be calculated in this thesis, as the log files would have to be edited to contain more unique information about the data package.

Overall, Splunk seemed like a very powerful tool for going through the log files. In Splunk, it was possible to program the searches to be automatic, and to set alarms for abnormal results, so the manual reviewing of the log files would be significantly reduced by using this tool.

Keywords: Algorithms, anomaly detection, machine learning, Splunk

SISÄLLYS

1 JOHDANTO

Toimeksiantaja tälle opinnäytetyölle on Nokia Solutions and Networks, joka on kansainvälinen langattomaan verkkoliikennetekniikkaan keskittynyt yritys. Nokia Solutions and Networks on osa Nokia Oyj:tä, jonka liiketoiminnan keskiössä ovat verkkoinfrastruktuuri, teknologiakehitys ja lisensointi. Nokia työllistää maailmanlaajuisesti lähes 87 000 henkilöä. (1.)

Toimeksiantaja haluaa selvittää koneoppimisen ja automaation mahdollisuuksia langattomien verkkoliikennelaitteiden pullonkaulojen tunnistamiseksi. Pullonkauloilla tarkoitetaan yleisesti asioita tai tilanteita, jotka hidastavat järjestelmän tai kokonaisuuden toimintaa ja vaikuttavat hitaudellaan suorituskykyyn negatiivisesti. Tietotekniikassa pullonkauloilla tarkoitetaan ohjelmiston, verkon tai laitteiston yksittäisiä komponentteja, jotka hitaalla toiminnallaan vaikuttavat koko järjestelmän suorituskykyyn. (2.) Koneoppimisella taas tarkoitetaan datalla koulutettujen algoritmien hyödyntämistä konetuotetun datan läpikäymiseen (3). Oikein toimiessaan algoritmit pystyvät ennustamaan tuloksia ja luokittelemaan tietoja automaattisesti. Tässä opinnäytetyössä tullaan keskittymään langattomien verkkoliikennelaitteiden ohjelmiston ja laitteiston mahdollisten pullonkaulojen tunnistamiseen.

Langattomien verkkoliikennelaitteiden testaamisesta syntyy useita lokitiedostoja. Automatisoimalla lokitiedostojen analysointia henkilöresursseja voitaisiin vapauttaa varsinaiseen kehitystyöhön, mikä luonnollisesti toisi yritykselle säästöjä kustannuksiin. Tässä työssä käsiteltävät lokitiedostot sisältävät langattomien verkkoliikennelaitteiden komponenteilta kerättyjä suoritusaikoja binäärimuodossa.

Tämän opinnäytetyön tarkoitus on selvittää, miten binäärimuotoisista lokitiedostoista saadaan tunnistettua automaation tai koneoppimisen avulla komponenttien poikkeavaa käytöstä ja sen mahdollista kestoa. Varsinainen suorituskykyongelmien ratkaiseminen ja korjaaminen rajataan tämän työn ulkopuolelle, sillä tässä opinnäytetyössä keskitytään mahdollisten pullonkaulojen ja niiden tapahtuma-ajan sekä keston tunnistamiseen testisuorituksista kerätyistä lokitiedostoista. Lokitiedostojen analysoimiseen tullaan toimeksiantajan ehdotuksesta käyttämään Splunk Enterprise -nimistä ohjelmistoa, joka on tehokas työkalu datan indeksoimiseen ja indeksoidun datan valvomiseen sekä hakujen suorittamiseen.

Opinnäytetyön tavoitteena on luoda toimiva reitti jatkuvasta integraatiosta eli CI:stä (Continuous Integration) verkkoliikennelaitteiden testauksesta kerättyjen lokitiedostojen viemiseksi Splunk-palvelimelle. Reitti tullaan luomaan Python-ohjelmointikielellä, mikä on yhdenmukainen toimeksiantajalla käytössä olevan ratkaisun kanssa. Tavoitteena on myös purkaa binäärimuotoinen data Splunk-ohjelmiston ymmärtämään muotoon ennen palvelimelle tallentamista sekä kokeilla, miten Splunk-ohjelmistoa ja koneoppimista voidaan hyödyntää lokitiedostoista saatujen ytimien suoritusaikojen läpikäymiseen ja mahdollisten pullonkaulojen löytämiseen.

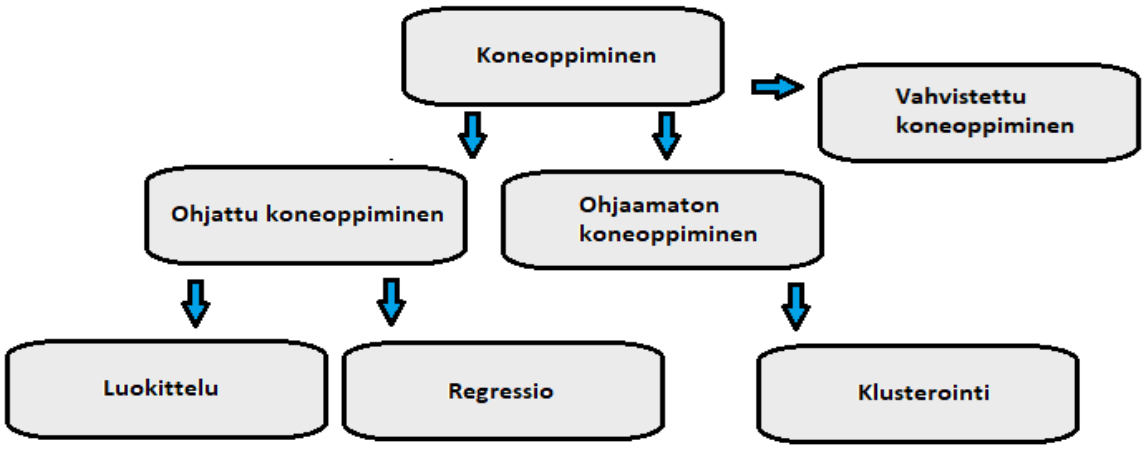
2 KONEOPPIMINEN

Arkikielessä käsitteet **koneoppiminen** (machine learning, ML) ja **tekoäly** (artificial intelligence, AI) menevät usein helposti sekaisin, sillä koneoppimisen käyttäminen on nykyään yleinen ratkaisu tekoälysovelluksissa: monet tekoälysovellukset perustuvat datasta oppimiseen. Merkityksellisesti nämä kaksi termiä tarkoittavat kuitenkin hieman eri asioita. Tekoäly viittaa yleisesti yritykseen luoda koneita, jotka kykenevät ihmisille tyypillisiin kognitiivisiin suorituksiin. Koneoppiminen taas on yksi tekoälyn osa-alue, joka käyttää datalla koulutettuja algoritmeja eli sääntöluetteloita itseoppivien mallien luomiseen. Mallit pystyvät ennustamaan tuloksia ja luokittelemaan tietoja itsenäisesti ja automaattisesti. (3.)

Koneoppimisessa algoritmeja opetetaan käyttämällä aikaisempia tietojoukkoja. Algoritmeja hyödynnetään uusien tietojoukkojen luokittelussa ja ennustamisessa. Algoritmien tehokkaan toiminnan varmistamiseksi niitä on kuitenkin yleensä koulutettava useita kertoja, kunnes algoritmeille on kertynyt riittävän tarkka luettelo oikean toiminnan varmistavista ohjeista. Riittävästi koulutetuista algoritmeista tulee lopulta koneoppimismalleja, jotka ovat tietyn tehtävän suorittamiseen koulutettuja algoritmeja. (3.)

2.1 Koneoppimisen jaottelu

Koneoppimista käytetään nykyään monissa tietotekniikan sovelluksissa, kuten esimerkiksi roskapostin suodatuksessa, kuvan ja äänen tunnistamisessa sekä suoratoistopalveluissa katseluehdotusten tekemiseen. Koneoppiminen jaetaan yleensä kolmeen luokkaan oppimisen tyylin perusteella: ohjattuun, ohjaamattomaan ja vahvistettuun koneoppimiseen (kuvio 1). (4, s. 12.)



KUVIO 1. Koneoppimisen jaottelu (4, s. 12)

Joissain lähteissä koneoppiminen on jaettu vielä neljänteenkin ryhmään: osittain ohjattu oppiminen (semisuperviced learning). Osittain ohjattu oppiminen hyödyntää sekä ohjaamatonta että ohjattua oppimista. (5.)

2.1.1 Ohjattu oppiminen

Ohjatussa oppimisessa (supervised learning) algoritmeja koulutetaan tunnettujen tietojoukkojen perusteella, jotka sisältävät oikeat ratkaisut ennusteen tekemiseen. Haluttu tulos on siis tiedossa jo ennen datan syöttämistä. Toisin sanoen algoritmeille syötetään koulutusdataa, joka kuvaa, miten tiedot tulisi tulkita. Uusi data tulkitaan koulutusdatan avulla syntyneen mallin mukaisesti. Ohjattu koneoppiminen voidaan edelleen jakaa kahteen luokkaan tavoitedatan tyypin perusteella: luokitteluun ja regressioon (kuvio 1). (4, s. 13; 6.)

Mikäli data voidaan jakaa erillisiin ryhmiin, on kyse luokittelusta, kuten esimerkiksi sähköpostin jakaminen tärkeisiin ja roskaposteihin. Datan ollessa jatkuvaa on kyse regressiosta, kuten esimerkiksi lämpötilan määrittäminen. Ohjattua oppimista käytetään ainakin kuvien tunnistamiseen, tekstin luokitteluun ja ennustamiseen. (4, s. 13; 6.)

2.1.2 Ohjaamaton oppiminen

Ohjaamattomassa oppimisessa (unsupervised learning) järjestelmälle ei kerrota oikeaa vastausta, vaan algoritmin on pääteltävä haluttu tulos. Tavoitteena on tutkia dataa ja löytää siitä jokin rakenne. Yleensä datasta etsitään riippuvuuksia, toistoja tai suhteita. Algoritmi opetetaan tunnistamaan tällaiset säännöllisyydet ja ryhmittelemään eli klusteroimaan tiedot omiin joukkoihinsa (kuvio 1). Ohjaamaton oppiminen toimii hyvin esimerkiksi poikkeavien tapahtumien tunnistamisessa. Ohjaamatonta oppimista käytetään myös esimerkiksi asiakassegmentointiin ja suosittelujärjestelmiin. (4, s. 13.)

Klusterointialgoritmit, esimerkiksi K-means, yrittävät löytää yhtäläisyyksiä tietojoukosta ryhmittelemällä objektit siten, että saman klusterin objektit ovat samankaltaisempia keskenään kuin toisen klusterin objektit. Ryhmittely klusteriin tehdään käyttämällä erilaisia kriteerejä, kuten esimerkiksi pienimmät etäisyydet, datapisteiden tiheys tai erilaiset tilastolliset jakaumat. K-means ryhmittelee samanlaiset datapisteet klusteriksi minimoimalla geometristen pisteiden keskimääräisen etäisyyden. Tätä varten se jakaa tietojoukot iteratiivisesti kiinteään määrään (K) ei-päällekkäisiä aliryhmiä eli klustereita, joissa jokainen datapiste kuuluu klusteriin, jolla on lähin keskimääräinen klusterin keskus. K-means on yksi suosituimmista käytetyistä klusterointialgoritmeista, ja sitä käytetään erityisesti poikkeamien tunnistamiseen ja tekstien sekä dokumenttien läpikäymiseen. (7.)

2.1.3 Vahvistettu oppiminen

Vahvistetussa oppimisessa (reinforcement learning) dataa ei ole saatavilla, vaan kone oppii saamastaan palautteesta. Kone saa ympäristössään positiivista ja negatiivista palautetta toiminnastaan ja muokkaa algoritmiaan saadun palautteen mukaisesti. Kone pyrkii minimoimaan saadun negatiivisen palautteen. Tämän tyyppisessä oppimisessa on kolme pääkomponenttia: kone (oppija), ympäristö (koneen toimintaympäristö ja vuorovaikutukset) ja toimet (mitä kone voi tehdä). Tavoitteena on, että kone valitsee toimet, jotka maksimoivat positiivisen palautteen. (4, s. 14; 5.)

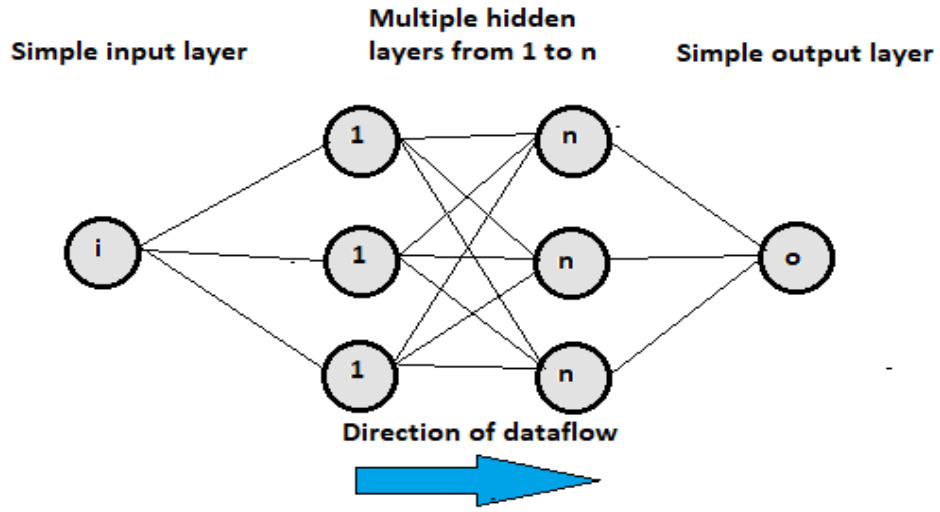
Vahvistusoppimista käytetään usein robotiikassa, pelaamisessa ja itseohjautuvissa autoissa (4, s. 14; 5). Vahvistusoppimista voidaan havainnollistaa esimerkiksi shakkipeliesimerkin avulla. Siinä ympäristönä toimii pelilauta, oppijana pelaaja ja toimina ovat siirrot. Yksittäisellä siirrolla ei varsinaisesti ole merkitystä, vaan tavoitteena on voittaa peli (positiivinen palaute). (8 s. 13–14.)

2.2 Data Mining

Data Mining tai **KDD** (Knowledge Discovery in Data) on tietojen hakemiseen liittyvä prosessi, joka hyödyntää koneoppimisen, tilastotieteen ja tietokantajärjestelmien eri menetelmiä. Terminä Data Mining on hieman harhaanjohtava, sillä sen tavoitteena on suurista tietomääristä tiedon ja riippuvuuksien löytäminen, ei niinkään datan kaivaminen tai louhiminen. Tavoitteena Data Mining -menetelmässä on käydä läpi suuria tietomääriä käyttäen älykkäitä keinoja ja muuntaa tiedot ymmärrettävään rakenteeseen edelleen käytettäväksi. Itse Data Mining -menetelmä on suurten tietomäärien puoliautomaattinen tai automaattinen analyysi, joka pyrkii löytämään aiemmin tuntemattomia ja mielenkiintoisia kuvioita datasta. Data Mining koostuu yleensä neljästä päävaiheesta: tavoitteiden asettamisesta, tiedon keräämisestä ja valmistelusta, tiedolle käytettävien algoritmien soveltamisesta ja tulosten arvioinnista. (9.)

Data Mining -menetelmässä käytetyimpiä algoritmeja ja menetelmiä ovat assosiaatiosäännöt, neuroverkot, KNN-algoritmi (K-Nearest Neighbor) ja päätelmät. Assosiaatiosäännöt ovat sääntöihin perustuvia menetelmiä tietyn tietojoukon muuttujien välisten suhteiden löytämiseksi. Näitä menetelmiä käytetään usein esimerkiksi markkinakorianalyysissä, mikä antaa yrityksille mahdollisuuden ymmärtää paremmin eri tuotteiden välisiä suhteita. (9.)

Neuroverkot pyrkivät matkimaan ihmisaivojen toimintaa hyödyntämällä syväoppimisalgoritmeja koulutusdatan tietojen käsittelemiseksi. Syväoppimisen katsotaan kuuluvan osaksi koneoppimista, ja se on pohjimmiltaan neuroverkko, jossa on kolme tai useampia kerroksia. Neuroverkot rakentuvat kerroksellisista soluista (kuvio 2). Jokainen solu koostuu syötteistä, painoista, kynnysarvosta ja lähdöstä. Lähtöarvon ylittäessä tietyn kynnysarvon aktivoidaan seuraava solu samalla siirtäen tiedot verkon seuraavaan kerrokseen. (9.)



KUVIO 2. Yksinkertaistettu neuroverkko (9)

KNN-algoritmi on ohjatun oppimismenetelmän algoritmi ja se luokittelee datapisteet niiden läheisyyden ja muiden saatavilla olevien tietojen perusteella. Algoritmi olettaa, että samankaltaisia datapisteitä löytyy läheltä toisiaan. Se pyrkii laskemaan datapisteiden välisen etäisyyden matemaattisen menetelmän euklidisen etäisyyden avulla ja määrittämään tietojoukosta yleisimmin esiintyvän asian ilmaantuvuuden tai keskiarvon avulla. (9.)

Päätelmät (decision trees) käyttävät luokittelu- tai regressiomenetelmiä mahdollisten tulosten luokittelemiseen tai ennustamiseen. Nimensä mukaisesti päätelmät käyttävät puumaista visualisointia edustamaan syntyneiden päätösten tuloksia. (9.)

2.3 Ohjelmointikielet koneoppimisessa

Koneoppimisen käyttämiseen ei ole olemassa yhtä ainoaa ja parasta ohjelmointikieltä, vaan kaikki ovat sopivia omilla alueillaan. Osa kielistä sopii kuitenkin koneoppistehtäviin paremmin kuin toiset, ja yksi tällaisista kielistä on Python. Pythonia käyttävät koneoppimiseen ainakin seuraavat melko tunnetut yritykset: Google, Meta, Dropbox, Netflix, YouTube, Uber ja Amazon. (10).

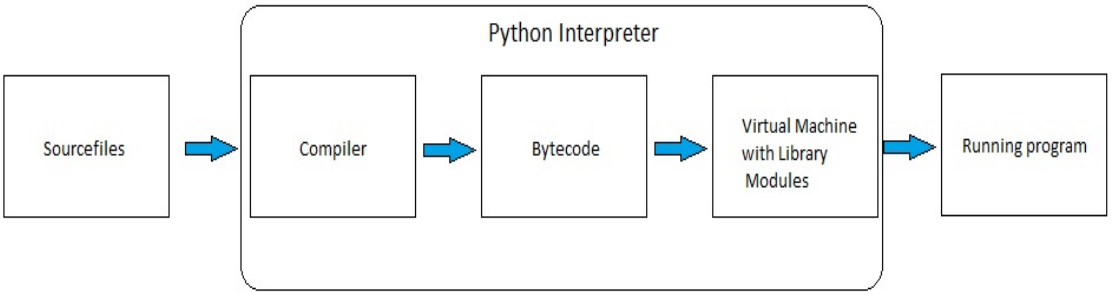
Python on nykyään yksi suosituimmista käytetyistä ohjelmointikielistä ja suosituin koneoppisen kanssa käytettävä kieli. Pythonia käytetään paljon myös esimerkiksi automaatiossa ja tieteellisessä tietojenkäsittelyssä sekä erilaisissa tekoälyyn liittyvissä projekteissa. Pythonin suosio perustuu laajoihin valmiisiin kirjastoihin ja paketteihin, jolloin ohjelmoijan ei tarvitse aloittaa kaikkea alusta, vaan

hän voi keskittyä suoraan ongelmiensa ratkaisemiseen käyttämällä sopivia kirjastoja. Koneoppiminen vaatii jatkuvaa datan käsittelyä, ja Pythonista löytyy sisäänrakennetut kirjastot lähes kaikelle tälle. Esimerkiksi kirjastot SciKit ja Numpy ovat erittäin käytettyjä tekstidatan läpikäymiseen. Pythonin suosion puolesta puhuvat myös sen joustavuus sekä selkeä syntaksi. Siksi onkin perusteltua käydä syntaksia myös tässä opinnäytetyössä hieman läpi. (10.)

2.3.1 Python-tulkin toiminta

Hollantilainen Guido van Rossum loi Python-kielen 1990-luvun alussa. Idean kielelle hän sai työskennellessään monimutkaisten C- ja Unix-komentojen parissa. Hän halusi luoda helpommin ymmärrettävän komentosarjakielen, jolla luodut komennot ja ohjelmat olisivat myös helpommin toteutettavissa kuin terminaaliympäristössä luodut vastaavat. (11.)

Python on tulkattava ja oliopohjainen käyttöjärjestelmästä riippumaton korkean tason ohjelmointikieli (12). Lähdekooditiedostot suoritetaan Python-tulkin avulla rivi kerrallaan. Tulkki kääntää ajon aikaisesti lähdekoodin tietokoneen ymmärtämäksi tavukoodiksi, jota suoritetaan virtuaaliympäristössä. Ohjelman suorittaminen lopetetaan välittömästi, mikäli ajon aikana esiintyy virhe. (13.) Python-ohjelman suorittaminen ja tulkin toiminta on havainnollistettu kuviossa 3. Pythonin sisäänrakennettujen tietoluokkien, kirjastojen ja helpon syntaksinsa sekä tehokkuutensa takia Python on nykyään yksi suosituimmista ohjelmointikielistä (12).



KUVIO 3. Python-tulkin toiminta (13)

2.3.2 Selkeä syntaksi

Kommentteja Pythonissa lisätään lähdekoodiin #-merkillä alkavilla riveillä. Näitä rivejä Python-tulkki ei käsittele, vaan ne ovat lukijalle selkeyttäviä ja informatiivisia rivejä. Yleisesti voidaan kuitenkin

sanoa, että muuttujien ja funktioiden nimet kannattaisi nimetä mahdollisimman kuvaaviksi, jolloin ohjelmakoodi on mahdollisimman helppolukuista ja itseään kommentoivaa (14). Tarvittavien kommenttien määrä on näin mahdollisimman vähäinen. Kuviossa 4 on kuvattu Pythonin syntaksia sekä ylimääräistä kommentointia, joka varsinkin laajemmissa sovelluksissa vaikuttaa jo merkittävästi ohjelmakoodin selkeyteen.

```
27      # luokan julkinen metodi, def-keyword määrittelee funktion/metodin
28      def tulosta_taulukko(self):
29          #python tukee for ja while silmukoita asioiden toistamiseen
30          for iter in self.taulukko:
31                  print(iter)
32
33      # luokan protected metodi, joka arpoo satunnaisen luvun
34      # väliltä 0-20 ja lisää sen taulukkoon
35      def _arvo_numero(self):
36          self.taulukko.append(random.randint(0,20))
37
38      # luokan privaatti metodi, joka toistaa metodia _arvo_numero
39      # numerot muuttujan arvon verran
40      def __tayta_taulukko(self):
41          for iter in range(self.numerot):
42              self._arvo_numero()
43              iter += 1
44  # pythonissa ehdollisia tapahtumia suoritetaan if, elif, else -rakenteella
45  if __name__ == "__main__":
46      # Objektin luominen luokasta
47      tulostin = Tulostin(24) # <-- taulukko täytetään annetun arvon mukaan
48      tulostin.tulosta_taulukko()
```

KUVIO 4. Palanen Pythonin syntaksia havainnollistavaa ohjelmaa

Pythonissa muuttujat ovat dynaamisesti tyypitettyjä eli muuttujille ei tarvitse erikseen ilmoittaa niiden tyyppiä. Tulkki päättelee muuttujien tyypin niille annettujen arvojen mukaisesti. Koodilohkot erotetaan Pythonissa toisistaan sisennyksin. Pythonissa tulkki tunnistaa luokan *class*-tunnussanasta ja funktiolle käytetään *def*-tunnussanaa.

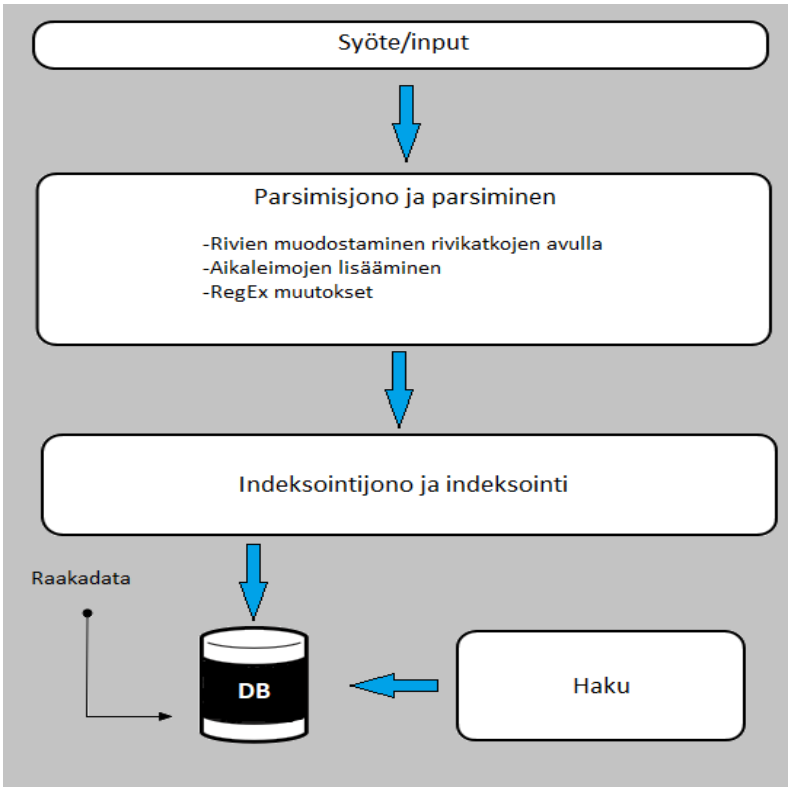
3 SPLUNK-OHJELMISTO

Splunk on konetuotetun datan (machine-generated data) etsimiseen, valvontaan ja analysointiin kehitetty alusta, jonka on kehittänyt Splunk Inc. 2000-luvun alussa. Splunk-ohjelmistoa hallitaan verkkoselaimen avulla. Alun perin Splunk-ohjelmistossa keskityttiin tehokkaaseen hakukoneeseen, jolla pystyttiin etsimään ja tallentamaan tietoa erilaisista systeemeistä kerätyistä lokitiedostoista. Nykyään Splunk voi käsitellä eri järjestelmien ja laitteiden tuottamaa dataa jäsennellyistä JSON-tiedostoista jäsentelemättömiin lokitiedostoihin. (15.)

Splunk-ohjelmiston käyttö vaatii lisenssin, ja tarjolla onkin useita erilaisia vaihtoehtoja erilaisiin käyttötarpeisiin, kuten esimerkiksi Free trial, Splunk Enterprise ja Splunk Cloud. Free trial -lisenssiä käytetään lähinnä palveluun tutustumiseen ennen ostopäätöstä. Splunk Enterprise on varmasti käytetyin lisenssi, ja siinä maksetaan yleensä indeksoidun datan määrän mukaisesti. Splunk Cloud -lisenssi on käytännössä pilvessä oleva Splunk Enterprise. Splunk-ohjelmistosta löytyy myös esimerkiksi tietoturvaan painottuvia lisenssejä. (16.)

3.1 Datan kerääminen ja indeksointi

Splunk ei voi käyttää raakadataa suoraan, vaan se täytyy ensin käsitellä ja muuntaa Splunk-ohjelmiston hakukoneiston ymmärtämään muotoon. Splunk-ohjelmiston dataputkessa katsotaankin olevan neljä eri vaihetta: syöte, parsiminen, indeksointi ja haku. Dataputken malli on kuvattu kuviossa 5. (17.)



KUVIO 5. Splunk-ohjelmiston dataputki (17)

Syöte-vaiheessa Splunk saa raakadatan lähteestään ja pilkkoo sen 64 kilotavun palasiin ja lisää jokaiseen palaseen kuvailevaa tietoa eli metadataa, minkä jälkeen Splunk Enterprise parsii syötteen palaset loogisiin komponentteihin (event processing) (17). Event eli tapahtuma voi olla mitä tahansa, kunhan tällä tapahtumalla on tietty tapahtuma-aika (18).

Parsimisvaiheessa data jaetaan riveihin ja jokaisesta rivistä tunnistetaan tai riveihin lisätään aikaleimat. Data myös muunnetaan vastaamaan säännöllistä lauseketta (Regular Expression), mikä tekee tapahtumien hausta erittäin tehokasta. Seuraavaksi parsittu data indeksoidaan. Indeksoinnissa data muunnetaan etsittäviksi tapahtumiksi ja tallennetaan raakadatan kanssa levylle. Lopuksi hakuvaihe määrittää, miten käyttäjä suorittaa hakuja, käyttää indeksoitua dataa ja luonnollisesti hallinnoi koko hakuprosessia. (17.)

3.2 Tehokas hakukieli

SPL (Search Processing Language) on Splunk-ohjelmistossa käytettävä hakukieli, jolla tehdään monimutkaisia hakuja indeksoituun tietokantaan. Se on oikeastaan sarja hakukomentoja ja erityisen tehokas konetuotetun datan haussa, tutkimisessa, analysoimisessa sekä visualisoimisessa.

Nykyään SPL:stä löytyy uudempi versio SPL2, joka tukee sekä SPL:n hakukomentoja että SQL (Structured Query Language) -tietokantakielen komentoja. (19.)

SPL-hakukomentoja on mahdollista putkittaa (pipe) |-merkkiä käyttäen. Komentojen putkittamisessa seuraava komento käyttää edellisestä komennosta palautettuja arvoja syötteenään; vastaava tapa on käytössä esimerkiksi Unix-terminaaliympäristössä. (19.)

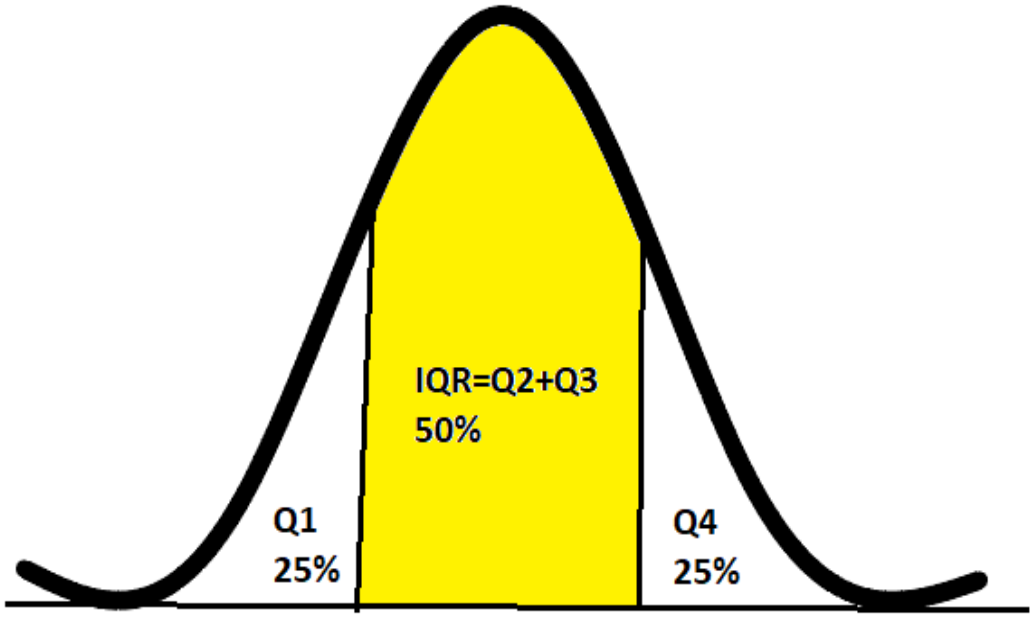
SPL2 tukee jatkettua hakua ja riippumatonta hakua. Jatketussa haussa ideana on hyödyntää aiempaa hakua uuden haun tekemiseen, kun taas riippumattomassa haussa voidaan suorittaa useita toisistaan riippumattomia hakuja. Kuviossa 6 on havainnollistettu yksinkertaisella esimerkillä SPL2:n jatkettua hakua SQL-syntaksia käyttäen. (19.)

```
1  // PALAUTTAA KRITEEREITÄ VASTAAVAT EVENTIT
2  $example_base_search = from example_events where status=200
3
4  // PALAUTTAA KATEGORIAT, JOTKA ALKAVAT "S" WWW4 PALVELIMELTA
5  $example_child1 = from $example_base_search
6  where categoryId LIKE("S%") AND host="www4"
7  select _time, action, productId, categoryId
8
9  // SUODATTAA NULL ARVOT ACTION KENTÄSTÄ
10 $example_child2 = from $example_child1 where action!="NULL"
11
```

KUVIO 6. SPL2-haku SQL-syntaksia käyttäen

Splunk-ohjelmiston *anomalydetection* on SPL-komento, joka tunnistaa poikkeavia tapahtumia laskemalla niiden todennäköisyyden ja havaitsemalla epätavallisen pieniä todennäköisyyksiä. Menetelmässä on mahdollista valita kolme eri tapaa poikkeamien tunnistamiseen: histogrammi, z-piste tai iqr. Menetelmä myös mahdollistaa tapahtumien palauttamisen lisäkentillä, poikkeavien arvojen poissuodattamisen tai yhteenvedon palauttamisen poikkeamien tilastoista. (20.)

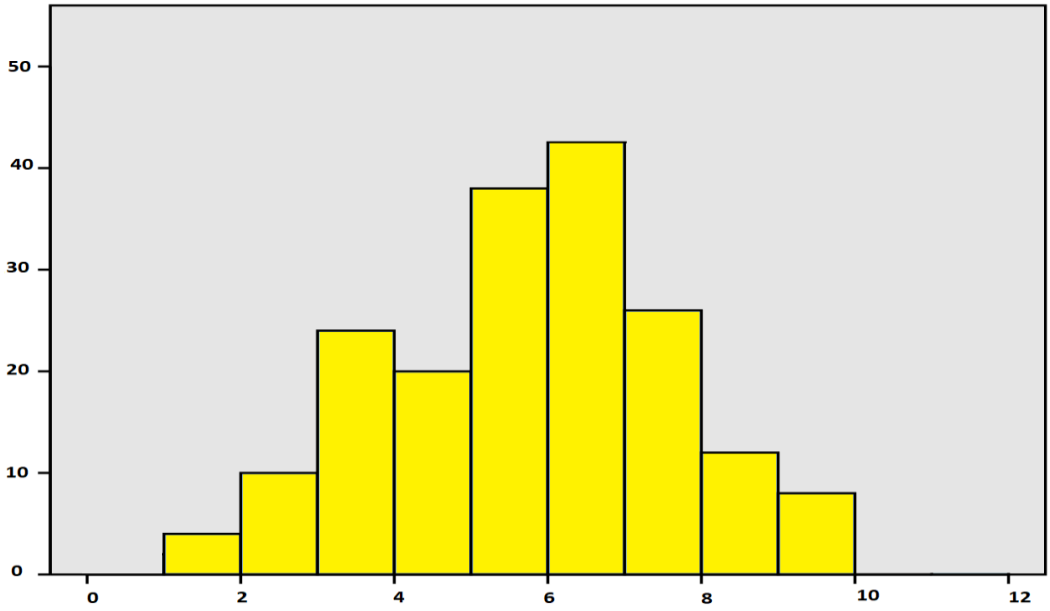
Iqr-menetelmä (IQR) perustuu tilastolliseen käsitteeseen nimeltä interkvartiiliväli, joka on erotus datan ylemmän ja alemman neljänneksen välillä (kuvio 7). IQR kuvaa datan hajontaa keskialueella, ja siihen kuuluu 50 prosenttia tutkimusdatasta tai joukosta. Iqr-menetelmä laskee jokaiselle tapahtumalle poikkeavuusarvon, joka on sen etäisyys IQR:stä. Jos poikkeavuusarvo on suurempi kuin määritelty raja-arvo, tapahtuma luokitellaan poikkeavaksi. (21.)



KUVIO 7. Interkvartiilivälin muodostuminen (21)

Z-piste-menetelmä perustuu datan keskiarvoon ja keskihajontaan, jotka ovat tilastollisia tunnuslukuja. Nämä tunnusluvut kuvaavat datan keskusta ja hajontaa. Z-piste-menetelmä laskee jokaiselle tapahtumalle z-pisteen, joka on sen etäisyys keskiarvosta keskihajonnan yksiköissä. Jos z-piste on suurempi kuin määritelty raja-arvo, tapahtuma luokitellaan poikkeavaksi. (22.)

Histogrammi on tilastotieteessä käytetty graafinen esitys, joka perustuu datan jakamiseen tasavälisiin luokkiin ja niiden frekvenssien laskemiseen. Kuviossa 8 on kuvattu yksinkertainen esimerkki histogrammista, jossa on esitetty 185 satunnaisen havainnon esiintyminen (x-akselilla havainnon arvo ja y-akselilla havaintojen lukumäärä). Histogrammi-menetelmä laskee jokaiselle tapahtumalle todennäköisyyden, että se kuuluu tiettyyn luokkaan. Jos todennäköisyys on pienempi kuin määritelty raja-arvo, tapahtuma luokitellaan poikkeavaksi. (23.)



KUVIO 8. Esimerkki histogrammista (23)

3.3 Koneoppimisen hyödyntäminen

Yksi Splunk-ohjelmiston tärkeimpiä ominaisuuksia on koneoppimisen hyödyntäminen konetuotetun datan läpikäymiseen. SMLTK-lisäosasta (Splunk Machine Learning Toolkit) löytyy valmiita avoimen lähdekoodin algoritmeja sekä visualisointityökaluja datan käsittelemiseksi ja tulosten havainnollistamiseksi. Laajennusosa tarjoaa laajemmat työkalut hakutulosten poikkeamien havaitsemiseen, ennakoivaan analytiikkaan sekä klusterointiin, mikäli esimerkiksi Splunk-ohjelmiston anomalydetection-komento ei toimi halutulla tavalla. SMLTK-lisäosasta löytyvät ainakin seuraavat valmiit, ohjaamattomaan koneoppimiseen perustuvat, algoritmit poikkeamien tunnistamiseen: DensityFunction, LocalOutlierFactor ja OneClassSVM. (24; 25.)

SMLTK-lisäosassa koneoppimisen algoritmeja voidaan käyttää suoraan poikkeamien tunnistamiseen. Splunk-ohjelmistoon on myös mahdollista lisätä omia algoritmeja, mikäli käytössä olevista ei löydy sopivaa. SMLTK on suoraan yhteydessä SPL-hakukieleen, joten koneoppisen hyödyntäminen onnistuu jo hakuvaiheessa. (26.)

Yleensä Splunk-ohjelmistossa yksi SPL-haku riittää kattamaan tarvittavat tiedot. Koneoppimisanalytiikan tapauksessa koneoppimismalli on ensin koulutettava datalla, jota käytetään myöhemmin tarvittavien tulosten käsittelyyn. Jos hakutiedoille on mahdollista luoda hakutuloksia rikastava

lookup-tapahtuma, koneoppimisen keinot voidaan ottaa käyttöön. Splunk-ohjelmiston lookup mahdollistaa tapahtumien kenttä-arvoparien yhdistämisen ulkoisten hakutaulukoiden kenttä-arvopareihin, jos Splunk löytää nämä kenttä-arvoparit hakutaulukosta, lisätään vastaavat kenttä-arvoparit hakutaulukosta tapahtumiin. (27.)

Splunk-ohjelmistossa on neljä erilaista lookup-tyyppiä: CSV lookup, ulkoinen lookup, KV Store lookup ja Geospatial lookup. Jokainen lookup-tyyppi vaatii lookup-määritelmän, joka kertoo Splunk-ohjelmistolle käytettävät hakutaulukot, vertailtavat kentät ja tapahtumiin lisättävät kentät. Lookup-määritelmän voi luoda Splunk-ohjelmistolle käyttöliittymässä tai muokkaamalla konfiguraatiotiedostoa. (28.)

CSV lookup -tyyppiä (Comma Separated Values) käytetään, kun halutaan käyttää csv-tiedostoja hakutaulukoina. Csv-tiedostot ovat staattisia taulukoita, joissa jokainen sarake edustaa mahdollisia kenttien arvoja. CSV lookup -tyyppiä voidaan käyttää, kun käytössä on pieniä ja suhteellisen muuttumattomia tietoja. (28.)

Ulkoista lookup -tyyppiä käytetään, kun halutaan käyttää Python-skriptejä tai binäärisiä ohjelmia hakutaulukoina. Ulkoiset lookup -tyypit voivat hakea kenttien arvoja ulkoisista lähteistä, kuten esimerkiksi DNS-palvelimista tai tietokannoista. Ulkoista lookup -tyyppiä voidaan käyttää, kun käytössä on dynaamisia tai monimutkaisia tietoja. (28.)

KV (Key Value) Store lookup -tyyppiä käytetään, kun halutaan käyttää KV Store -kokoelmia hakutaulukoina. KV Store -kokoelmat ovat MongoDB-tietokantoja, jotka tallentavat avain-arvopareja. KV Store lookup -määritelmiä voidaan käyttää, kun käytössä on suuria tai usein päivittyviä tietoja. (28.)

Geospatial lookup -tyyppiä käytetään, kun halutaan käyttää KMZ-tiedostoja (Keyhole Markup Zipped) hakutaulukoina. KMZ-tiedostot määrittelevät esimerkiksi karttojen ja niistä maiden tai osavaltioiden alueiden rajat. Geospatial lookup -tyyppiä voidaan käyttää luomaan kysely, jota Splunk käyttää kloropleettikartan luomiseen. Kloropleettikartta eli teemakartta kuuluu alueluokituskarttoihin, jossa asiat tai aineisto esitetään yleensä alueittain. Kloropleettikartassa värin tummuudella välitetään erilaista tietoa, kuten esimerkiksi asukastiheyttä eri maissa. (28.)

4 POIKKEAMIEN TUNNISTAMINEN SPLUNK-OHJELMISTOLLA

Tämän opinnäytetyön tavoitteena oli luoda toimiva reitti jatkuvasta integraatiosta verkkoliikennelaitteiden testauksesta syntyvien lokitiedostojen viemiseksi Splunk-ohjelmiston palvelimelle. Toimeksiantajan ehdotuksesta päädyimme kuitenkin kokeilemaan asetelmaa aluksi ohjelmistokehittäjien paikallisessa Docker-container-kehitysympäristössä, johon Splunk-ohjelmisto asennettiin paikallisesti.

Osastolla, jolle tämä opinnäytetyö tehdään, ei ole tällä hetkellä käytössä Splunk-ohjelmistoa, joten päädyimme PoC-ratkaisuun (Proof of Consept). PoC-ratkaisussa on ensin tarkoitus tutkia Splunk-ohjelmiston toimivuutta, tarvetta ja mahdollista hyötyä osastolle paikallisessa ympäristössä. Mikäli tarve löydetään, tullaan Splunk mahdollisesti myöhemmin lisäämään osaston työkaluvalikoimaan DevOps-tiimin toimesta. DevOps on lyhenne sanoista ohjelmistokehitys (Development) ja palveluntarjonta (Operations). DevOps:n ideana on esimerkiksi automatisoida ylläpitoon ja ohjelmistokehitykseen liittyvät IT-tukitoiminnot (29). Toimeksiantaja osasi ehdottaa PoC-ratkaisua nopeasti, sillä aiemmissa Splunk-ohjelmistoon liittyvissä töissä toimeksiantajan palomuurit ja mahdolliset muut yhteysongelmat ovat olleet suurimmat kompastuskivet ja aikatauluun vaikuttaneet asiat työn aikana (30 s. 19–20).

4.1 Testiympäristön adaptoiminen

Toimeksiantaja käyttää Pythonin pytest-testialustaa osastollamme langattomien verkkoliikennelaitteiden testaamiseksi. Järjestelmälle on mahdollista testisuorituksen aikana antaa komento, jolla aloitetaan lokien kerääminen zip-paketiksi. Binääritiedostot, jotka sisältävät ytimien ja objektien suoritusaikoja, sisältyvät tähän zip-pakettiin. Testiympäristössä ei kuitenkaan ollut valmiina keinoja purkaa zip-pakettia tai parsia suoritusaikatietoja sisältäviä binääritiedostoja, joten testiympäristöä oli muokattava datan saamiseksi oikeanmuotoisena Splunk-ohjelmistolle. Nämä muokkaukset toimivat sekä paikallisessa- että CI-ympäristössä sellaisenaan.

Testiympäristöön lisättiin toiminnallisuus zip-paketin keräämisen käyttöönottamiseksi. Käyttöönottaminen oli melko suoraviivaista, mutta vaati käytännössä kaikkien testilistoilla olevien testien muokkaamista. Toiminnallisuus on tällä hetkellä käytössä myös CI-ympäristössä siten, että zip-

paketti kerätään epäonnistuneista ajoista, mutta zip-paketin voi halutessaan myös kerätä erillisellä komennolla haluamastaan testitapauksesta.

4.1.1 Binääritiedostojen purkaminen

Zip-paketin keräysskriptiin lisättiin toiminnallisuus paketin purkamiseksi ja tarvittavien tiedostopolkujen luomiseksi. Pythonin subprocess-moduulia ja sen *run*-metodia hyödynnettiin luodun komennon suorittamiseksi testiajon aikana. Kuviossa 9 on esimerkin avulla havainnollistettu zip-paketin purkamiseen tarvittava toiminnallisuus.

```
8       cmd = f"cd {self.log_path}/ && mkdir -p logs" \
9           '&& mkdir -p zip_files && unzip -qq logs*.zip -d temp/ ' \
10          '&& tar -xzf temp/em_time_logs.tgz -C temp/ ' \
11          '&& mv temp/em_time_log*.bin logs/ && rm -rf temp/ ' \
12          '&& mv logs*.zip zip_files/'
13      subprocess.run(cmd, shell=True, check=True)
```

KUVIO 9. Binääritiedostojen purkamista havainnollistava toiminnallisuus

Kuvion 9 zip-paketin purkamisen mahdollistava toiminnallisuus on sekoitus Pythonin syntaksia sekä komentorivi- eli terminaalikomentoja. Purkamisen jälkeen binääritiedostot täytyi vielä parsia Splunk-ohjelmiston ymmärtämään muotoon.

4.1.2 Binääritiedostojen parsiminen

Toimeksiantajalla oli käytössä sisäinen työkalu binäärimuotoisten suoritusaikatiedostojen parsimiseen. Tämä työkalu osasi muuntaa binäärimuotoiset suoritusaikatiedostot suoraan Splunk-ohjelmiston ymmärtämään .csv-formaattiin. Muunnetuissa csv-tiedostoissa ytimien sekä tapahtumien objektien suoritusajat ja muut oleelliset tiedot olivat listattuna pilkuilla erotettuina. Kuviossa 10 on havainnollistettu sisäisen työkalun käyttöönottoon tarvittu toiminnallisuus.

```
18      def _parse_logs_with_inner_tool(self):
19          tool_path = {self.root} / 'tool'  / 'tool_runner.py'
20          tool_command = f"cd {self.log_path}/logs/ && python3 \'{tool_path}\' " \
21              '--args_first_bin m_time_log_first.bin ' \
22              '--args_second_bin m_time_log_second.bin'
23          subprocess.run(tool_command, shell=True, check=True)
24          self._remove_tool()
```

KUVIO 10. Binäärimuotoisten lokitiedostojen parsimista havainnollistava toiminnallisuus

Sisäisen työkalun käyttämiseen oli luotu Python-skripti, joka haki ohjelman suorittamiseen tarvittavat, viimeisimmän version mukaiset, binäärit ohjelman omasta repositorystä ja parsi unix-komennossa argumentteina annetut binääritiedostot csv-muotoisiksi. Työkalun käyttö komentoriviltä valmiilla skriptillä oli verrattain yksinkertaista, kunhan huolehdittiin oikeista hakemistopoluista. Binääritiedostojen parsimisen jälkeen sisäinen työkalu ja siihen liittyvät tiedostot poistettiin ja csv-tiedostot siirrettiin omaan hakemistoonsa, josta Splunk-ohjelmisto indeksoi tiedostot tapahtumiksi.

4.2 Splunk-ohjelmiston ja lisäosien asentaminen

PoC-ratkaisun mukaisesti Splunk Enterprise asennettiin paikallisesti Docker-containeriin. Splunk voitiin asentaa esimerkiksi Mac-, Windows- ja Linux-käyttöjärjestelmille, mutta tässä työssä käytettiin Linux-ympäristöä. Ennen asentamista tarvittiin käyttäjätili. Mikäli sellaista ei ollut, käyttäjä pystyi rekisteröitymään varsin helposti Splunk-ohjelmiston kotisivuilla. Maininnan arvoista on, että Splunk Enterprise -ohjelmistoa pystyi kokeilemaan 2 kuukauden ajan maksutta niin sanotulla kokeilulisenssillä, jonka jälkeen käyttöä pystyi edelleen jatkamaan ilmaislisenssillä. Ilmaislisenssillä Splunk Enterprise -ohjelmiston käyttö oli toki rajoitetumpaa, eikä esimerkiksi klusterointi ollut enää mahdollista. (31.)

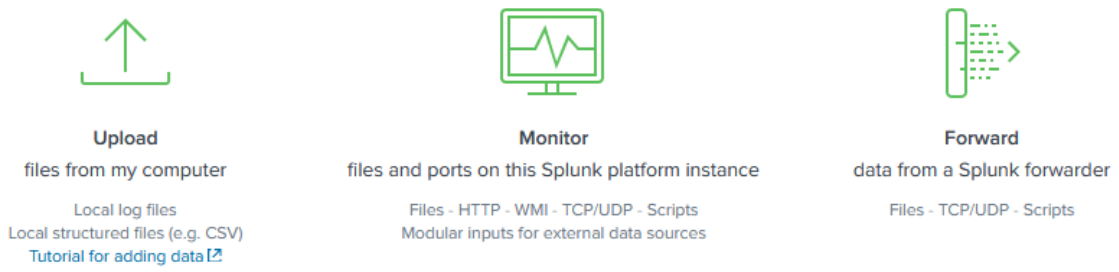
Rekisteröitymisen jälkeen valittiin asennuspaketille oikea alusta Splunk-ohjelmiston kotisivuilta, minkä jälkeen kotisivut tarjosivat linkkiä asennuspaketin lataamiseksi. Asennuspaketti ladattiin paikalliseen kehitysympäristöön ja Splunk asennettiin ja käynnistettiin. Splunk-ohjelmistoa voitiin käynnistämisen jälkeen hallita selainpohjaisen web-käyttöliittymän avulla. Kirjautumiseen käytettiin asennusvaiheessa luotuja tunnuksia. Paikallisessa asennuksessa käyttöliittymä löytyi osoitteesta http://127.0.0.1:8000.

Splunk-ohjelmiston Machine Learning Toolkit on laajennusosa Splunk-ohjelmistoon, ja toimiakseen se vaati myös Python for Scientific Computing -laajennusosan asentamista. Lisäosat asennettiin komentoriviltä Splunk-ohjelmistoon. Kuten edellä olevasta voidaan huomata, Splunk-ohjelmiston ja lisäosien asentaminen paikallisesti oli hyvinkin yksinkertaista.

4.3 Datan lisääminen Splunk-ohjelmistoon

Splunk-ohjelmistoon voitiin lisätä dataa indeksoitavaksi usealla eri tavalla. Splunk-ohjelmisto ja indeksoitava data olivat samalla palvelimella, joten data voitiin lisätä esimerkiksi käsin tai asettamalla Splunk seuraamaan jotain tiettyä tiedostopolkua. Mikäli dataa oli vain muutamia tiedostoja, voitiin ne lähettää indeksoitavaksi manuaalisesti. Datamäärä oli kuitenkin suuri ja jatkuva, joten Splunk asetettiin seuraamaan tiedostopolkua. Splunk tunnisti polussa tapahtuvat tiedostomuutokset, ja lähetti uudet tiedostot automaattisesti indeksoitavaksi.

Splunk-ohjelmistoon oli mahdollista lähettää dataa myös lähettimen kautta (Universal Forwarder). Jos Splunk-ohjelmisto ja indeksoitavaksi lähetettävä data olivat eri palvelimilla, voitiin datan sisältävälle palvelimelle asentaa lähetin, joka yksinkertaisesti lähetti tarvittavan datan Splunk-ohjelmistolle. Kuviossa 11 on esitetty Splunk-ohjelmiston hallintaliittymän keinot, joilla Splunk-ohjelmistoon voitiin lisätä dataa indeksoitavaksi. Nämä keinot jakautuivat edelleen tarkempiin alaosiinsa.



KUVIO 11. Splunk-ohjelmiston hallintaliittymän keinot datan lisäämiseksi

Tässä työssä asetettiin Splunk seuraamaan tiedostopolkua, joka luotiin suoritusaikabinääritiedostojen purkamis- ja parsimisvaiheessa. Seurattavassa tiedostopolussa csv-muotoisia tiedostoja oli yhteensä kaksi kappaletta testisuoritusta kohden. Tiedostomuodon takia Splunk osasi suoraan yhdistää tiedostojen kenttä-arvoparit oikein indeksoinnissa. Tiedostot sisälsivät useita tuhansia rivejä tietoa suoritettavista objekteista, suoritukseen käytettävästä ytimestä sekä suoritukseen käytetystä

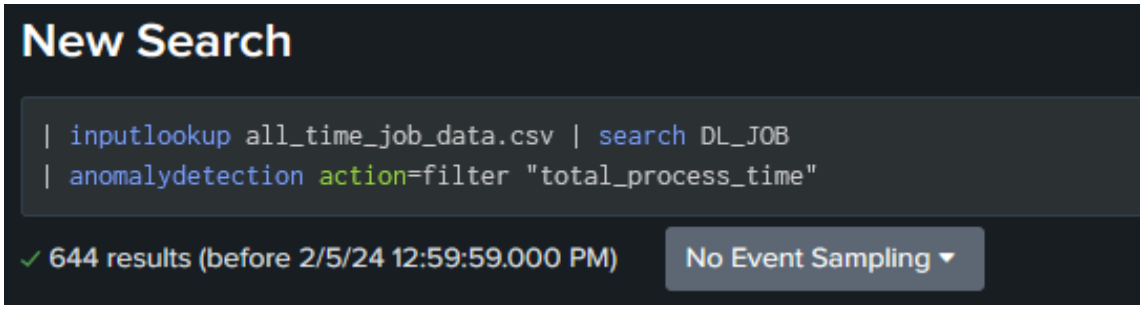
kokonaisprosessointiajasta. Tiedostoissa oli kuitenkin joitain eroja kenttien nimissä, joten tämä täytyi ottaa huomioon SPL-hakuja suunniteltaessa.

4.4 Poikkeamien tunnistaminen anomalydetection-komennolla

Poikkeamien tunnistaminen tehtiin molempien indeksoitujen csv-tiedostojen datalle erikseen, sillä tiedostojen sisältämä data ei ollut keskenään vertailukelpoista. Testiajoista indeksoidut suoritusaikalokitapahtumat tallennettiin tiedostokohtaisesti Splunk-ohjelmiston avulla kahdeksi isoksi csv-tiedostoksi, joita voitiin helposti käyttää pohjahakujen suorittamiseen. Kaikkien tapahtumien kerääminen ja tallentaminen yhteen tiedostoon ei ollut välttämätöntä, mutta tähän ratkaisuun päädyttiin tämän työn kuvioissa esitettyjen esimerkkien selkeyden lisäämiseksi. Tallentaminen suoritettiin komennolla

source="splunk_file_monitor_path/file_name1" | outputlookup file_name1.csv.

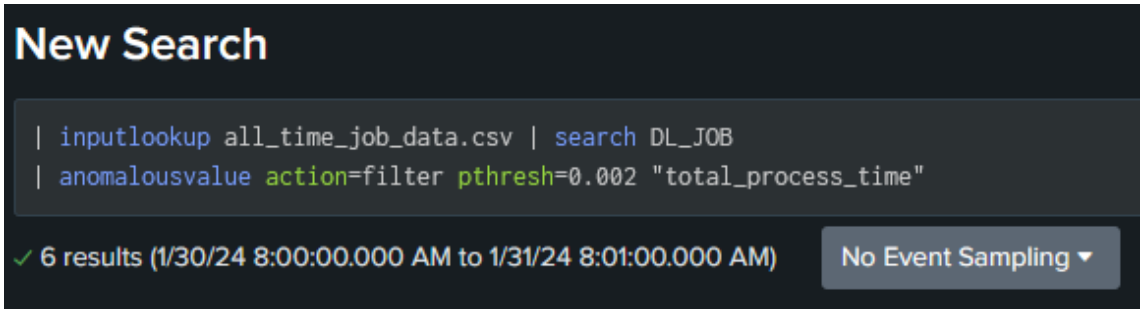
Poikkeamien tunnistaminen Splunk-ohjelmistoon indeksoiduista tiedoista oli mahdollista toteuttaa *anomalydetection*-komennolla kolmen eri menetelmän avulla: histogrammi-, z-piste- ja iqr-menetelmä. Oletuksena anomalydetection-komento käytti histogrammimenetelmää. Menetelmää voitiin vaihtaa käyttämällä method-argumenttia. Histogrammimenetelmässä anomalydetection-komennolle kerrottiin action-argumentilla, mitä poikkeamiksi tunnistetuille tapahtumille tehdään. Action-argumentille voitiin, histogrammin ja z-pisteen tapauksessa, antaa kolme eri arvoa: filter, annotate ja summary. Filter-menetelmä poisti tuloksista muut kuin poikkeavat tulokset, summary-menetelmä teki yhteenvedon löydetyistä poikkeamista ja annotate-menetelmä säilytti kaikki tulokset, mutta lisäsi tuloksiin uusia kenttiä ilmoittamaan poikkeamasta. Kuviossa 12 on esitetty histogrammimenetelmään perustuva SPL-haku suoritusaikatiedostojen poikkeamien tunnistamiseksi.



KUVIO 12. Histogrammi-menetelmään perustuva SPL-hakukomento poikkeamien tunnistamiseksi

Kuviossa 12 esitetty komento etsi ensin kaikista noin viidestä miljoonasta indeksoidusta tapahtumasta DL_JOB-nimiset prosessit sisältävät tapahtumat, jonka jälkeen anomalydetection käytti hausta palautettuja tuloksia syötteenään. Anomalydetection-komennolle annettiin vielä vapaaehtoisena argumenttina kentän nimi, jolle poikkeaman tunnistaminen haluttiin toteuttaa. Haku löysi 644 mahdollisesti poikkeavaa tapahtumaa noin 19 000 tapahtuman kokonaisjoukosta.

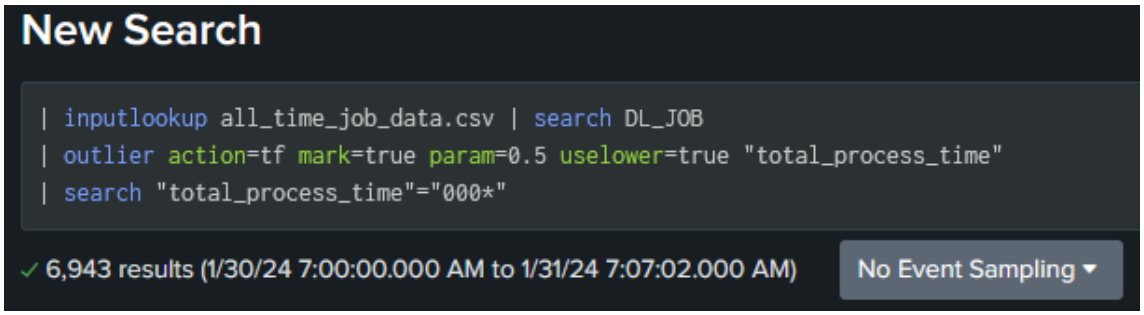
Kuviossa 13 esitetty komento käytti z-piste-menetelmää poikkeamien tunnistamiseen. *Anomalousvalue*-komentoa päädyttiin käyttämään komennon selkeyden lisäämiseksi. Anomalousvalue-komento palautti samat tapahtumat kuin komento *anomalydetection method=zscore.*



KUVIO 13. Z-piste-menetelmään perustuva SPL-hakukomento poikkeamien tunnistamiseksi

Z-piste-menetelmässä voitiin antaa vapaaehtoinen pthresh-argumentti, jolla pystyttiin säätämään poikkeamien tunnistamisen todennäköisyyskynnystä: pthresh-argumentin arvoa pienentämällä saatiin suodatettua enemmän tuloksia. Z-piste-menetelmälle annettiin myös vapaaehtoisena argumenttina kentän nimi, jolle poikkeaman tunnistaminen haluttiin toteuttaa. Haku löysi 6 mahdollisesti poikkeavaa tapahtumaa noin 19 000 tapahtuman kokonaisjoukosta. Pienentämällä pthresh-argumentin arvoon 0.0015 z-piste-menetelmä ei enää löytänyt mahdollisesti poikkeavia arvoja. Vastaavaa hakua voitiin käyttää myös muille tapahtumista löytyville prosesseille, vaihtamalla luonnollisesti etsittävän prosessin nimeä ja säätämällä pthresh-arvoa.

Kuviossa 14 esitelty komento käytti iqr-menetelmää poikkeamien tunnistamiseen. *Outlier*-komentoa päädyttiin käyttämään selkeyden lisäämiseksi. Outlier-komento toimi vastaavasti kuin komento *anomalydetection method=iqr.*



KUVIO 14. IQR-menetelmään perustuva SPL-hakukomento poikkeamien tunnistamiseksi

Iqr-menetelmässä outlier-komennolle voitiin antaa vapaaehtoisia argumentteja: action, mark, param, uselower ja field-list. Field-list-argumentille annettiin arvona kentän nimi, jolle poikkeaman tunnistaminen haluttiin toteuttaa. Param-argumentilla pystyttiin säätämään poikkeamien tunnistamisen todennäköisyyskynnystä. Param-argumentin arvoa pienentämällä poikkeavia tuloksia löytyi enemmän.

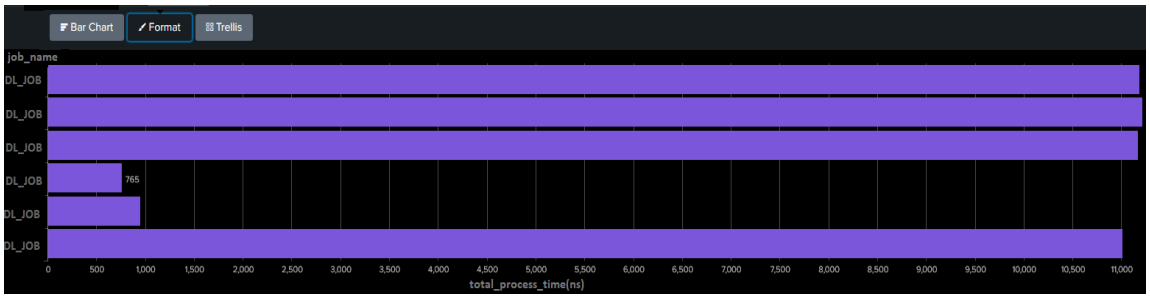
Action-argumentin arvona iqr-menetelmässä voitiin antaa joko transform (tf) tai remove (rm). Käytettäessä action-argumentin arvoa transform ja mark-argumentin arvoa true yhdessä, merkitsi Splunk vapaaehtoisena parametrina annetun kentän poikkeamiksi tunnistettuihin arvoihin etuliitteen 000. Etsimällä tämän etuliitteen sisältäviä arvoja, löydettiin iqr-menetelmän mahdollisesti poikkeamiksi luokitellut arvot, joita param-argumentin arvolla 0.5 löytyi 6 943 kappaletta noin 19 000 tapahtuman kokonaisjoukosta.

Iqr-menetelmässä action-argumentin arvolla remove, Splunk-ohjelmiston oletettiin poistavan poikkeavan tapahtuman palautettujen tulosten listalta. Splunk kuitenkin poisti vain poikkeavan arvon kyseisen tapahtuman kentästä, jätti total_processing_time arvon tyhjäksi ja palautti kaikki noin 19 000 tapahtumaa.

4.5 Hakutulosten visualisoiminen

Hakutulosten visualisoiminen oli myös mahdollista Splunk-ohjelmistolla. Visualisoiminen onnistui Splunk-ohjelmiston sisäänrakennetuilla keinoilla, joista löytyivät vähintäänkin yleisimmät graafiset esitysmuodot. Hakua täytyi yleensä hieman muokata, jotta visualisoiminen yleensä onnistui tai oli selkeää. Kuvion 14 hakua muokattiin lisäämällä komento | *table ”job_name”, ”total_process_time”,*

joka nimensä mukaisesti loi taulukon annetuista kenttien arvoista ja mahdollisti tulosten visualisaation. Kuviossa 15 on havainnollistettu visuaalisesti kuvion 13 SPL-komennon löytämät mahdollisesti poikkeavat suoritusajat DL_JOB-prosessissa.



KUVIO 15. Pylväsdiagrammiesitys poikkeavista prosessointiajoista DL_JOB-prosessissa

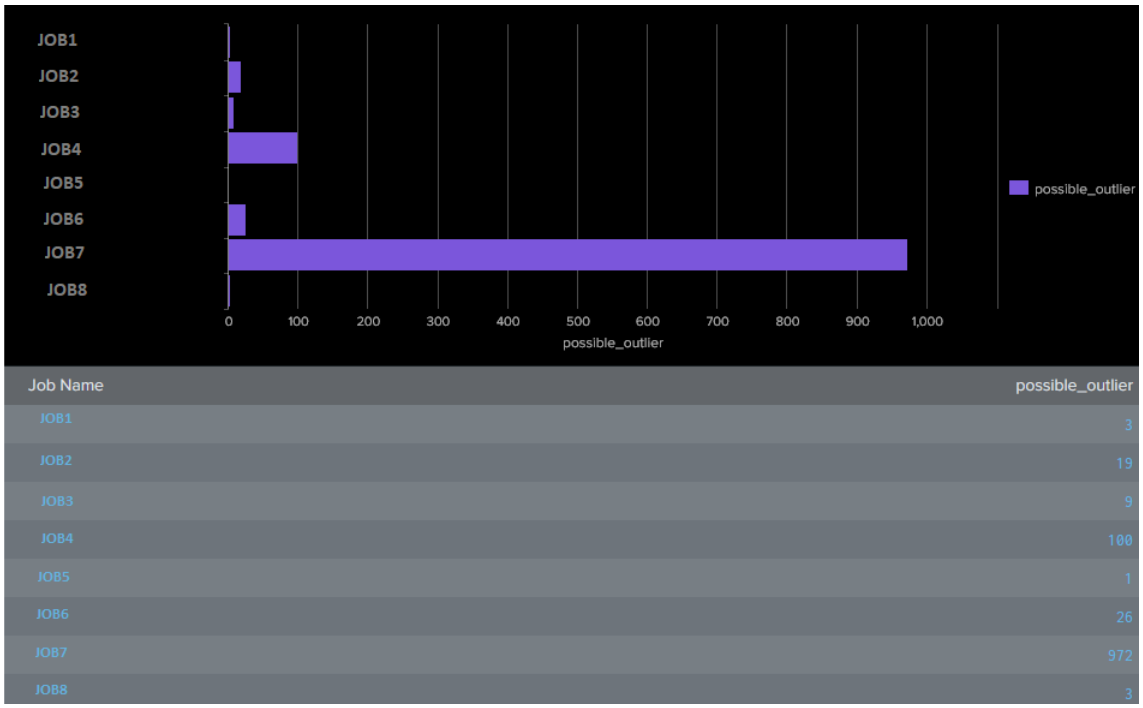
Edellä olevissa esimerkeissä ei käyty kuin yhden prosessin tulokset läpi Splunk-ohjelmistolle indeksoidusta datasta selkeyden lisäämiseksi. Todellisuudessa prosesseja oli yhteensä kaksitoista kappaletta, joten SPL-hakuja täytyi yhdistellä, jotta koko tutkimusdatalle voitiin suorittaa poikkeamien tunnistaminen. Kuviossa 16 on esitetty lyhennelmä vaaditusta SPL-hakukomennon syntaksista kaikelle indeksoidulle datalle.

New Search

```
source="all_job_data" "DL_JOB1"
| anomalousvalue action=filter pthresh=0.002 "total_process_times"
| stats count as possible_outlier by "Job Name"
| append
    [| search source="all_job_data.csv" "DL_JOB2"
     | anomalousvalue action=filter pthresh=0.002 "total_process_time"
     | stats count as possible_outlier by "Job Name"
      ]
      ...
| append
    [| search source="all_job_data.csv" "JOB12"
     | anomalousvalue action=filter pthresh=0.002 "total_process_time"
     | stats count as possible_outlier by "Job Name"
      ]
```

KUVIO 16. SPL-hakukomento kaikkien prosessien poikkeamien löytämiseksi

Kuviossa 17 on esitetty mahdollisten poikkeamien määrä csv-tiedostosta indeksoidusta datasta löytyneille tapahtumille. Tapahtumia oli yhteensä noin 4.8 miljoonaa kappaletta. Haussa käytettiin z-piste-menetelmää pthresh-arvolla 0.0001.



Job Name	possible_outlier
JOB1	3
JOB2	19
JOB3	9
JOB4	100
JOB5	1
JOB6	26
JOB7	972
JOB8	3

KUVIO 17. Mahdollisten poikkeamien määrä eri prosesseissa

4.6 Koneoppimisen hyödyntäminen poikkeamien tunnistamiseen

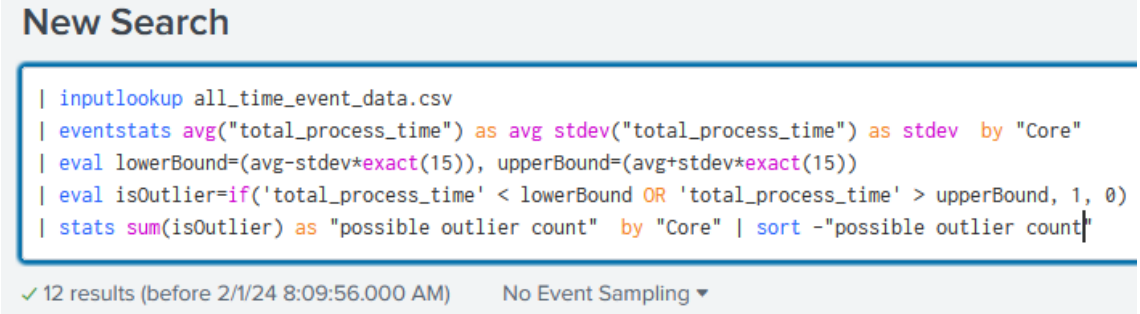
SMLTK-lisäosan asentamisen jälkeen lisäosa oli käytettävissä Splunk-ohjelmiston hallintaliittymän kautta. Lisäosassa oli käytettävissä useita erilaisia tapahtumien käsittelemiseen suunniteltuja menetelmiä, jotka löytyivät **Experiments**-välilehden alta. Tähän työhön valittiin älykäs poikkeaman tunnistaminen (Smart Outlier Detection) ja numeeristen poikkeamien tunnistaminen (Detect Numeric Outliers). Työkalusta löytyi myös esimerkiksi seuraavat menetelmät: älykäs ennustaminen (Smart Forecasting, Smart Prediction) sekä älykäs klusterointi (Smart Clustering).

Splunk asetettiin seuraamaan kahta eri csv-muotoista tiedostoa samasta hakemistopolusta. Toinen tiedostoista, jota käytettiin anomalydetection-komennon kanssa, sisälsi satoja tuhansia rivejä dataa yhtä testisuoritusta kohden. Toinen tiedosto sisälsi noin neljä tuhatta riviä dataa testisuoritusta kohden. SMLTK-lisäosan kanssa päädyttiin käyttämään pienemmän tiedoston dataa, sillä isomman tiedoston datamäärän kanssa tuli ongelmia prosessointiaikojen kanssa. Algoritmien kouluttaminen

ja käyttöönottaminen varasivat huomattavan määrän käytössä olleen virtuaalikoneen resursseista. Virtuaalikone oli jaettu eri kehittäjien kesken, joten tässä työssä ei haluttu vaikuttaa muiden kehittäjien työhön varaamalla liikaa resursseja.

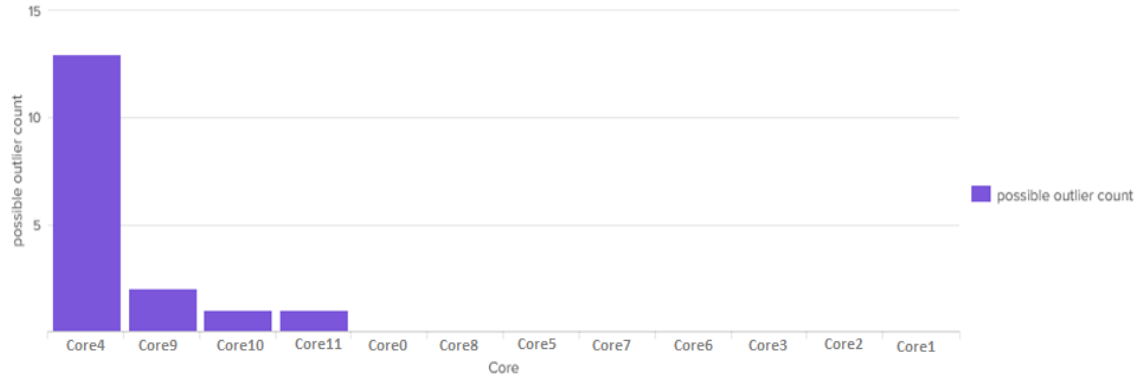
4.6.1 Poikkeamien tunnistaminen standardisoidulla normaalijakaumalla

Numeeristen poikkeamien tunnistaminen perustui standardisoidun normaalijakauman, keskimääräisen absoluuttisen poikkeaman tai iqr-menetelmän käyttämiseen tutkittavalle tiedolle. Tässä työssä päädyttiin käyttämään standardisoitua normaalijakaumaa. Kuviossa 18 on esitetty SPL-haku poikkeamien tunnistamiseksi standardisoitua normaalijakaumaa hyödyntäen.



KUVIO 18. SPL-hakukomento poikkeamien määrän selvittämiseksi normaalijakauman avulla

Kuvion 18 SPL-hakukomento laski ensin keskiarvon (avg) ja standardin normaalijakauman (stdev) eri suorittimien ydinten (Core) suhteen kokonaisprosessointiajan (total_process_time) kentän arvoja käyttäen, ja määritti ala- ja ylärajakynnykset poikkeamien tunnistamiseen kynnysarvon kertoimella 15. Tätä arvoa kasvattamalla poikkeamia löytyi vähemmän. Kynnysarvojen määrittämisen jälkeen luotiin ehto poikkeamien (isOutlier) tunnistamiselle, ja lopuksi laskettiin ja järjestettiin poikkeamien määrä ydinten suhteen. Kuviossa 19 on esitetty visualisaatio kuvion 18 hakukomennon tuloksista kynnysarvon kertoimella 30.



Core	possible outlier count
Core4	13
Core9	2
Core10	1
Core11	1
Core0	0

KUVIO 19. Pylväsdiagrammiesitys mahdollisten poikkeamien määrästä eri ydinten suhteen

SMLTK-lisäosan Detect Numeric Outliers -toiminnallisuus ei hyödyntänyt ainakaan suoraan mitään tunnettua algoritmia poikkeamien tunnistamiseen, vaan poikkeamien tunnistaminen perustui tilastotieteen keinoihin. Tässä menetelmässä ei tarvinnut esimerkiksi kouluttaa algoritmeja. Kuvion 18 hakukomento olisi siis toiminut ilman SMLTK-lisäosan asentamistakin. Toki SMLTK-lisäosassa tuleva graafinen käyttöliittymä mahdollisti SPL-hakukomennon rakentamisen ja suorittamisen käytännössä hiiren painalluksilla.

4.6.2 Poikkeamien tunnistaminen algoritmin avulla

Smart Outlier Detection -menetelmässä oli mahdollista hyödyntää olemassa olevia algoritmeja poikkeamien tunnistamiseen. Tässä työssä päädyttiin käyttämään DensityFunction-nimistä algoritmia, jonka pitäisi soveltua hyvin poikkeamien tunnistamiseen (25). Algoritmia voitiin käyttää joko suoraan poikkeamien tunnistamiseen komennolla *apply* tai se voitiin ensin kouluttaa uudelleen pienemmällä osalla dataa komennolla *fit*, minkä jälkeen koulutettua algoritmia voitiin hyödyntää poikkeamien tunnistamiseen. Kuviossa 20 on esitetty SPL-hakukomento algoritmin kouluttamiseksi ja tallentamiseksi uudeksi malliksi.

```
| inputlookup train_model_with_eventdata.csv
| fit DensityFunction "total_process_time" by "Core" dist=gaussian_kde threshold=0.001
     show_density=true show_options="feature_variables,split_by,params" into "trained_event_model"
```

KUVIO 20. SPL-komento algoritmin kouluttamiseksi ja käyttöön ottamiseksi

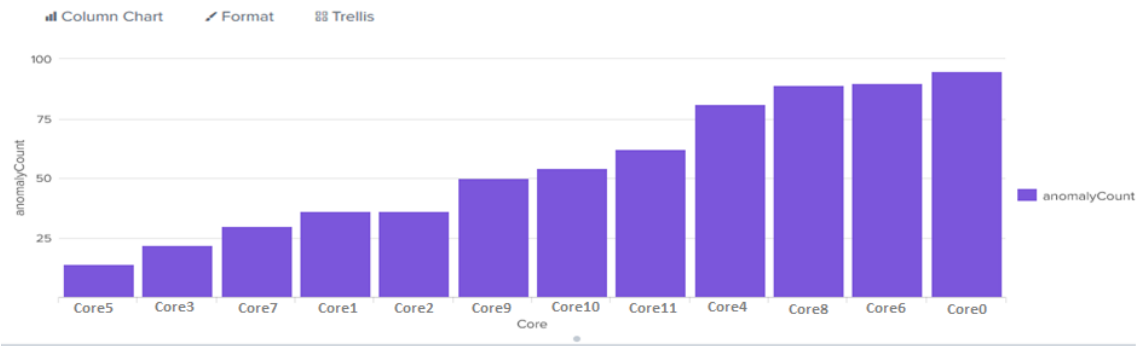
Kuviossa 20 fit-komennolle voitiin antaa distribution-argumentilla datan tilastollisen käyttäytymisen tapa. Tässä työssä päädyttiin käyttämään Gaussian KDE -menetelmää. Argumentille voitiin antaa myös arvo auto, jolla algoritmin pitäisi itse tunnistaa datan tilastollisen käyttäytymisen tapa.

Uudelleen koulutettua mallia voitiin käyttää koko indeksoidun datan poikkeamien tutkimiseen. Kuviossa 21 on esitetty, miten koulutettu koneoppimismalli otettiin käyttöön apply-komennolla. Apply-komento kävi kaikki tapahtumat läpi ja lisäsi tapahtumiin kentän IsOutlier. Tähän kenttään lisättiin arvo 1, mikäli tulos oletettiin poikkeavaksi. Etsimällä IsOutllier-kenttää arvolla 1 pystyttiin mahdolliset poikkeamat laskemaan ja listaamaan.

```
| inputlookup all_time_event_data.csv
| apply trained_event_model | search "IsOutlier(total_process_time)"="1.0"
| stats count as anomalyCount by "Core" | sort anomalyCount
```

KUVIO 21. SPL-komento koulutetun mallin käyttöönottamiseksi poikkeamien tunnistamiseen

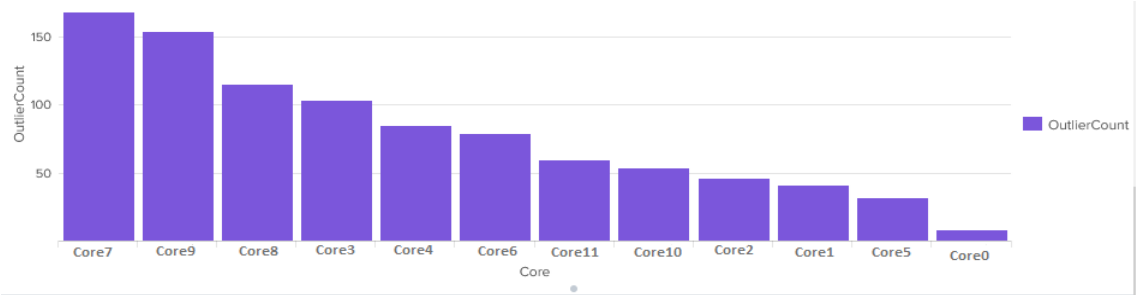
Kuvion 21 SPL-hakukomento palautti mahdolliset poikkeamat indeksoiduista tapahtumista. Algoritmin koulutuksessa käytettiin dist-argumentin arvona Gaussian KDE. Tällä asetuksella mahdollisia poikkeamia löytyi 659 kappaletta. Kuviossa 22 on esitetty kuvion 21 SPL-hakukomennon tuottamat tulokset eri ydinten suhteen threshold-argumentin arvolla 0.0001.



Core	anomalyCount
Core5	14
Core3	22
Core7	30
Core1	36
Core2	36
Core9	50
Core10	54
Core11	62
Core4	81
Core8	89
Core6	90
Core0	95

KUVIO 22. Mallin löytämät mahdolliset poikkeamat eri ydinten suhteen dist-arvolla Gaussian KDE

Dist-argumentin arvolla auto poikkeamia löytyi 945 noin 150 000 tapahtuman joukosta. Myös ytimistä löytyneet mahdolliset poikkeamat olivat eri suhteessa vaihdettaessa dist-argumentin arvoa. Kuviossa 23 on esitetty kuvion 21 SPL-hakukomennon tulokset dist-argumentin arvolla auto ja threshold-argumentin arvolla 0.0001 käänteisesti järjestettynä.



Core	OutlierCount
Core7	168
Core9	154
Core8	115
Core3	103
Core4	85
Core6	79
Core11	60
Core10	54
Core2	46
Core1	41
Core5	32
Core0	8

KUVIO 23. Mallin löytämät mahdolliset poikkeamat eri ydinten suhteen dist-arvolla auto

4.7 Johtopäätökset

Tämän opinnäytetyön tarkoituksena oli selvittää, saadaanko binäärimuotoisista lokitiedostoista tunnistettua langattomien verkkoliikennelaitteiden komponenttien poikkeavaa käytöstä ja sen mahdollista kestoa mahdollisesti koneoppimista hyödyntäen, eli löydetäänkö suoritusaikalokitiedostojen datan perusteella systeemistä mahdollisia pullonkauloja. Splunk-ohjelmistolla hakujen automaatio ja koneoppimisen hyödyntäminen oli melko suoraviivaista. Splunk ei kuitenkaan osannut indeksoida binäärimuotoista dataa, joten data täytyi ensin parsia Splunk-ohjelmiston ymmärtämään muotoon. Edellä esiteltyjä menetelmiä ja Splunk-ohjelmistoa hyödyntäen mahdollisia yksittäisiä numeerisia poikkeamia eri prosessien kokonaiskestossa eri prosessoriytimissä pystyttiin tunnistamaan suoritusaikalokitiedostoista ilman koneoppimistakin, sillä yleensä hyvin suunniteltu SPL-hakukomento riitti kattamaan tarvittavat tiedot. Toki tämä edellytti, että haettavasta datasta löytyi tarvittavat tiedot ja kentät. Tässä työssä mahdollisten poikkeamien kestoa ei kuitenkaan pystytty mittaamaan.

Tähän työhön kerättiin dataa ajamalla yhtä testitapausta 20 kertaa onnistuneesti, joten oletus oli, että poikkeamia löytyisi muutamia tai ei ollenkaan. Tämän työn koneoppimismenetelmillä ja anomalydetection-komennolla löytyneitä poikkeamia käsiteltiin siksi mahdollisina poikkeamina, sillä tuloksista ei pystytty suoraan päättelemään komponentin poikkeavaa käyttäytymistä. Langattoman verkkoliikennelaitteen toimintaa ajatellen yksittäisellä prosessin kestolla ei oikeastaan ole juurikaan merkitystä, vaan ratkaisevaa on koko prosessien ketjun pituuden selvittäminen ja niistä mahdollisten poikkeamien löytäminen. Toisin sanoen täytyisi ensin selvittää verkkoliikennelaitteessa kulkevan datapaketin reitti eri prosessien läpi ja tähän reittiin kulutettu aika.

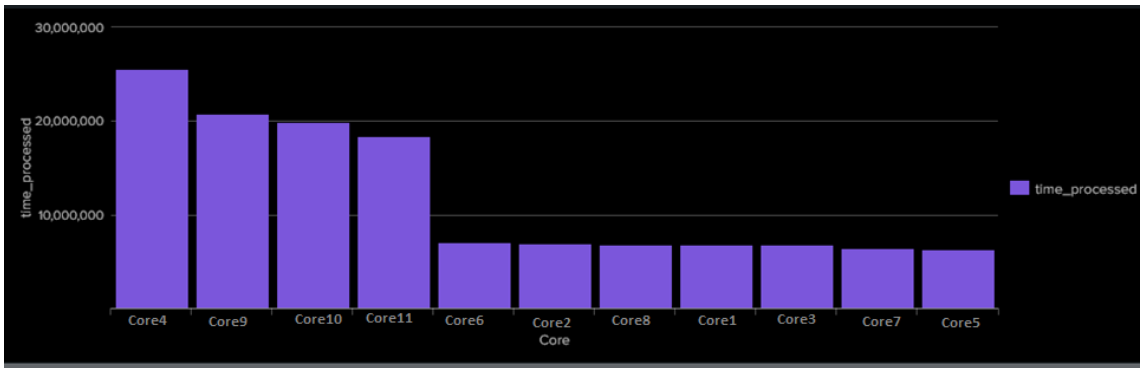
Anomalydetection-komentoa kokeiltiin tässä työssä kolmella eri menetelmällä. Tämän työn datan ja testisuoritusten perusteella z-piste-menetelmä palautti vähiten mahdollisia poikkeamia, joten suoritusaikadatan numeerisia arvoja sisältävien kenttien poikkeamien tunnistamiseen, sopi se menetelmistä parhaiten. Menetelmä myös mahdollisti poikkeamien tunnistamiskynnyksen säätämisen, joten se sopisi myös mahdollisesti muiden lokitiedostojen numeeristen arvojen poikkeamien tutkimiseen. Iqr-menetelmän voidaan katsoa olevan enemmänkin dataa valmisteleva menetelmä, sillä se palautti noin puolet alkuperäisestä tutkimusjoukosta. Iqr-menetelmää voisi esimerkiksi käyttää ensin SPL-haussa tutkittavalle datalle, jonka jälkeen suoritetaan vielä tuloksille toinen SPL-haku poikkeamien tunnistamiseksi esimerkiksi koneoppimisen menetelmiä hyödyntäen.

Tässä työssä ei pystytty laskemaan datapaketin käyttämää kokonaisaikaa prosessireitin läpi, sillä suoritusaikalokitiedostoja täytyisi muokata sisältämään enemmän yksilöivää tietoa prosessoinnissa olevasta paketista. Ilman tätä tietoa Splunk-ohjelmistolla, tai millään muullakaan tietokannalla, ei todennäköisesti voida laskea datareittiin käytettyä kokonaisaikaa. Toisaalta testin pitäisi itsessään epäonnistua, mikäli datapaketti ei pysy tuotteen määrittelyssä annetussa aikaikkunassa.

Koneoppimista ei pystytty tässä työssä täysin hyödyntämään kaiken datan läpikäymiseen. Isompaa datatiedostoa läpikäytäessä täytyi ensin muuttaa useampaa konfiguraatiotiedostoa Splunk-ohjelmistosta, jotta algoritmeja päästiin yleensä kouluttamaan datalla. Algoritmin kouluttaminen datalla onnistui vielä kohtuudella, mutta varsinainen datan tutkiminen kuormitti kehittäjien yhteiskäyttöön jaettua virtuaalikonetta hyvin paljon. Tämän seurauksena tässä työssä päädyttiin tutkimaan pienemmän tiedoston dataa koneoppimisen avulla, sillä tässä työssä ei haluttu vaikuttaa muiden kehittäjien työhön varaamalla liikaa resursseja.

Pienemmälle indeksoiduista csv-tiedostoista pystyttiin toteuttamaan algoritmin kouluttaminen. Tätä koulutettua algoritmia hyödynnettiin poikkeamien tunnistamiseksi datasta, mutta kuten anomaly-detection-komennon tapauksessa poikkeamat olivat mahdollisia yksittäisiä poikkeamia.

Splunk-ohjelmistolla onnistuttiin luomaan SPL-hakuja, jotka laskivat ja visualisoivat tulokset jaettujen prosessien määristä eri prosessoriytimien kesken. Splunk-ohjelmistolla luotiin myös onnistuneesti SPL-hakuja, jotka laskivat ja visualisoivat käytetyn kokonaisprosessointiajan eri prosessoriytimien kesken, kuten on esitetty kuviossa 24. Core4:n huomattiin olevan kuormitetuin ydin. Core0 suodatettiin tässä visualisaatiossa hakutuloksista pois, sillä Core0:n tiedettiin olevan kuormitetuin ydin erilaisten käynnistymissekvenssien takia.



Core	time_processed
Core4	25588077
Core9	20818516
Core10	19924227
Core11	18369699
Core6	7057048
Core2	6924893
Core8	6837602
Core1	6837323
Core3	6789048
Core7	6371894
Core5	6357863

KUVIO 24. Jaettu prosessointiaika eri prosessoriytimissä

Tässä työssä onnistuttiin Splunk-ohjelmiston avulla tunnistamaan suoritusaikalokitiedostoista erittäin kuormitetut prosessoriytimet (kuvio 24) ilman koneoppimistakin. SPL-hakua edelleen muokkaamalla pystyttiin tunnistamaan Core4-prosessoriydintä kuormittaneet yksittäiset prosessit, ja toimeksiantajalle pystyttiin ehdottamaan prosessien mahdollista optimoimista. Pienin muutoksin esitettyihin SPL-hakukomentoihin vastaavia keinoja voitaisiin hyödyntää mahdollisesti muidenkin lokitiedostojen läpikäymiseen Splunk-ohjelmistolla.

5 POHDINTA

Tämän opinnäytetyön tavoitteena oli luoda toimiva reitti jatkuvasta integraatiosta tulevien lokitiedostojen lisäämiseksi Splunk-ohjelmistolle indeksoitavaksi sekä kokeilla, miten Splunk-ohjelmistoa ja koneoppimista voitiin hyödyntää binäärimuotoisista lokitiedostoista saatujen ytimien suoritusaikojen läpikäymiseen ja mahdollisten pullonkaulojen löytämiseen. Tähän tavoitteeseen ei täysin tässä opinnäytetyössä päästy, sillä Splunk-ohjelmisto ei ollut käytössä osastolla, jolle tämä työ tehtiin. Toimivaa reittiä ei siis saatu luotua jatkuvasta integraatiosta Splunk-palvelimelle. Reitti kuitenkin suunniteltiin ja toteutettiin muilta osin valmiiksi, mikäli osastolla halutaan Splunk-ohjelmisto ottaa myöhemmin käyttöön. Binäärimuotoisten tiedostojen purkaminen ja parsiminen toteutettiin onnistuneesti, ja työssä onnistuttiin myös havainnollistamaan, miten Splunk-ohjelmistolla voitiin etsiä yksittäisiä poikkeamia indeksoidusta datasta koneoppimisen avulla. Työssä onnistuttiin myös SPL-hakukomentojen avulla havainnollistamaan eri prosessoriytimien käyttämä kokonaisprosessointiaika ja edelleen tunnistamaan kuormitetut prosessoriytimet ja edelleen kuormituksen aiheuttavat prosessit.

Splunk-ohjelmiston huomattiin käyttävän melko paljon resursseja kehittäjien kesken jaetusta virtuaalikoneesta käytettäessä lisäosien mahdollistamia eri koneoppimismenetelmiä. Varsinkin *fit*-komennon käyttäminen laajemmalle datalle kulutti huomattavan paljon virtuaalikoneen resursseja. Mikäli osastolla päätettäisiin Splunk-ohjelmisto ottaa käyttöön, tulisi sille mielestäni varata täysin oma yksikkö riittävällä laskentateholla. Mikäli koneoppimisen menetelmiä halutaan edelleen hyödyntää osastolla, täytyisi yksikön teho mitoittaa sen mukaisesti tai vähintäänkin välttää tekemästä SPL-hakuja, jotka kohdistuisivat kaikelle indeksoidulle datalle. Mielestäni Splunk-ohjelmistolla kuitenkin onnistuttiin löytämään tärkeää tietoa ilman koneoppimistakin, kunhan huolehditaan, että etsittävästä datasta löytyvät SPL-haun tarvitsemat oleelliset kentät.

Splunk oli minulle täysin tuntematon ohjelmisto ennen tätä opinnäytetyötä. Tämän työn aikana opin ottamaan Splunk-ohjelmiston käyttöön ja ymmärsin, miten indeksoidusta datasta löydetään haluttuja asioita SPL-syntaksin avulla. Opin myös hyödyntämään Splunk-ohjelmistoa poikkeamien tunnistamisessa, ottamaan eri koneoppimismenetelmiä tarjoavia lisäosia Splunk-ohjelmistossa käyttöön sekä hyödyntämään koneoppimismenetelmiä indeksoitujen tapahtumien käsittelyssä.

Tämä opinnäytetyö syvensi osaamistani tietokannoista. Ymmärsin, miten tärkeä yksilöivä kenttä, kuten esimerkiksi id-kenttä, on tietokantahakujen kanssa. Ilman tätä kenttää tietojen yhdistäminen ja toivotun tuloksen saaminen hauista oli lähes mahdotonta. Tässä työssä yksittäisen datapaketin reitin kestoa prosessien läpi ei pystytty mittaamaan, sillä suoritusaikalokitiedostoista puuttui tieto kulloinkin prosessissa käsittelyssä olevasta paketista.

Jatkokehitysideana suoritusaikalokitietojen keräystä voisikin muuttaa mahdollisuuksien mukaan siten, että kulloinkin käsittelyssä olevasta paketista lisättäisiin tieto suoritusaikalokeihin. Näin Splunk-ohjelmistolla pystyttäisiin melko helposti selvittämään paketin käyttämä kokonaisaika eri prosessien läpi ja vertaamaan, pysyykö paketin käyttämä kokonaisaika eri prosessien läpi tuotteen määrittelyssä annettujen aikaikkunoiden sisässä.

Kaiken kaikkiaan Splunk-ohjelmistolla suoritusaikalokitiedostojen läpikäyminen oli huomattavasti helpompaa ja nopeampaa kuin tiedostojen manuaalinen läpikäyminen, ja hakujen automatisoiminen ja mahdollisten hälytysten lähettäminen poikkeavuuksista onnistui Splunk-ohjelmistossa käytännössä hiiren painalluksilla. Splunk-ohjelmiston käyttöönottaminen toimeksiantajan osastolla toisi todennäköisesti kustannussäästöjä toimeksiantajalle vapauttamalla henkilöresursseja ohjelmistokehitystyöhön.

LÄHTEET

1. Kauppalehti 2023. Yrityshaku. Hakupäivä 24.11.2023. https://www.kauppalehti.fi/yritykset/yritys/nokia+oyj/0112038-9

2. Reshma, Amina 2022. What Does a Bottleneck Mean in Computer Science? Byjusfutureschool. Hakupäivä 05.01.2024. https://www.byjusfutureschool.com/blog/what-does-a-bottleneck-mean-in-computer-science/

3. Coursera 2023. What Is Machine Learning? Definition, Types, and Examples. Hakupäivä 05.12.2023. https://www.coursera.org/articles/what-is-machine-learning

4. Tuominen, H., Neittaanmäki, P., Niinimäki, E., Pölönen, I., Rautiainen, I., Äyrämö, S., Ruohonen, T., Nyrhinen, R., Ojalainen, A., Vähäkainu, P. & Äyrämö, S. 2019. Tekoälyn perusteita ja sovelluksia. Jyväskylä: Jyväskylän yliopisto. JYX Digital Repository. Hakupäivä 07.12.2023. https://jyx.jyu.fi/handle/123456789/64975

5. Sas Insight 2023. Machine Learning. Hakupäivä 07.12.2023. https://www.sas.com/en_ae/insights/analytics/machine-learning.html

6. Aurélien, Géron 2022. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Chapter 1. The Machine Learning Landscape. Third edition. Sebastopol: O'Reilly Media, Inc. Hakupäivä 05.12.2023. O'Reilly for Higher Education. Vaatii käyttöoikeuden.

7. Nvidia 2023. K-Means Clustering Algorithm. Hakupäivä 08.01.2024. https://www.nvidia.com/en-us/glossary/data-science/k-means/

8. Holopainen, Ville 2022. Koneoppimisen hyödyntäminen vesijohtoverkostojen vuotojen hallinnassa – Systemaattinen kirjallisuuskartoitus. Pro gradu -tutkielma. Jyväskylä: Jyväskylä yliopisto. JYX Digital Repository. Hakupäivä 07.01.2024. https://jyx.jyu.fi/handle/123456789/84073

9. IBM 2023. What is data mining? Hakupäivä 10.01.2024. https://www.ibm.com/topics/data-mining

10. Sakshi, Gupta 2021. What Is the Best Language for Machine Learning? Springboard. Hakupäivä 05.01.2024. https://www.springboard.com/blog/data-science/best-language-for-machine-learning/

11. Ostrowska, Kamila 2022. A Brief History of Python. LearnPython. Hakupäivä 10.11.2023. https://learnpython.com/blog/history-of-python/

12. Python 2023. What is Python? Executive Summary. Hakupäivä 10.11.2023. https://www.python.org/doc/essays/blurb/

13. Python geeks 2023. Python Interpreter. Hakupäivä 13.11.2023. https://pythongeeks.org/interpreter-in-python/

14. Swimm Team 2023. 5 tips for creating self-documenting code. Hakupäivä 27.11.2023. https://swimm.io/learn/documentation-tools/tips-for-creating-self-documenting-code

15. Kidd, Chrissy 2022. What Is Splunk & What Does It Do? An Introduction to Splunk. Splunk. Hakupäivä 13.11.2023. https://www.splunk.com/en_us/blog/learn/what-splunk-does.html

16. Splunk 2023. Simple and predictable pricing for security professionals. Hakupäivä 13.11.2023. https://www.splunk.com/en_us/products/pricing/cyber-security.html

17. Splunk 2023. Distributed Deployment Manual. How data moves through Splunk deployments: The data pipeline. Hakupäivä 14.11.2023. https://docs.splunk.com/Documentation/Splunk/9.1.1/Deploy/Datapipeline

18. Tibco 2023. What is Event Processing? Hakupäivä 14.11.2023. https://www.tibco.com/reference-center/what-is-event-processing

19. Splunk 2023. SPL2 Search Reference: Introduction. Hakupäivä 14.11.2023. https://docs.splunk.com/Documentation/SCS/current/SearchReference/Introduction

20. Splunk 2023. Search Reference: Anomalydetection. Hakupäivä 07.01.2024. https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Anomalydetection

21. Statistics How To 2023. Interquartile Range (IQR): What it is and How to Find it. Hakupäivä 07.01.2024. https://www.statisticshowto.com/probability-and-statistics/interquartile-range/#interquartile%20range%20by%20hand

22. Statistics How To 2023. What is a Two Proportion Z-Interval? Hakupäivä 07.01.2024. https://www.statisticshowto.com/two-proportion-z-interval/

23. Statistics How To 2023. Relative Frequency Histogram: Definition and How to Make One. Hakupäivä 07.01.2024. https://www.statisticshowto.com/relative-frequency-histogram/

24. Splunk 2023. Splunk Enterprise Product Features. Hakupäivä 11.12.2023. https://www.splunk.com/en_us/products/splunk-enterprise-features.html

25. Splunk 2024. Machine Learning (ML). Quick Reference Guide. Hakupäivä 25.01.2024. https://docs.splunk.com/images/3/3f/Splunk-MLTK-QuickRefGuide-2019-web.pdf

26. Splunk 2023. Custom Anomaly Detection with Splunk IT Service Intelligence and Machine Learning Toolkit v3.2 – Part 2. Hakupäivä 07.01.2024. https://www.splunk.com/en_us/blog/platform/custom-anomaly-detection-with-splunk-it-service-intelligence-and-machine-learning-toolkit-v3-2-part-2.html

27. Ainslie-Malik, Greg 2022. Getting Started with Machine Learning at Splunk. Splunk. Hakupäivä 07.01.2024. https://www.splunk.com/en_us/blog/platform/getting-started-with-ml-at-splunk.html

28. Splunk 2023. Knowledge Manager Manual. Introduction to lookup configuration. Hakupäivä 07.01.2024. https://docs.splunk.com/Documentation/Splunk/9.1.3/Knowledge/Addfieldsfromexternaldatasources

29. Atlassian 2023. What Is DevOps? Hakupäivä 08.02.2024. https://www.atlassian.com/devops

30. Barker, Robert 2020. The Uses and Benefits of Splunk in Continuous Integration. Oulun ammattikorkeakoulu. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö. Hakupäivä 23.01.2024. https://www.theseus.fi/bitstream/handle/10024/333929/Robert_Barker_Thesis.pdf?sequence=2

31. Splunk 2023. Splunk Enterprise 9.1.3. Hakupäivä 24.01.2024. https://www.splunk.com/en_us/download/splunk-enterprise.html