Tampereen ammattikorkeakoulu

Sijaintitieto datakäsittelyssä ja koneoppimisessa

Ordobike-palvelun pyörävarkausdatan hyötykäyttö ja soveltaminen uusissa ohjelmakomponenteissa

Juho Mäntynen

OPINNÄYTETYÖ
Marraskuu 2022

Dataosaamisen ja tekoälyn ylempi tutkinto-ohjelma
Insinööri (ylempi AMK)

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Dataosaamisen ja tekoälyn ylempi tutkinto-ohjelma
Insinööri (ylempi AMK)

MÄNTYNEN, JUHO:
Sijaintitieto datakäsittelyssä ja koneoppimisessa
Ordobike-palvelun pyörävarkausdatan hyötykäyttö ja soveltaminen uusissa ohjelmakomponenteissa

Opinnäytetyö 50 sivua
Marraskuu 2022

Polkupyörävarkaudet lisääntyvät Suomessa vuosi vuodelta, pyöräilyn suosion ja sen lieveilmiöiden myötä. Tehokasta tapaa varkauksien ehkäisemiseen ei ole, ja viranomaiset eivät voi tehokkaasti puuttua ongelmaan resurssien vähäisyyden takia. Tekoäly ja koneoppiminen tarjoavat mahdollisuuksia varkauksien analysoimiseen ja ehkäisemiseen, käyttäen hyödykseen niiden sijaintitietoja.

Opinnäytetyön raportissa käydään läpi sijaintidatan käytön vaiheet datan käsittelystä tekoälyn koneoppimisratkaisuun, ja saatujen lopputulosten jalostamiseen ohjelmistopalvelussa käytettäväksi. Sovelluskohteena toimii opinnäytetyön tekijän vapaa-ajan projektina kehitetty pyörävarkauksien ilmoittamiseen tarkoitettu ohjelmistopalvelu ja sen data.

Työssä käsitellään sijaintitiedon luonne, ja annetaan esimerkkejä, miten sitä on käytetty informaatioteknologian ja tekoälyn saralla. Samalla yritetään löytää ideoita, miten käyttää sijaintidataa opinnäytetyön sovelluksessa.

Raportissa esitellään myös sovelluskohteena oleva pyörävarkauspalvelu, ja käydään läpi sen datan luonne, ja siitä poimittava, työn suhteen relevantti, avaindata. Esitellään toteutukseen valitut sovellustavat, ja suunnitelmat uusien ohjelmatoiminnallisuuksien suhteen.

Lopuksi raportoidaan työn aikana tehdyt uudet käytännön prototyyppitoteutukset, ja pohditaan opinnäytetyön eri vaiheita ja tuloksia.

Asiasanat: tekoäly, koneoppiminen, sijaintidata

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Master's Degree in Data Expertise and Artificial Intelligence

MÄNTYNEN, JUHO:
Location Data: Data Handling and Machine Learning
Utilization and Application of Ordobike Bike Theft Data in New Software Components

Master's thesis 50 pages
November 2022

Bike thefts in Finland are increasing yearly due to cycling becoming a more popular way of travel. However, there are no efficient ways to prevent these thefts, and authorities are short on resources to tackle the problem. Artificial intelligence and machine learning offer opportunities to analyze and prevent these thefts by utilizing their location information.

This thesis report takes the reader through different phases of using location data, from analyzing and handling the data and using it in machine learning, to applying the results in a software service as concrete new functionality. The target of the application is a bike theft reporting software service, developed by the thesis' writer as a personal project, and its data.

The report explores the nature of location data and provides examples on how it has been applied in the areas of information technology and artificial intelligence. At the same time, possible location data application ideas in the context of the thesis are explored.

The report also introduces a bike theft reporting service on a software level, and presents the nature of its data, and the relevant key information, ie. the location data, to be extracted from it. The chosen new practical functionalities to be developed to the service are also introduced.

Finally, the new developed proof of concept -software functionalities are explored, and deliberations about the results and different phases of the thesis are made.

Keywords: artificial intelligence, machine learning, location data

SISÄLLYS

LYHENTEET JA TERMIT

backend	Palvelinohjelmisto, joka tarjoaa datarajapinnan (REST-rajapinta) ohjelman web-käyttöliittymälle.
big data	Yleistermi nykypäivän tietojärjestelmien tuottamaan isoon datamäärään.
CSV	Comma separated values. Tapa esittää suuri määrä tietoa tekstitiedostossa. CSV-tiedosto koostuu riveistä, joissa eri tiedot on eroteltu pilkuilla (eng. comma).
diskreetti data	Datan arvoina voi olla vain ennalta määriteltyjä vaihtoehtoja, kuten esimerkiksi puhuttu kieli tai kansallisuus.
ei-diskreetti data	Datalla on arvoja, jotka eivät ole ennalta määriteltyjä, kuten esimerkiksi lämpötila tai pituus. Kutsutaan toisinnaan myös jatkuvaksi dataksi.
endpoint	REST-rajapinnan yksi päätepiste, johon asiakassovellus voi ottaa yhteyttä.
framework	Yleensä jonkin tiettyyn teemaan, esim. käyttöliittymän rakentamiseen tai koneoppimiseen, liittyvä kokoelma ohjelmakoodeja, jonka ohjelmistokehittäjä voi ottaa projektissaan käyttöön.
frontend	Web-käyttöliittymäsovellus.
heatmap	Tiheyskartta. Visualisointikeino, jolla näytetään jonkin ilmiön tiheyttä, yleensä aluekartan päällä. Tiheimmät alueet voidaan värjätä esimerkiksi punaisiksi.
JSON	Sanoista javascript object notation. Rakenteellisen datan tallennustapa tekstimuotoon.
ML	Machine learning. Koneoppiminen.
on-demand-ajotapa	Jonkin ohjelman ajo ei-automaattisesti, eli silloin kun ajoa pyydetään erikseen, esimerkiksi manuaalisesti käynnistäminen käyttäjän toimesta.
PoC, proof of concept	Ketterässä ohjelmistokehityksessä käytetty termi eräänlaisesta prototyyppiohjelmasta. Tällainen ohjelma todistaa jonkinlaisen konseptin toimivuuden, niin kuin englanninkielinen termi indikoikin. Tällainen ohjelma ei

	ole lopullinen tuotteistettu versio, ja se pitää sisällään vain oleellisimmat toiminnallisuudet.
REST-rajapinta	Web-sovelluskehityksen parissa yleisesti käytetty tiedonsiirtomenetelmä backendin ja frontendin välillä.

1 JOHDANTO

Pyöräilyn suosio Suomessa on kasvanut tasaisesti sitä mukaan, kun siitä on tullut taloudellisesti kannattavampaa mm. kallistuneiden autoilukustannusten, sähköpyörävalikoiman lisääntymisen, sekä kannustimien, kuten sähköpyörien työsuhde-etujen lisäämisten, myötä. Taustalla pyöräilyn suosion kasvussa on myös ilmastoystävällisemmän liikkumisen megatrendi, johon myös julkishallinnot yrittävät myötävaikuttaa lisäämällä pyöräilyn edellytyksiä kaupunkien pyöräilyinfrastruktuuria parantamalla (Liikenne- ja viestintäministeriö, 2021).

Pyöräilyn kasvu on johtanut myös polkupyörävarkauksien lisääntymiseen (Palmgren, 2021). Itse pyörien määrän kasvun lisäksi vaikuttavana tekijänä on myös päihdeongelmaisten määrän lisääntyminen kaupungeissa, sillä monesti kyseisen ihmisryhmän suorittamat rikokset ovat pyörävarkauksia (Von Bell, 2020).

Vuonna 2019 kehitettiin vapaa-ajan ohjelmistoprojektina Ordobike-palvelu, jonka on määrä helpottaa pyörävarkauksien selvittämistä. Palvelun käyttäjät voivat lisätä siihen varastettuja pyöriä ja niiden tunnistetietoja, ja varkauden yksityiskohtia, kuten sijainnin ja tapahtuma-ajan, sekä lisätä hylättyinä löytyneitä pyöriä, joita käyttäjät ovat saattaneet löytää esimerkiksi maastosta tai katujen varsilta.

Sijaintidatan sovellusmahdollisuudet tekoäly- ja koneoppimiskäytössä ovat tiedostettu alalla jo jonkin aikaa, ja Ordobike-palvelun pyörävarkaustapahtumista kerääntyvää sijaintidataa on myös mahdollista hyödyntää, ja saada sen kautta lisäarvoa tuottavia ominaisuuksia palveluun. Tämä opinnäytetyö käsittää eri vaiheet palvelun sijaintidatan tekoälyhyödyntämisestä käytännön toteutuksessa, ja raportissa selostetaan eri vaiheet ja niiden tulokset. Työn päätavoitteena on saada yleiskäsitys siitä, mitä eri vaiheita tekoälytoiminnallisuuden käyttöönotto sijaintidatasovelluksessa pitää sisällään.

Raportin toisessa kappaleessa käydään läpi sijaintidatan yleinen perusluonne, sekä annetaan esimerkkejä, miten sitä voidaan käyttää tekoälysovelluksissa.

Kolmannessa kappaleessa annetaan yleiskuvaus Ordobike-palvelusta, ja millaista sijaintadataa se tuottaa. Kappaleessa asetetaan myös tavoitteet, miten palvelun sijaintidataa halutaan hyödyntää, ja millaisia lisäominaisuuksia palveluun halutaan tuottaa tekoälyä soveltamalla.

Raportin neljäs kappale sisältää kuvaukset valituista sovellustavoista. Kappaleessa kuvataan käytettäväksi valitut koneoppimistavat, ja niihin liittyvät käsitteet, sekä teoriatasolla ohjelmistoon luotavat uudet komponentit ja toiminnallisuudet, niiden roolit, sekä suhteet muihin ohjelmiston komponentteihin.

Viidennessä kappaleessa esitellään syntyneet käytännön toteutukset, ja Ordobike-palveluun lisätyt ominaisuudet.

Viimeinen, kuudes, kappale sisältää loppumietelmiä opinnäytetyön eri työvaiheista ja tuloksista, sekä mietteitä, joita heräsi sen lopussa.

Koska työn aihe on tieto- ja ohjelmistotekniikan alaa, raportti pitää sisällään paljon englanninkielisiä termejä ja lyhenteitä.

2 SIJAINTIDATA JA TEKOÄLY

2.1 Spatiotemporaalinen paikkatieto ja koordinaattijärjestelmä

Paikkatieto kuvaa yleensä jonkin määritellyn alueen, kuten esimerkiksi luontoalueen tai rakennetun ympäristön kohteita ja niiden ominaisuuksia, mutta sillä voidaan myös kuvata mitä tahansa toimintaa tai ilmiötä, jonka sijainti tunnetaan. Esimerkiksi tämän opinnäytetyön viitekehyksessä paikkatiedon voi käsittää kuvaavan pyörävarkaustoiminnan paikkatietoja.

Paikkatietokohteen sijainnin määrittää sen sijaintitieto. Sijaintitieto voi olla koordinaatti, osoite, tai vaikkapa tunnettu paikannimi, kuten kaupunki tai paikkakunta. Paikkatiedon ei tarvitse myöskään aina indikoida maantieteellistä sijaintia maapallolla, kuten yleisimmin asian laita on, vaan se voi olla vaikkapa maalipotkun sijainti jalkapallokentällä tietyllä peliajan hetkellä (Anzer & Bauer, 2021). Tässä opinnäytetyössä käytettävä sijaintidata on koordinaattimuotoista, ja siinä kuvataan tapahtuman sijaintia maapallolla.

Kokonaisturvallisuuden sanaston (2017) mukaan spatiotemporaalinen paikkatieto on paikkatietoa, joka huomioi myös ajan. Ordobike-palvelun paikkatieto on spatiotemporaalista paikkatietoa, koska siinä pyörävarkaustapahtuma sisältää sijaintitiedon lisäksi myös varkauden arvioidun ajanhetken. Voidaan myös sanoa pyörävarkausdatan olevan ei-diskreettiä, eli jatkuvaa dataa, sillä varkauksien sijainnit ja ajat ovat vaihtelevia, eikä niille ole ennalta määriteltyjä rajoitettuja arvoja.

Jos poissuljetaan perinteiset postiosoitejärjestelmät, luultavasti yleisin maantieteellistä sijaintia indikoimaan tarkoitettu järjestelmä on jokin muunnelma maantieteellisestä koordinaattijärjestelmästä. Tällaisessa järjestelmässä maapallo on jaettu kahteen eri pallonpuoliskoon, pohjois- ja eteläsuunnassa päiväntasaajalta, sekä itä- ja länsisuunnassa Lontoon Greenwhichistä. Koordinaatteina toimivat pituusaste, joka ilmaisee Greenwhichin ja kohdesijainnin välistä kulmaeroa itä- ja länsisuunnassa, sekä leveysaste, joka ilmaisee kulmaeroa päiväntasaajasta pohjois- ja eteläsuunnassa.

Tietotekniikassa yleisin koordinaattijärjestelmä on luultavasti alun perin Yhdysvaltain puolustusministeriön kehittämä WSG84-järjestelmä (kirjainlyhenne sanoista World Geodetic System). Sitä käyttävät standardina mm. yleisesti käytetty GPS-järjestelmä, ja sitä kautta lähes kaikki älylaitteiden GPS-paikannusta hyödyntävät ohjelmat, esimerkiksi Google Maps. Koska ohjelmistokehityksessä on tavallista käyttää englanninkielisiä termejä, on sopivaa todeta, että leveys- ja pituusasteista puhutaan englannin kielessä sanoilla latitude ja longitude. Esimerkiksi Tampereen kaupungin keskustan koordinaatit ovat latitude: 61,4979008 ja longitude: 23,7624057. Myös Ordobiken pyörävarkaustapahtumien sijainti on merkitty samantyyppisillä koordinaateilla.

2.2 Sijaintidata ja tekoäly

Sijaintidatan analysointi ja hyötykäyttö on ollut laajaa jo ennen tekoäly-teeman yleistymistä. Business intelligence -alalla sille on oma alakäsitteensä, location intelligence (LI), jolla tarkoitetaan erilaisen sijainti- ja ympäristödatan hyödyntämistä jonkin organisaation toiminnassa (Polaris Intelligence, 2020; Esri, n.d.). Organisaatio voi olla vaikkapa julkinen taho, kuten kaupunki, joka tekee väestönkartoitusta, käyttäen hyväksi eri sijaintien kotitalouksien tietoja, tai kaavoitustutkimusta, katsoen mille alueelle uusin asuinalue on paras sijoittaa, vaikkapa kulkuyhteyksien ja olemassa olevien palveluiden perusteella. Kiinteistönvälitysalan yritykset voivat käyttää sijaintidataan liitettyjä myytyjen asuntojen hintatietoja määrittääkseen sopivan myyntihinnan kohteilleen, tai tutkiakseen asuntojen hintakehitystä asuinalueittain.

Tekoälyn esiintulon myötä paikkatietodataa on pyritty hyödyntämään myös sen parissa. Suosittu käyttökohde on ns. sosiaalinen paikkatietodata, eli paikkatieto, johon liittyy ihmisten tekemiset, ja sen analysointi ja ennustamiskäyttö. Esimerkkinä ihmisten liikkumisdata kaupungissa, josta yritetään löytää erilaisia kaavoja, ja ennustaa, missä palveluissa ihmiset käyvät (Krueger & Han, 2019). Myös koronapandemian iskettyä, on sijaintidataa ja tekoälyä hyötykäytetty taudin leviämisen ennustuksessa, ja löytämään taudin potentiaalisia tartuntapesäkkeitä (Polaris Intelligence, 2020).

Myöskin eräs mielenkiintoinen tieteenala, jossa sijaintidatan käyttö on keskeisessä roolissa, on rikoksien maantieteellinen profilointi, jossa yritetään päätellä tietoja rikoksien tekijöistä rikospaikka- ja aikatietojen perusteella, tai profiloida eri kaupunkialueita rikollisuuden määrän mukaan (Texas State University, n.d.). Tämä tieteenala on ollut olemassa jo kauan ennen tekoälyn ja sijaintipainotteisen big datan esiintuloa. Ennen digitalisoitumista rikostutkijat ovat käyttäneet perinteisiä paperikarttoja tutkimuksissaan.

Tämän työn puitteissa, tieteellisten artikkelien tutkinta antoi joitakin heikkoja pohjaideoita datan käyttötavoille, mutta tutkimusta tehdessä, olivat ideat halutuista toteutuksista oikeastaan selkiytyneet jo valmiiksi, ilmankin niitä. Tahtotila oli toteuttaa heatmap-tyylinen, informatiivinen karttanäkymän, joka kertoisi mille alueille pyörävarkaudet ovat keskittyneet. Alustavat toteutusideat esitellään kappaleessa *3.4 Uudet tekoälypohjaiset ominaisuudet*, nykyisen Ordobike-palvelun tarkemman esittelyn jälkeen.

Artikkelit eivät myöskään tarjonneet vinkkejä toteutuksen teknisiin ratkaisuihin, joten niitä varten käännyttiin teknisempien, yleensä ohjelmistotekniikan alueen, web-esimerkkiartikkelien etsintään ja selaamiseen. Myös ratkaisutapa, jota käytetään tässä opinnäytetyössä, löytyi tämän etsinnän tuloksena. Oikeille jäljille johti tekoälyyn ja koneoppimiseen erikoistuneen SciKit Learn -koodikirjaston esimerkkiartikkeli (Vanderplas, n. d.), jossa käytettiin kirjaston ydinestimoinnin työkaluja kahden eläinlajin keskittymien havaitsemiseen sijaintitiedoista. Tämä ratkaisutapa, sekä teoriaa siihen liittyvistä koneoppimisen osa-alueista, esitellään tarkemmin kappaleessa *4 Valitut sovellustavat*.

3 KOHDESOVELLUS: DATA, JA TAVOITTEET

3.1 Ordobike-palvelu

Odrobike on vuonna 2019 julkaistu, vapaa-ajan ohjelmistoprojektina kehitetty, web-palvelu. Palvelu rakentuu kahden eri tapahtumatyypin ympärille: pyörän varkaustapahtuma ja löytymistapahtuma. Ideana on, että käyttäjät voivat lisätä palveluun pyörävarkauksien lisäksi myös pyörien löytymisiä, jos henkilö on esimerkiksi löytänyt hylätyn pyörän ojasta tai kadulta.

Molemmille, sekä varkaus- että löytymistapahtumille, lisätään tieto tapahtuman kohteesta ja tapahtumapaikasta ja –ajasta. Tapahtuman kohteena on luonnollisesti polkupyörä, ja sille voidaan lisätä erilaisia tunnistetietoja, kuten pyörän merkki ja malli, väri, ja sarjanumero. Tapahtumapaikaksi käyttäjä antaa sijaintitiedon kartalta, ja tapahtuma-ajaksi valitaan päivänmäärä ja kellonaika.

Näiden lisäystoimintojen lisäksi, palvelussa käyttäjä voi selata ja hakea molempia tapahtumia erilaisilla hakukriteereillä, jotka perustuvat tapahtumapaikkaan ja kohdetietoihin. Käyttäjä voi lähettää viestejä tapahtuman luojalle, jos haluaa esimerkiksi ottaa yhteyttä henkilöön, joka on löytänyt hänen pyöränsä.

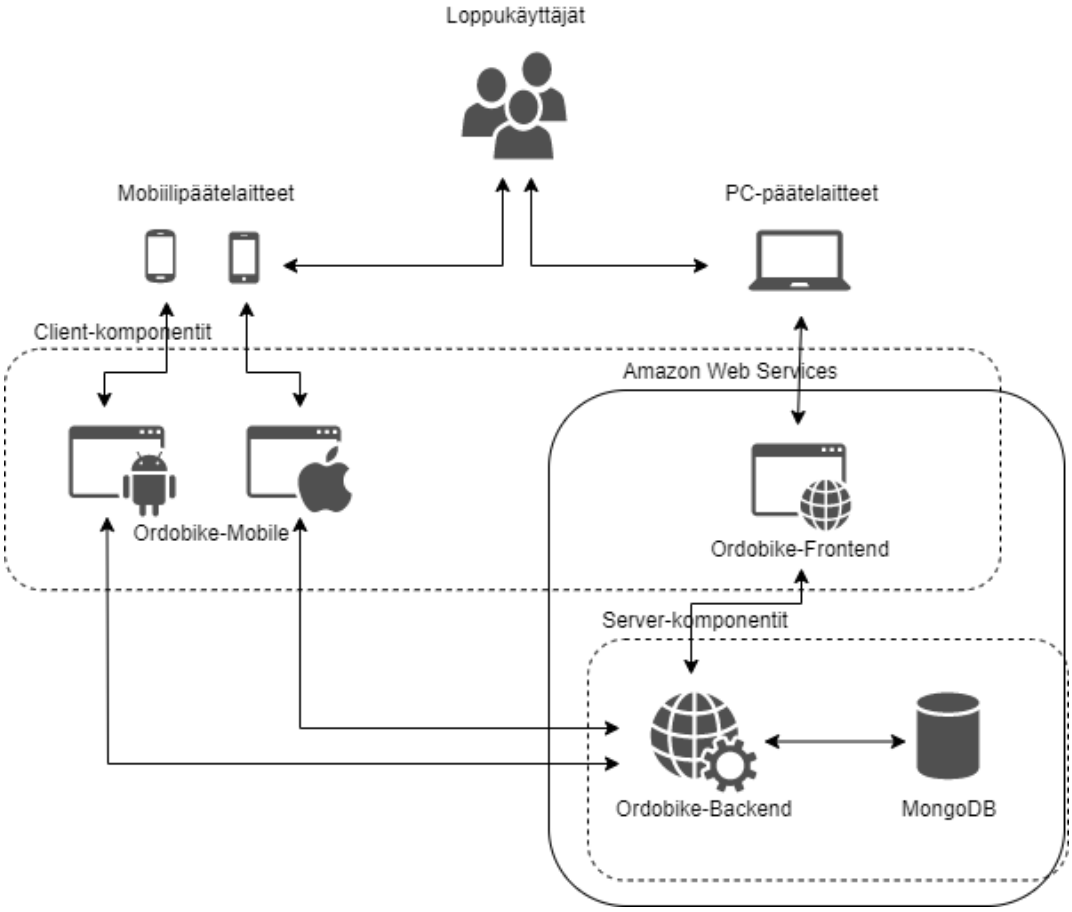
Palveluun tarvitsee rekisteröityä, jos siihen haluaa lisätä varkaus- tai löytötapahtumia, tai lähettää ja vastaanottaa viestejä. Rekisteröitymiseen tarvitsee ainoastaan sähköpostiosoitteen. Varastettuja ja löytyneitä pyöriä voi selata ilman rekisteröitymistä. Palvelu on kokonaisuudessaan ilmainen käyttää.

3.2 Palvelun rakenne ja teknologiat

Ordobike-järjestelmän komponentit jakautuvat karkeasti kahteen eri kategoriaan: palvelin-komponentteihin, sekä client-komponentteihin, joiden kautta loppukäyttäjät käyttävät palvelua. Palvelinohjelmistona toimii Ordibike-Backend, sekä client-ohjelmistoina ovat Ordobike-Frontend, joka tarjoaa palveluun internet-se-

laimilla käytettävän web-käyttöliittymän, sekä Ordobike-Mobile, jonka alaisuuteen kuuluvat Android- sekä iOS –mobiilisovellukset. Kaikkien näiden komponenttien kesken, Ordobike-palvelu rakentuu noin 21 000 koodirivistä.

Palvelun järjestelmäkaavio, joka visualisoi palvelun ohjelmistokomponentit sekä niiden suhteet, on esitelty alla (KAAVIO 1). Komponenttien toiminta, sekä käytetyt tekniikat on kuvattu lyhyesti seuraavissa niille varatuissa alikappaleissa.



KAAVIO 1. Ordobike-palvelun järjestelmäkaavio.

3.2.1 Ordobike-Backend ja MongoDB-tietokanta

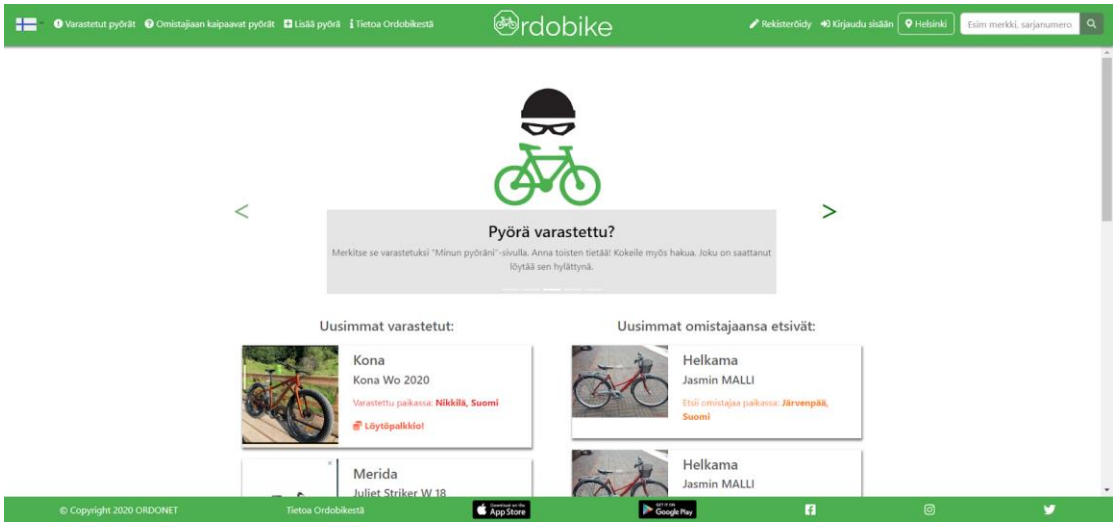
Ordobike-Backend käsittelee palvelun dataa, ja tarjoaa rajapinnan client-komponenteille, jonka kautta sitä tarkastellaan ja muokataan. Sovellus on node.js –palvelinohjelmistopohjainen, ja se on kehitetty Typescript-ohjelmointikielellä, joka käännetään ajoa varten Javascript-muotoiseksi. Sovellus käyttää Express.js –

frameworkia tarjotakseen REST-rajapinnan, johon client-sovellukset ovat yhteydessä http-protokollan yli lähetettävillä kutsuilla.

Palvelun data säilytetään MongoDB-tietokannassa, joka on valmis, avoimen lähdekoodin lisensoitu, NoSQL-tyypin tietokantaohjelma. Ordobike-Backendillä on rajapintayhteys tähän tietokantaan. Sekä Ordobike-Backend, että MongoDB-tietokantainstanssi suoritetaan Amazon Web Services –pilvipalelun sisällä.

3.2.2 Ordobike-Frontend

Ordobike-Frontend (KUVA 1) on palvelun client-sovellus, joka tarjoaa sille internet-selaimella käytettävän web-käyttöliittymän. Sovellus on toteutettu React-frameworkilla, joka on Meta-yrityksen (ent. Facebook) luoma avoimen lähdekoodin Javascript-kirjasto web-käyttöliittymien rakentamiseen. Frameworkillä kehityksessä käytetään Javascript-, HTML5-, ja SCSS- ohjelmointi- ja tyylittelykieliä.



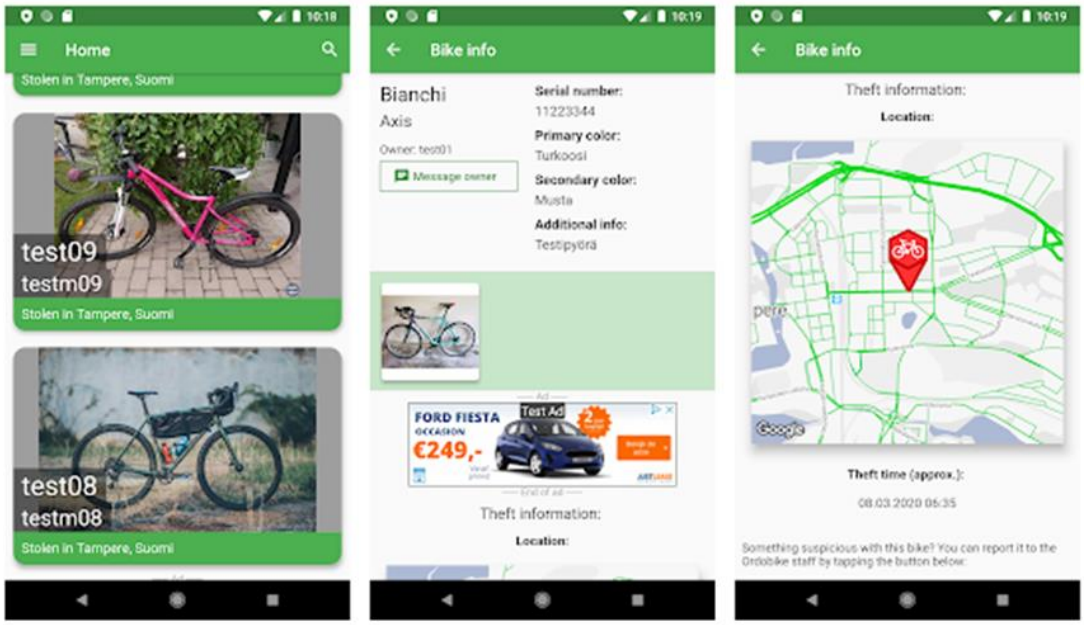
KUVA 1. Kuvankaappaus Ordobike-palvelun web-käyttöliittymästä.

Sovelluksen keskeisempiä toimintonäkymiä ovat käyttäjän pyörien lisäys- ja muokkausnäkymä, näkymä, jossa pyörä merkitään varastetuksi, sekä mm. kartan sisältävä, pyörävarkauden ja varastetun pyörän infosivu.

Ordobike-Frontend on hostattu Amazon Web Services –pilvipalvelussa, ja käyttäjät pääsevät siihen käsiksi yleisimmillä internet-selaimilla tämän raportin julkaisuhetkellä osoitteesta http://ordobike.fi.

3.2.3 Ordobike-Mobile

Ordobike-Mobile käsittää Ordobike-palvelun Android – ja iOS –mobiilisovellukset (KUVA 2). Sovellukset ovat ulkonäöltään ja toiminnallisuudeltaan samanlaiset, ja ne on käännetty samasta pääohjelmistoprojektista.



KUVA 2. Kuvankaappauksia Ordobike-mobiilisovelluksen eri näkymistä.

Ohjelmisto on kehitetty Googlen luomalla Flutter-ohjelmistokirjastolla, jolla voidaan rakentaa mobiilisovelluksia yhdellä koodipohjalla, ja sitten kääntää ne eri alustoille, kuten esimerkiksi Android-, iOS ja web-sovelluksiksi.

Kuten Ordobike-Frontend, myös mobiilisovelluksen keskeisimpiä näkymiä ovat pyörien tietojen lisäys- ja muokkausnäkymä, näkymät pyörävarkauksien ja niiden tietojen lisäämiseksi, sekä selausnäkymät varastettujen ja löytyneiden pyörien selaukseen.

Ordobike-Mobile –sovellukset ovat tämän raportin julkaisuhetkellä ladattavissa käyttäjien mobiililaitteille Google Play Storesta, sekä Apple Appstoresta, Ordobike-nimellä.

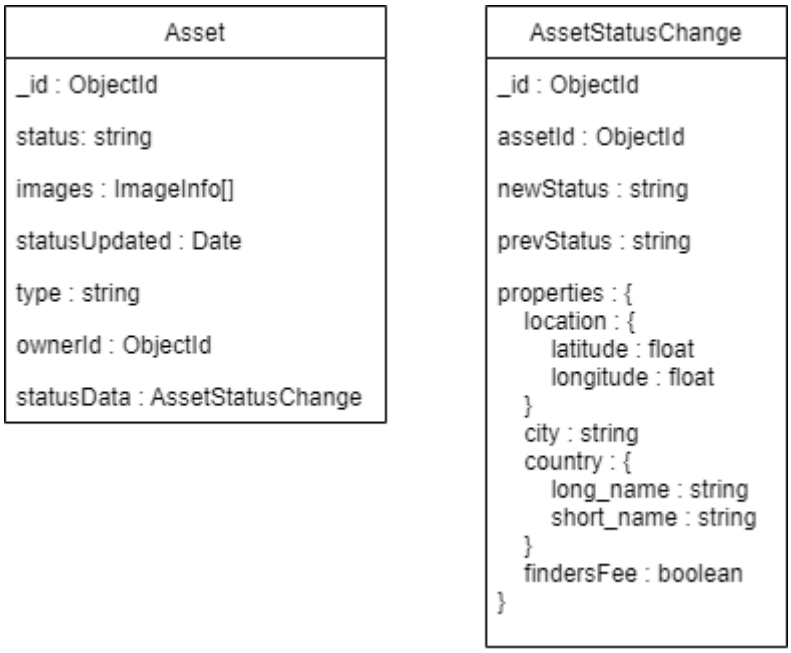
3.3 Palvelun data

Tämän työn viitekehyksessä relevantti data on Ordobiken varastettujen pyörien varkausmerkintätapahtumiin liittyvä data. Emme esimerkiksi tarvitse käyttäjätietoja, tai muuta palvelun dataa, vaan ainoastaan varkauksien sijainnit, ja mahdollisesti niihin liittyvät oheistiedot, kuten aikaleima ja varkauden kohdepyörä, mutta nämäkään eivät ole välttämättömiä perustoteutuksessa, jossa tarvitsemme ainoastaan tapahtuman sijaintitiedon.

Ordobiken varkaustapahtumilla tarvitsee aina olla kohde, ja koska palvelu käsittää polkupyörät, ovat kohteet käytännössä erilaisia polkupyöriä. Ennen kuin käyttäjä voi lisätä palveluun varkaustapahtuman, tarvitsee tällä olla lisättynä käyttäjätililleen polkupyörä, joka voidaan liittää varkaustapahtuman kohteeksi. Ohjelmistoarkkitehtuuritasolla ei puhuta kuitenkaan polkupyöristä, vaan on päädytty käyttämään abstraktia käsitettä *Asset* (KAAVIO 2), joka tarkoittaa vapaasti suomennettuna omaisuuserää tai resurssia.

Myöskään pyörävarkauksia ei ohjelmistoarkkitehtuuritasolla kutsuta sellaisiksi, vaan niitäkin puhutellaan abstraktilla käsitteellä *Asset Status Change*, eli resurssin tilan muuttuminen. Käytännössä ohjelmistotasolla varkausmerkinnän tapahtuessa luodaan tietokantaan resurssin tilamuutos, jossa uusi tila on *ASSET_STATUS_STOLEN*, eli varastettu. Muita tiloja voivat olla myös *ASSET_STATUS_FOUND* (löytynyt), jolloin on löytynyt hylätty pyörä, ja *ASSET_STATUS_OK*, jolloin pyörä on normaalisti omistajallaan, mutta sen tiedot ovat tallennettuna hänen käyttäjätililleen. Resurssin tila on tietokantatasolla myös itse resurssilla. Resurssin tilamuutokseen merkitään tietokannassa aina kohderesurssin tunniste. Tietokanta- ja kooditasolla näitä dokumentteja puhutellaan termillä *AssetStatusChange* (KAAVIO 2 ja KUVA 3).

Resurssin tilamuutoksen tietokantamerkintään liitetään myös tapahtuman sijaintitieto, sekä tapahtuman aikaleima. Sijaintitiedosta yritetään myös parsia Google Places –rajapinnan kautta tietokantamerkintään kaupunki- ja maatieto, palvelun hakutoiminnallisuutta varten.



KAAVIO 2. *Asset* (pyörä) ja *AssetStatusChange* (pyörän tilan muutostieto) -tietokantaluokkien luokkakaaviot.

```
{
    "_id" : ObjectId("5e62c664044lea2d34e4f7a7"),
    "status" : "ASSET_STATUS_STOLEN",
    "images" : [],
    "statusUpdated" : ISODate("2020-03-06T21:54:32.815Z"),
    "type" : "ASSET_TYPE_BIKE",
    "properties" : {
        "PARAM_COMMON_MANUFACTURER" : "testipyörä",
        "PARAM_COMMON_MODEL" : "testimalli",
        "PARAM_COMMON_SERIAL_NUMBER" : "123444",
        "PARAM_COMMON_PRIMARY_COLOR" : "musta",
        "PARAM_COMMON_SECONDARY_COLOR" : "valkoinen",
        "PARAM_COMMON_ADDITIONAL_INFO" : "testipyörä vaan"
    },
    "ownerId" : ObjectId("5e5d527424ce934f040d6be6"),
    "__v" : 0,
    "statusData" : {
        "_id" : ObjectId("5e62c698044lea2d34e4f7a9"),
        "assetId" : ObjectId("5e62c664044lea2d34e4f7a7"),
        "newStatus" : "ASSET_STATUS_STOLEN",
        "prevStatus" : "ASSET_STATUS_NORMAL",
        "created" : ISODate("2020-03-06T21:54:00.000Z"),
        "updated" : ISODate("2020-03-06T21:54:00.000Z"),
        "properties" : {
            "location" : {
                "type" : "Point",
                "coordinates" : [
                    23.7969996035099,
                    61.4901091035145
                ]
            },
            "city" : "Tampere",
            "country" : {
                "long_name" : "Finland",
                "short_name" : "FI"
            },
            "findersFee" : true,
            "price" : null
        },
        "__v" : 0
    }
}
```


KUVA 3. Kuvankaappaus *AssetStatusChange*-tietokantadokumentin JSON-rakenteesta.

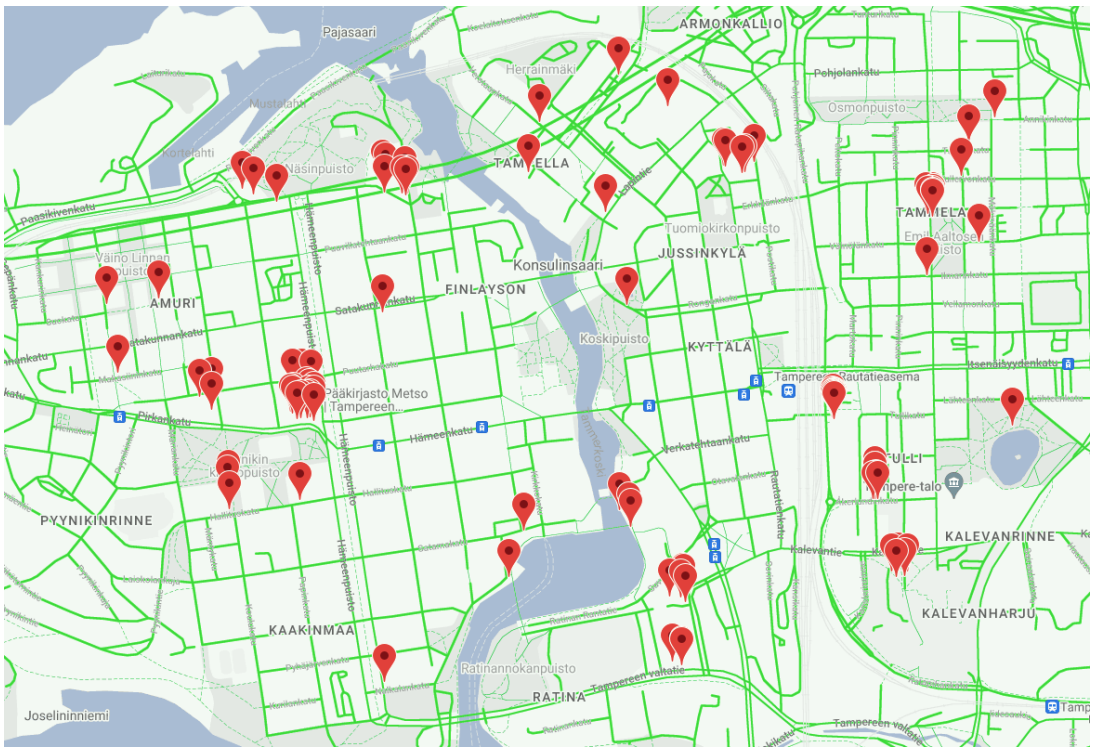
Nämä tilamuutokset ovat se data, jota tämän opinnäytetyön yhteydessä tarvitaan. Tarkalleen ottaen tilamuutoksien koordinaattimuodossa olevat sijaintitiedot, jotka keräämme ja syötämme koneoppimisalgoritmille koulutusdataksi.

3.4 Esimerkkidatan luonti

Työn aloitusvaiheessa, palvelun datan tarkastelun jälkeen, kävi ilmi, että varsinaista käyttäjien luomaa oikeaa dataa ei välttämättä olisi tarpeeksi, jotta sitä voitaisiin käyttää koneoppimisalgoritmien opetusdatana. Varkaustapahtumia oli ripoteltuna liian harvakseltaan koko Suomen alueelle, jotta saataisiin riittävän paikallisen tason varkauskeskittymätietoa.

Tämän ongelman ohittamiseksi luotiin työn puitteissa testidataa, eli tekaistuja pyörävarkaustapahtumia eri sijainteihin. Testidataa ei luotu julkisessa käytössä olevaan palveluun, vaan erillisenä offline-datana kehitysympäristössä.

Testidatan pyörävarkauksille asetettiin sijainniksi Tampereen kaupungin keskusta-alueen eri sijainteja (KUVA 4). Tapahtumia pyrittiin osittain ryhmittelemään useampi samoihin sijainteihin, jotta koneoppimisen tuloksien oikeankaltaisuutta voitaisiin tarkastella selkeämmin. Odotettavissa on, että tuloksissa ilmenisi jonkinlaisia klustereita sijaintirykelmien kohdalla.



KUVA 4. Esimerkkidatan pyörävarkaustapaukset punaisilla karttamerkeillä merkattuina.

3.5 Tavoitteet: uudet tekoälypohjaiset ominaisuudet

Opinnäytetyön viitekehyksessä päätettiin ottaa tavoitteeksi toteuttaa palveluun kaksi uutta ominaisuutta, jotka pohjautuvat tekoälyn hyötykäyttöön. Nämä ominaisuudet katsottiin tarpeeksi kompakteiksi, jotta työmäärä pysyisi sopivalla tasolla, mutta kuitenkin selkeän näkyviksi, joista palvelun käyttäjä voisi saada varteenotettavaa lisähyötyä. Tavoiteominaisuudet on selitetty lyhyesti kahdessa seuraavassa alikappaleessa, ja lopulliset toteutukset esitellään laajemmin kappaleessa 5.

Työn määrän rajaamiseksi uudet ominaisuudet päätettiin toteuttaa proof of concept (PoC) -periaatteella, eli lopulliseen palvelun tuotantoversioon ei niitä tämän työn puitteissa oteta käyttöön, ja mahdollinen tuotteistus tehdään opinnäytetyön ulkopuolella.

3.5.1 Käyttäjän sijaintiin perustuva varkausindikaattori

Ominaisuudella pyritään antamaan käyttäjälle pikatieto hänen sijaintinsa pyörävarkauksien tasosta. Sijainti otetaan käyttäjän mobiililaitteesta, ja se on luultavasti järkevää toteuttaa vain mobiilisovellukseen, koska web-sovellusta käyttävät pääasiassa PC-laitteet, jotka eivät ole älypuhelimien tavoin liikkuvia päätelaitteita.

Sovelluksen käyttöliittymään lisätään visuaalinen indikaattori senhetkisen sijainnin pyörävarkauksien tasosta. Indikaattori voi olla esimerkiksi pylväs-ikoni, jossa pylvään koko kasvaa sitä mukaan, mitä enemmän pyörävarkauksia alueella on, tai jonkinlainen varoitusikoni, jos pyörävarkaudet ovat alueella yleisiä. Indikaattorin yhteydessä voidaan antaa myös lyhyt selite, esimerkiksi ”Lähialueella on tapahtunut tavallista enemmän pyörävarkauksia viimeisen 7 päivän aikana”.

3.5.2 Pyörävarkauskeskittymien karttanäkymä

Tämä ominaisuus tulee pitämään sisällään karttanäkymän, johon indikoidaan lähialueen, esimerkiksi kaupungin, alueet, joihin pyörävarkaudet keskittyvät. Ominaisuus voidaan toteuttaa sekä mobiilisovellukseen, että web-sovellukseen.

4 VALITUT SOVELLUSTAVAT

4.1 Tekoälytoiminnallisuus: ydinestimointi ja haversine-etäisyysmittari

Tämän työn tekoälyosuuden keskeisin vaihe oli koneoppimisalgoritmin valinta, jolla pyörävarkausdatan koneoppimismalli opetetaan. Sijaintidatan tekoälykäytön tutkimuksessa löytyi kappaleessa 2.2 mainittu koneoppimiseen erikoistuneen Scikit Learn -koodikirjaston esimerkki, jossa kahden eläinlajin koordinaattimuotoista sijaintidataa oltiin käytetty määrittämään missä kummankin lajin vahvimmat esiintymät olivat, käyttämällä hyväksi ydinestimointia (engl. kernel density estimation).

Scikit Learn –kirjasto toteuttaa ydinestimoinnin *KernelDensity*-koodiluokalla. Sana ”kernel” on englantia ja tarkoittaa ydintä. Käsite ”ydin” voidaan tässä yhteydessä ymmärtää jonkin ilmiön tiheimpien esiintymiskohtien keskipisteenä. Myöhemmin tässä pääkappaleessa mainitussa tiheyspintakartassa esimerkiksi ytimet sijaitsevat vahvimmin värjättyjen karttakohtien keskellä. *KernelDensity* ottaa parametrikseen Nearest Neighbour –kategoriaan kuuluvan algoritmin, joksi tässä työssä valikoitui Ball Tree –puualgoritmi, jota se käyttää ytimien sijainnin ja tiheyden laskentaan. Parametrina annetaan myös metriikka, jolla Ball Tree laskee sille annettujen datapisteiden etäisyyden toisistaan. Metriikaksi valikoitui Scikit Learnin esimerkistä haversine-etäisyysmittari. Mainitsemisen arvoista tässä yhteydessä on, että toisena algoritmivaihtoehtona *KernelDensitylle* olisi ollut myös KDTree-algoritmi, mutta haluttua haversine-mittaria ei voinut käyttää sen kanssa.

Nearest Neighbour –algoritmeja voidaan käyttää joko luokitteluun, tai regressioon. Luokittelussa data jaetaan erinäisiin luokkiin, joissa datan ominaisuudet ovat samankaltaisia, tai lähellä toisiaan. Kun algoritmi on opettanut mallin, siltä voidaan kysyä, mihin luokkaan annettu data todennäköisimmin kuuluu. Regressiossa algoritmi opettaa mallin ilman luokittelua, ja kysyttäessä kyselydatalla malli kertoo, kuinka paljon samankaltainen annettu data on verrattuna opetettuun dataan. Pyörävarkausdatan tapauksessa kyse on regressiosta, sillä eri luokkia ei ole, ainoastaan sijainteja, joissa on tapahtunut varkaus. Malli siis opetetaan pyörävarkausdatalla, jonka jälkeen se on määrittänyt tyypilliset pyörävarkaussijainnit,

ja opetuksen jälkeen siltä kysytään kyselydatan samankaltaisuutta. Toisin sanottuna, onko annettu latitude/longitude -sijainti lähellä tyypillistä varkaussijaintia.

Nearest Neighbour –algoritmit sellaisenaan ovat myös ns. “lazy learner” –algoritmeja, joka tarkoittaa, että ne opettelevat datan vasta kun niiltä kysytään jotain kyselydatalla. *KernelDensity*-estimaattorin kanssa käytettynä voidaan tosin tallentaa algoritmilla opetettu estimaatiomalli myöhemmin uudelleenladattavaksi käyttäen Sklearnin joblib-koodikirjastoa (KUVA 5).

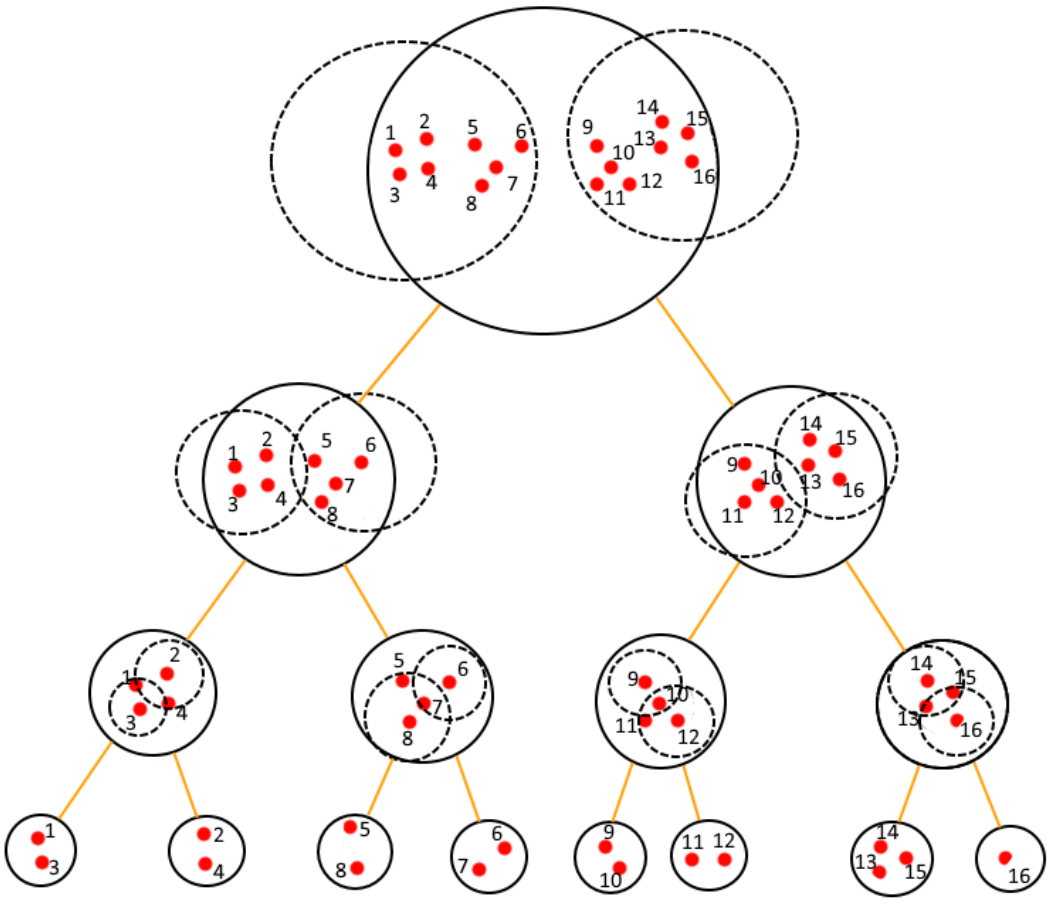
```
from sklearn.externals import joblib
joblib.dump(kdeModel, './kdeModel.pkl')
kde = joblib.load('./kdeModel.pkl')
kde.score_samples(([123.4, 567.8])
```

KUVA 5. Esimerkkikoodi opetetun *KernelDensity* -mallin tallentamisesta ja lataamisesta.

Ydinestimointi sekä käytetty Ball Tree –Nearest Neighbor –algoritmi ovat osa ohjaamattoman oppimisen koneoppimiskategoriaa. Tämä tarkoittaa, että ne saavat ei-diskreettiä dataa opittavaksi, jota ei ole kategorisoitu tai ”selitetty” ihmisen toimesta. Tämän oppimistyylin vastakohta on ohjattu oppiminen, jossa opetusdata on selitetty, esimerkiksi eläinlajin tunnistuksessa opetusdatassa on kerrottu valmiiksi mihin eläinlajiin annetut opetuskuvat kuuluvat. Nearest Neighbors –algoritmikategorian alle kuuluu sekä ohjaamatonta, että ohjattua koneoppimista, mutta ydinestimoinnin yhteydessä käytössä ovat vain ohjaamattoman oppimisen algoritmeja (Vanderplas, n. d). Pyörävarkaussijaintidatan suhteen ohjaamaton oppiminen on luontevampi vaihtoehto, koska dataa ei ole millään tavoin ennalta luokiteltu, ja datapisteet kaikki kuvastavat sijainteja, joissa varkaus on tapahtunut, ja haluamme vain tietää, kuinka samankaltaista kyselydata on opetusdataa.

Ball Tree –algoritmi on nimensä mukaisesti puualgoritmi, eli algoritmi järjestää sille annetun datan hierarkkisesti puumaiseen rakenteeseen, entistä tarkempiin klustereihin, joissa ylätason noodin, eli oksan, alle muodostetaan lapsinoodeja. Lapsinoodit sisältävät ylätason noodin datan tarkemmin luokiteltuna eri luokkiin, tai klustereihin (Marius, 2020). Ball Tree on myös binäärialgoritmi, eli jokaisella noodilla voi olla vain kaksi lapsinoodia. Nimen ”ball” tulee siitä, että algoritmi pys-

tyy jakamaan datansa joko ympyrän muotoisten klustereiden sisälle kaksiulotteisessa avaruudessa, tai kolmiulotteisessa avaruudessa pallomaisten klusterien sisälle. Tämän vuoksi algoritmi on hyvä myös koordinaattidatan käsittelyyn, koska ne ovat kaksiulotteisia arvoja. Algoritmin perustoimintaperiaate on määrittää annetusta datasta kaksi arvoiltaan kauimpana toisistaan olevaa datapistettä, ja muodostaa niitä keskipisteinä käyttäen kaksi uutta ympyrä- tai palloaluetta, jonka sisälle klusterin datapisteet jaetaan. Näistä kahdesta uudesta ympyrästä muodostuvat kaksi uutta datapisteklusteria, joista tulee puun lapsinoodeja. Samaa aliklusterien muodostustapaa noudatetaan myös näille uusille klustereille niin kauan, kunnes dataa ei ole enää luokiteltavana alemmalle tasolle, tai kunnes jonkinlainen rajaparametriarvo on saavutettu. Algoritmin toimintaa on kuvattu visuaalisesti seuraavassa kaaviossa (KAAVIO 3).

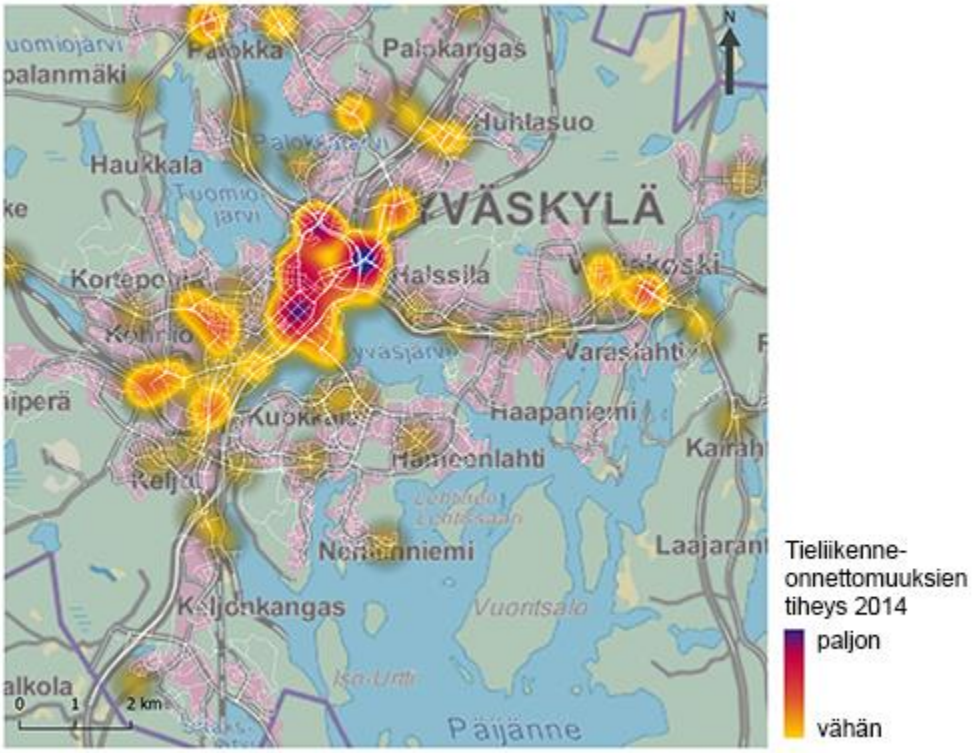


KAAVIO 3. Ball Tree -puualgoritmin toiminta visualisoituna. Punaiset pisteet ovat datapisteitä tunnistenumeroineen, ja katkopisteellä olevat ympyrät kahden kauimpana toisistaan olevan datapisteen ympärille luotuja uusia klusterialueita, joiden sisälle datapisteet jaotellaan.

Haversine-mittaria käytettäessä algoritmi laskee lyhimmän välimatkan kahden pallomaisella pinnalla sijaitsevan koordinaattipisteen päällä, eli se on luonteva ja yleisessä käytössä maapallolla sijaitsevien koordinaattipisteiden väliseen laskentaan (Geeks for Geeks, 2022). Pyörävarkausdatassa haversine-metriikalla siis lasketaan varkaussijaintien etäisyyksiä toisiinsa, ja tällä tavalla löydetään klustereita, joissa varkaudet ovat lähellä toisiaan.

4.2 Tiheyspintakartta

Tiheyspintakartta on tilastoteemakarttatyyppi, joka visualisoi jonkin ilmiön esiintymistiheyttä jollain määritellyllä alueella. Tiheyttä ilmaistaan erilaisilla väriskaaloilla, joissa ilmiön tiheimmät esiintymisalueet värjätään voimakkaimmalla värillä. Väriskaala voi olla esimerkiksi vihreästä punaiseen, jossa punaisella värjätään tiheimmät esiintymäalueet. Tiheyspintakartta on yleinen tapa esittää rikoksia. Esimerkiksi Tilastokeskuksen Tilastokoulun esimerkki (n. d.) tiheyspintakartasta on kartta Jyväskylän alueen liikenneonnettomuuksien yleisimmistä alueista (KUVA 6). Sen katsottiin olevan myös Ordobiken tapauksessa selkeästi paras tapa indikoida pyörävarkauksia ja niiden tiheyttä eri alueilla. Tiheyspintakartasta käytetään informaatioteknologiassa yleisemmin sen englanninkielistä nimitystä heatmap.

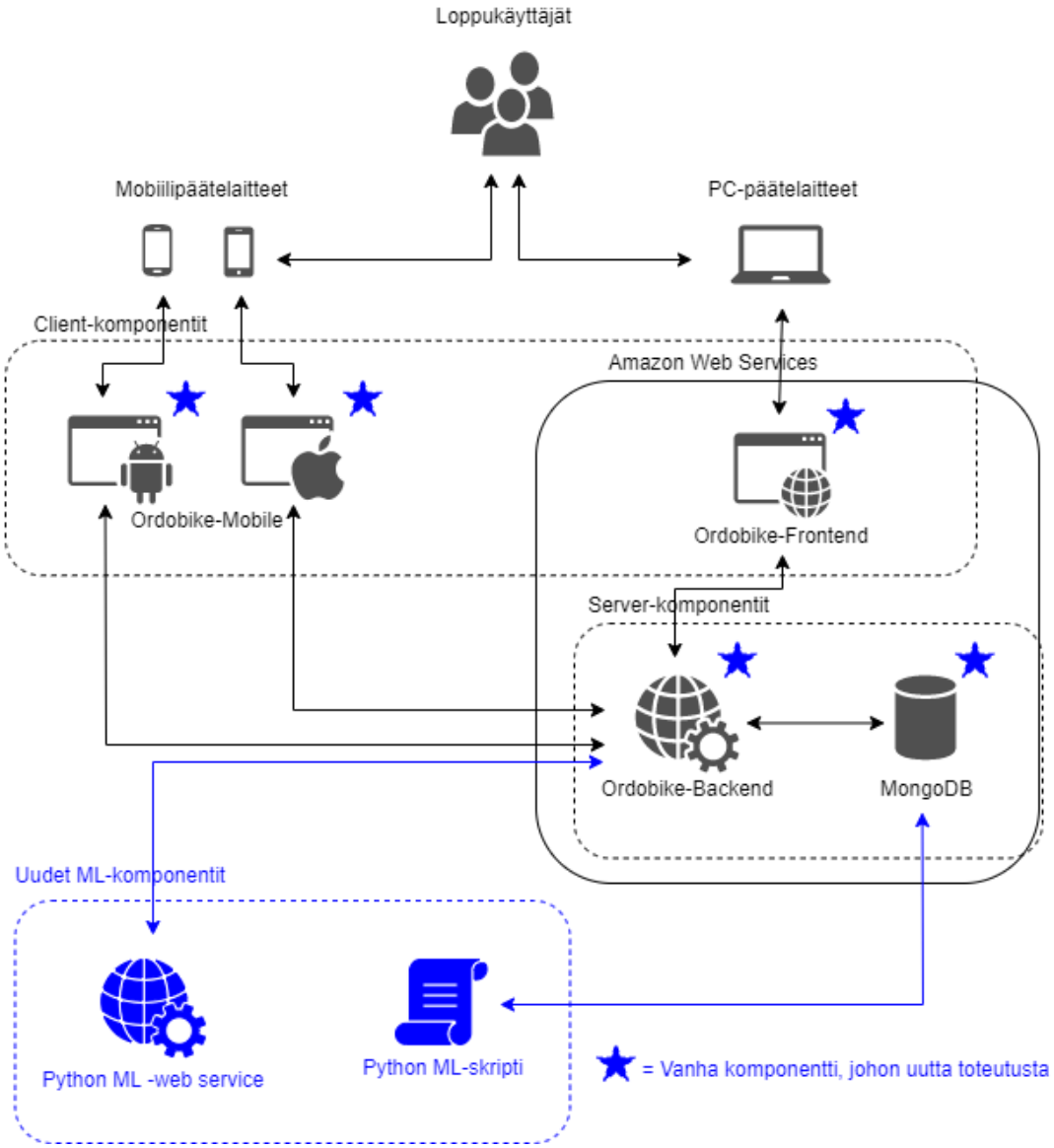


KUVA 6. Tilastokeskuksen Tilastokoulun (n. d.) esimerkkikuva tiheyspintakartasta, eli heatmapista, jossa esitetään Jyväskylän liikenneonnettomuuksien yleisyyttä eri alueilla.

Tässä työssä ei ole tarkoitus syöttää pyörävarkaustapahtumien sijainteja suoraan tiheyskarttaan, vaan siihen syötetään ydinestimointi-koneoppimisalgoritmista ulos saatu koordinaattimatriisi. Nämä vaihe käydään läpi tarkemmin toteutuskappaleissa 5.1 ja 5.4.

4.3 Komponentit ja niiden suhteet

Palveluun kehitetään työn puitteissa sekä kokonaan uusia komponentteja, että uusia ominaisuuksia olemassa oleviin komponentteihin. Uudet komponentit ovat koneoppimiseen liittyviä Python-ohjelmakomponentteja, ja uusia ominaisuuksia tulee kaikkiin ennestään olemassa oleviin palvelun komponentteihin. Komponenttien uudet suhteet on esitelty alla olevassa järjestelmäkaaviossa (KAAVIO 4), ja niiden toiminnallisuudet on selostettu seuraavissa alikappaleissa.



KAAVIO 4. Ordobike-palvelun järjestelmäkaavio PoC-komponenteilla ja uutta toiminnallisuutta sisältävillä komponenteilla päivitettynä. Uudet toiminnot värjätty sinisellä.

4.3.1 Uusi komponentti: Python ML-skripti

Skriptin tehtävänä on ensin noutaa Ordobike-tietokannasta varkaustapahtumat, kerätä niistä koordinaattitiedot, ja sen jälkeen opettaa koneoppimismalli koordinaattidatalla käyttäen valittua *KernelDensity*-luokkaa, Ball Tree –algoritmilla.

Kun malli on opetettu, skriptillä on vielä kaksi tehtävää: ajaa saadulle mallille ennustus massaoperaationa koordinaattimatriisin arvoille, ja tallentaa saadut, 0-1

arvoväliin normalisoidut, score-arvot Ordobiken MongoDB-tietokantaan, sekä tallentaa opetettu malli seuraavaksi esitellyn Python -web servicen ladattavaksi ja käytettäväksi.

4.3.2 Uusi komponentti: Pyhton ML -web service

Komponentin tehtävänä on ladata Python-koneoppimisskriptin luoma koneoppimismalli, ja tarjota REST-rajapinta, jonka kautta voidaan kutsua varkaustasopisteytyksen antavaa *predict()*-metodia ladatulle mallille. Rajapintakutsuun annetaan latitude/longitude -koordinaatit, joilla kyseistä metodia kutsutaan.

Rajapinta palauttaa arvoväliin 0-1 normalisoidun score-arvon. Rajapintaa tullaan kutsumaan Ordobike-Backendin kautta, jotta Ordobike-Frontend- ja Ordobike-Mobile –client-ohjelmien ei tarvitse muodostaa erillisiä yhteyksiä kahteen eri rajapintaan.

4.3.3 Uudet ominaisuudet: Ordobike-Frontend

Ordobike-Frontend -web-sovellukseen on määrä toteuttaa erillinen varkauskarttasivu. Tämä sivu pitää sisällään kartan, jonka päälle luodaan tiheyskartta paikallisalueen pyörävarkausesiintymistä. Sivu yrittää päätellä missä käyttäjä sijaitsee, ja ladata kartan automaattisesti kyseiseen sijaintiin.

PoC-kehitystavan johdosta, kehitystyön vähentämiseksi, varkauskarttasivulle pääsevät kaikki palvelun käyttäjät, eikä se vaadi käyttäjätiliä palveluun.

4.3.4 Uudet ominaisuudet: Ordobike-Mobile

Mobiilisovellukseen toteutetaan samanlainen varkauskarttanäkymä, kuin web-sovellukseenkin, ja se lataa kartalle sijainniksi käyttäjän mobiililaitteen GPS-sijainnin.

Varkauskartan lisäksi mobiilisovellukseen toteutetaan varkaustasoindikaattori, joka tulee olemaan jonkinlainen palkki-ikoni, jossa palkkien määrä kasvaa, mitä enemmän varkauksia esiintyy kyseisessä sijainnissa. Toiminnallisuus käyttää käyttäjän mobiililaitteen sijaintia, ja se päivittää sijainnin ja kysyy sen varkausesiintymätason Ordobike-Backend-sovellukselta esimerkiksi minuutin välein. Backend taas kysyy arvon uudelta Python -web serviceltä. Indikaattori tulee näkymään sovelluksen oikeassa yläkulmassa sovelluspalkissa, josta se on aina nähtävillä sovellusta käytettäessä. Ikonia klikkaamalla voidaan myös päästä varkauskarttanäkymään.

4.3.5 Uudet ominaisuudet: Ordobike-Backend ja tietokanta

Ordobike-Backendin REST-rajapintaan tarvitsee lisätä uusia polkuja, jolla clientit voivat hakea sekä varkauspisteytykset tietyn alueen sisältä (varkausmääräkarttaa varten), että yksittäisen sijainnin varkauspisteytyksen, mobiilisovelluksen varkausindikaattoria varten.

Tietokantaan tarvitsee luoda uusi kokoelma varkauspisteytyksille, jotta ne voidaan tallentaa sinne Python -ML-skriptistä.

5 KÄYTÄNNÖN TOTEUTUKSET

5.1 Python ML-skripti

Python-ympäristönä käytettiin Anaconda 2 -ympäristöä, joka mahdollistaa usean eri Python-ympäristön asentamisen ja ajamisen, ja ohjelmien tekemisen näille ympäristöille. Python-ohjelmoinnissa moni ohjelmakirjasto on riippuvainen tietystä Python-versiosta ja mahdollisesti toisen kirjaston tietystä versiosta, jota varten Anaconda-ympäristö on hyödyllinen.

Mainitsemisen arvoiset kirjastot tämän työn yhteydessä olivat itse ohjelmointikieli Pythonin versio 3.7, scikit-learn -tekoäly- ja koneoppimiskirjaston versio 1.0.1, pandas-kirjaston versio 1.3.4 datan käsittelyyn, sekä pymongo- kirjaston versio 4.1.1 MongoDB-tietokantayhteyden muodostamiseen ja tietokantaoperaatioihin.

Skriptiä ajetaan on-demand -ajotavalla, ja sen toiminta on seuraava:

1. Luodaan tietokantayhteys Ordobiken MongoDB-tietokantaan.
2. Luetaan tietokannasta *AssetStatusChange*-tapahtumat, joissa tapahtumatyyppi on varkaus.
3. Eritellään tapahtumista koordinaattidata.
4. Luodaan Ball Tree –algoritmia ja haversine-etäisyysmetriikkaa käyttävä *KernelDensity* -ydinestimaattori ja opetetaan se koordinaattidatalla.
5. Tallennetaan estimaattorista saatu malli.
6. Ajetaan massaennuste saadulle mallille Tampereen ydinkaupunkialueen koordinaattimatriisin sijaintiarvoilla.
7. Normalisoidaan massaennusteesta saadut score-arvot arvovälille 0-1 ja yhdistetään niihin sijaintiarvot.
8. Suodatetaan score-arvoista pois arvot, jotka ovat 0.0001 tai pienempiä.
9. Tallennetaan score-arvot Ordobiken MongoDB-tietokantaan scores-kokoelmaan. Poistetaan mahdolliset vanhat arvot ennen tallennusta.

Pyörävarkauksien koordinaattidatan erittely Ordobiken pyörävarkausdatasta tapahtui suoraviivaisesti lataamalla kaikki *AssetStatusChange*-dokumentit, joiden *newStatus*-parametrin arvo oli ”ASSET_STATUS_STOLEN” niiden MongoDB-

kokoelmasta. Saaduista dokumenteista eriteltiin *properties.location.coordinates* –tietueiden koordinaatit omaan listamuuttujaan (KUVA 7).

```
import numpy as np
import pymongo

mongoClient = pymongo.MongoClient("mongodb://localhost:27017/")
db = myclient["ordoDB"]
statusChangesCollection = db["assetstatuschanges"]

# Tietokantadokumenteista tarvitaan ainoastaan properties-kenttä,
# joten ohjeistetaan kyselyä palauttamaan vain se jokaisesta tulosdokumentista
scDocs = statusChangesCollection.find({ "newStatus": "ASSET_STATUS_STOLEN" }, { "properties": 1 })

# Eritellään koordinaatit dokumenteista omaan listaan
coordsArray = []
for statusChange in scDocs:
    coordsArray.append({'lat' : statusChange['properties']['location']['coordinates'][0],
                        'lon' : statusChange['properties']['location']['coordinates'][1]})

# Muunnetaan koordinaatit KernelDensitylle sopivaan tietorakenteeseen,
# ja pyöristetään koordinaatit samalla 7 desimaaliin
theftLatLons = np.vstack([np.around(coordsArray['lat'], 7), np.around(coordsArray['lon'], 7)]).T
```

KUVA 7. Pyörävarkaussijaintidatan lataus tietokannasta *KernelDensity*-estimaattorin käyttöön.

KernelDensity-estimaattori ajettiin muuten oletusasetuksilla (KUVA 8), lukuun ottamatta *bandwidth*-asetusta, joka säätelee tiheysvaihtelun löytämisen tarkkuutta. Tätä asetusarvoa tarvitsi muuttaa huomattavasti pienemmäksi oletusarvosta 1.0, koska koordinaattien vaihtelutaso on pienten desimaalilukujen luokkaa. Esimerkiksi koordinaattiarvon muutos yhdellä tarkoittaa tosimaailmassa jo 111 kilometrin sijaintimuutosta (GIS Wiki, 2011). Oikeantasoinen arvo löydettiin lopulta vain kokeilemalla eri arvoja, ja parametrille asetettiin lopuksi arvo 0.00001.

```
from sklearn.neighbors import KernelDensity

# Luodaan KernelDensity-estimaattori
kde = KernelDensity(bandwidth=0.00001, algorithm='ball_tree', metric='haversine')
kde.fit(np.radians(theftLatLons))
```

KUVA 8. *KernelDensity*-estimaattorin luonti, sekä opettaminen varkaussijaintidatalla *fit()*-funktiota käyttäen.

Kun koneoppimismalli oli opetettu pyörävarkauksien sijaintidatalla, koska pyörävarkauskarttaa varten tarvittiin varkauspisteytykset valmiiksi laskettuina tietokantaan, sen sijaan että siltä kysyttäisiin yksittäisiä pisteityksiä yhdelle sijainnille kerrallaan, käytettiin massaoperaatiota *score_samples()*, jolta kysyttiin kerralla Tampereen ydinalueen koordinaattimatriisin pisteytykset (KUVA 9). Matriisissa on

koordinaatteja noin 20 metrin välein, eli koordinaattiarvojen vaihteluväli on 0.0004, ja sen koko on 5000 * 5000, joten se sisältää yhteensä 25 miljoonaa eri koordinaattisijaintia. Matriisin koordinaattipisteet ovat välillä 61 – 61,999999 leveysasteella ja 22,5 – 24,499999 pituusasteella, joka katsottiin sopivaksi Tampereen ydinkeskustan kattavaksi alueeksi.

```
# Luodaan suorakulmainen koordinaattimatriisi,
# joka alkaa vasemman yläkulman koordinaatista 22.5, 61.0,
# ja päättyy oikean alakulman koordinaattiin 24.499999, 61.999999
mgTestY = np.linspace(61.0, 61.999999, 5000) # välin pituus 1, koska latitude välillä -90 - 90
mgTestX = np.linspace(22.5, 24.499999, 5000) # välin pituus 2, koska longitude välillä -180 - 180
mgTestY = np.around(mgTestY, 7)
mgTestX = np.around(mgTestX, 7)
mgx, mgy = np.meshgrid(mgTestX, mgTestY)
mgxy = np.vstack([mgy.ravel(), mgx.ravel()]).T

# Kysellään tiheyspisteytys matriisin sijainneille
densityScores = kde.score_samples(np.radians(mgxy))
```

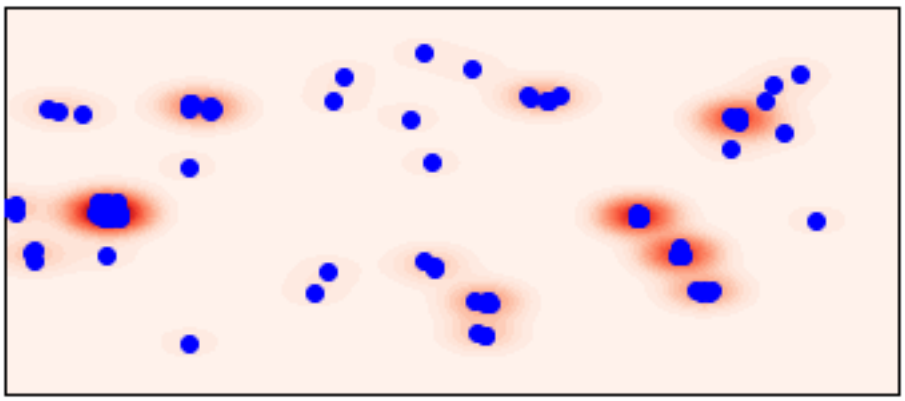
KUVA 9. Koordinaattimatriisin luonti ja tiheyspisteytysten kysely *score_samples()*-funktiolla.

Matriisiajon tuloksista suodatetaan mukaan tietokantaan tallennettavat pisteytykset, joiden pisteytysarvo on isompi kuin 0.0001. Suodatuksen jälkeen ajosta jäi jäljelle 7177 pisteytyssijaintia 25 miljoonasta, jotka tallennettiin tietokantaan (KUVA 10).

```
# Tiheyspisteytysten normalisointi välille 0-1
mgz = np.exp(densityScores)
mgz = mgz.reshape(mgx.shape)
stacked = np.vstack([mgy.ravel(), mgx.ravel(), mgz.ravel()])

def normalize(value, min, max):
    return (value - min) / (max - min)

normalized = []

stackMin = stacked[2].min()
stackMax = stacked[2].max()

for i in range(len(stacked[2])):
    normalized.append(np.round(normalize(stacked[2][i], stackMin, stackMax), 5))

# Yli 0.0001 pisteytysarvosijaintien tallennus tietokantaan
scoreLocations = []
for i in range(len(normalized)):
    if normalized[i] > 0.0001:
        scoreLocations.append([stacked[0][i], stacked[1][i], normalized[i]])

dataF = pd.DataFrame(scoreLocations, columns=['lat', 'lon', 'weight'])

# Poistetaan kokonaan vanhat pisteytykset
db['scores'].delete_many({})

docArray = []
for index, row in dataF.iterrows():
    docArray.append({'lat' : row['lat'], 'lon' : row['lon'], 'score' : row['weight']})

db['scores'].insert_many(docArray)
```

KUVA 10. Pisteytysarvojen normalisointi ja niiden tallennus tietokantaan sijaintitietoineen.

Kehityksen aikana skriptiin asetettiin väliaikainen kuvan tulostus, jossa indikoitiin käytettyjä sijaintipisteitä, ja koneoppimisalgoritmin massaennusteesta saatuja alueita, jotta voitiin todeta algoritmin toimivuus. Viimeisimmän skriptiversion antama kuva alla (KUVA 11).



KUVA 11. Pyhton ML-sktiptin piirtämä havainnekuva tuloksina saaduista tiheysalueista (punainen väritys), annetuilla koordinaattisijainneilla (siniset pisteet).

Skripti myös tallentaa opetetun *KernelDensity*-mallin tiedostoksi ladattavaksi Python ML –web-servicelle (KUVA 12).

```
from sklearn.externals import joblib

joblib.dump(kde, './theft_kde.pkl')
```

KUVA 12. Opetetun mallin tallennus tiedostoksi.

Lopullisen skriptin kooksi tuli noin 240 koodiriviä, sisältäen kommentit ja erilaiset ajon aikaiset debug-tulostukset.

5.2 Python ML -web service

Ympäristönä käytettiin samaa Anaconda 2 -ympäristöä, kuin koneoppimisskriptin kehityksessäkin. Ympäristöön asennettiin Python 3.7, scikit-learn 1.0.1 koneoppimismallin lataamista ja käyttöä varten, sekä flask versio 2.2, joka on kirjasto web-palveluiden luontiin Python-kielellä, ja tässä ohjelmassa sitä käytettiin luomaan REST-rajapinta, johon voidaan tehdä HTTP-kutsuja.

Service lataa käynnistyessään Python ML –skriptin luoman ydinestimointimallin tiedostosta, ja tarjoaa siihen *predict()*-funktion kyselyrajapinnan. Web servicen REST-rajapintaan tarvitsi toteuttaa ainoastaan yksi endpoint:

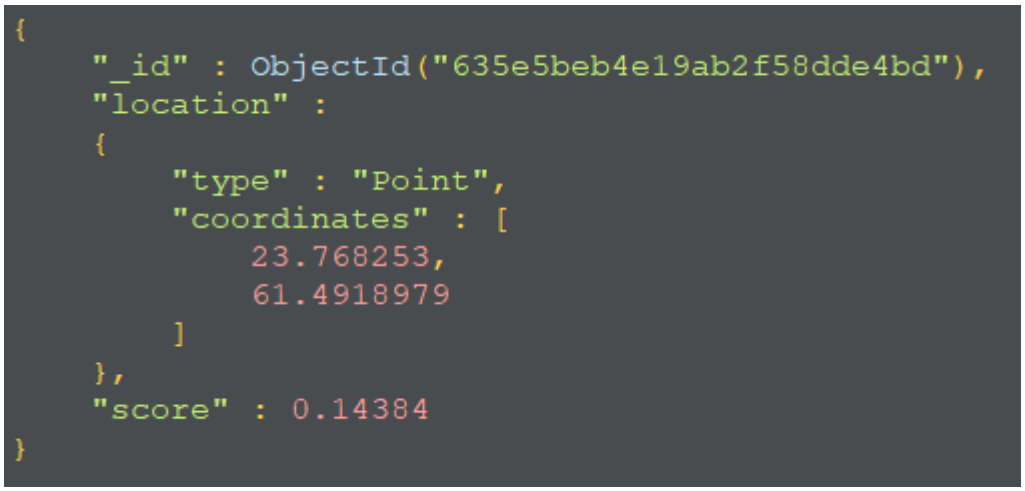
- */theftscore*
    - Ottaa parametreina koordinaatit latitude- ja longitude -muodossa.
    - Palauttaa *score*-arvon kyseiselle sijainnille, joka saadaan koneoppimismallin *predict()*-metodista.

Koodirivejä ML -web servicelle tuli noin 70.

5.3 Ordobike-Backend ja tietokanta

Varkaustasopisteytystä varten tietokantaan tarvitsi luoda uusi dokumenttikokoelma, joka sisältää *Score*-tyypin dokumentteja (KUVA 13). Näitä pisteytystietueita tietokantaan luo aiemmassa kappaleessa esitelty Python ML-skripti. *Score*-dokumenttien rakenne on hyvin yksinkertainen. Ne sisältävät vain kaksi tietuetta:

- *location*: MongoDB:n maantieteelliseen hakuun soveltuva sijaintirakenne. Pitää sisällään *coordinates*-koordinaattikentän, johon tulevat pisteytyssijainnin koordinaatit listamuodossa, sekä *type*-kentän, jonka arvo on aina ”Point”-merkkijono.
- *score*: pisteytysarvo, desimaaliluku väliltä 0-1



KUVA 13. Esimerkki Score-luokan JSON-tietokantadokumentista, jonka Python ML-skripti on luonut.

Jotta varkauspisteytysdata saadaan client-sovellusten käyttöön, tarvitsi Ordobike-Backend-ohjelmaan toteuttaa kaksi uutta REST-rajapinnan endpointia:

- */theftscore* (KUVA 14):
  - Ottaa parametreina clientin sijainnin latitude- ja longitude –koordinaattiarvoina
  - Palauttaa kyseisen sijainnin varkauspisteytyksen desimaalilukuna. Backend tekee tätä varten sisäisesti kutsun yllä kuvaillulle ML -web servicelle.
- */areatheftscores* (KUVA 15):

- Ottaa parametreina suorakulmaisen muotoisen alueen vasemman alalaidan ja oikean ylälaidan sijainnit latitude/longitude -koordinaattipareina.
- Palauttaa annetun alueen sisällä olevat varkauspisteytykset *Score*-dokumentteina tietokannasta JSON-muodossa, käyttäen MongoDB:n maantieteellistä hakua.

```
}).get('/theftscore', (req, res, next) => {
    https.get(config.pythonWebApiUrl + '/theftscore?lat= ' + req.query.lat + '&lon=' + req.query.lon, (resp) => {
        res.status(200).json({
            success: true,
            data: resp.data
        });
    }).on("error", (err) => {
        res.status(500).json({
            success: false,
            error: ree
        });
    });
})
```

KUVA 14. */theftscore* -endpointin toteutus Ordobike-Backendissä.

```
router.get('/areatheftscores', (req, res, next) => {
    const query = {
        location: {
            $geoWithin: {
                $box: [
                    [req.query.blLon, req.query.blLat],
                    [req.query.urLon, req.query.urLat]
                ]
            }
        }
    }

    Score.find(query, (err, scores) => {
        if (err) {
            res.next(err);
        }
        else {
            res.status(200).json({
                success: true,
                message: langManager.getLangString('SETTINGS_SCORES_RETRIEVED', req.query.lang),
                data: scores
            });
        }
    })
```

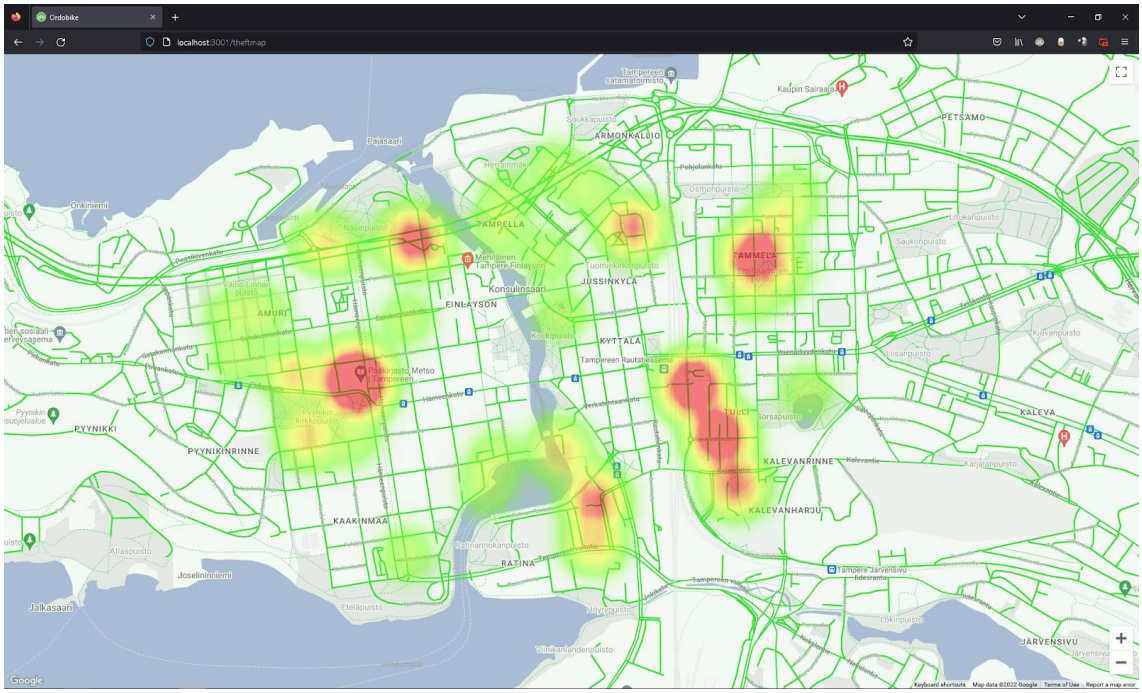
KUVA 15. */areatheftscores* -endpointin toteutus.

Olemassa olevaan Ordobike-Backend -projektiin tuli tämän uuden toteutuksen myötä vain noin 50 uutta koodiriviä.

5.4 Ordobike-Frontend

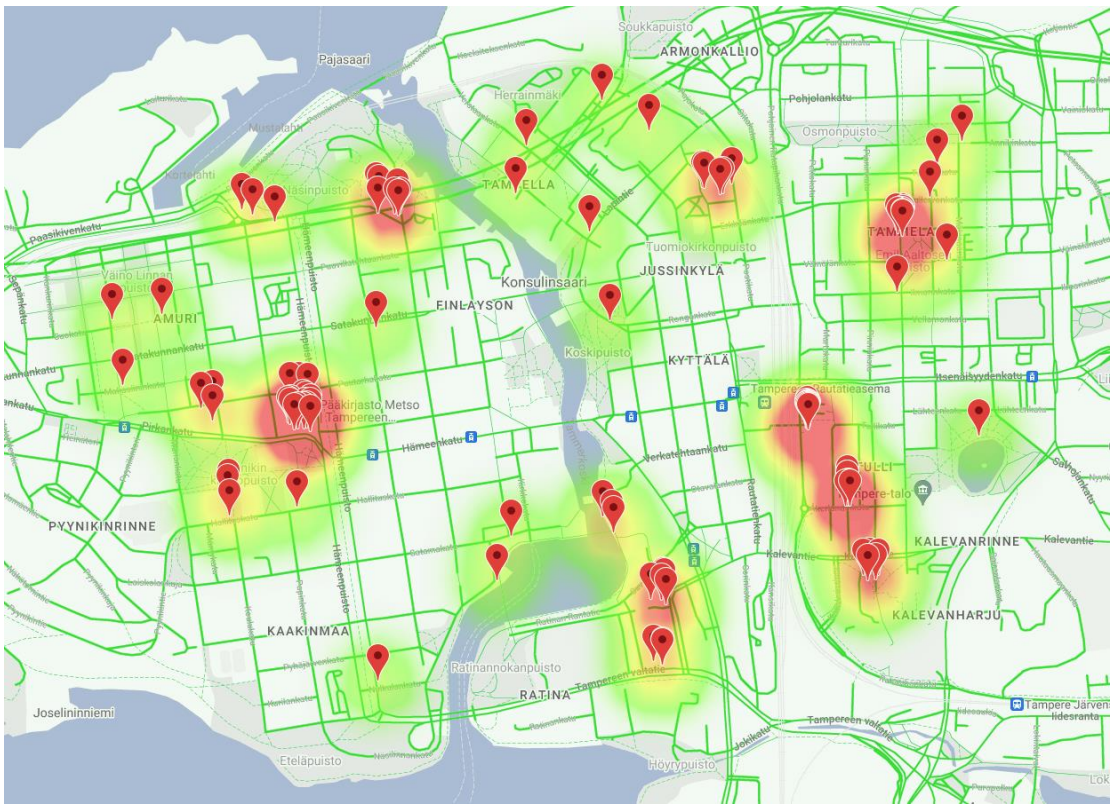
Ordobike-Frontend -websovellukseen toteutettiin uusi varkauskarttasivu. Tälle sivulle lisättiin myös linkki sovelluksen yläpalkkiin, ja siihen pääsevät kaikki käyttäjät, myös kirjautumattomat.

Varkauskarttasivu (KUVA 16) pitää sisällään koko sivun kokoisen Google Maps-karttaelementin, johon latautuvat Ordobike-Backend-sovelluksesta kartan alueen varkauspisteytykset. Näistä pisteytyksistä muodistettiin heatmap-lämpökartta indikoimaan varkauksien tiheyksiä. Väritys on vihreästä keltaiseen ja punaiseen, sitä mukaan kuinka tiheästi varkauksia esiintyy. Heatmap-toiminnallisuus on osa Google Maps –karttaelementtiä, joten sille tarvitsee vain syöttää pisteytysten koordinaatit, sekä itse pisteytys-arvo painotusarvona.



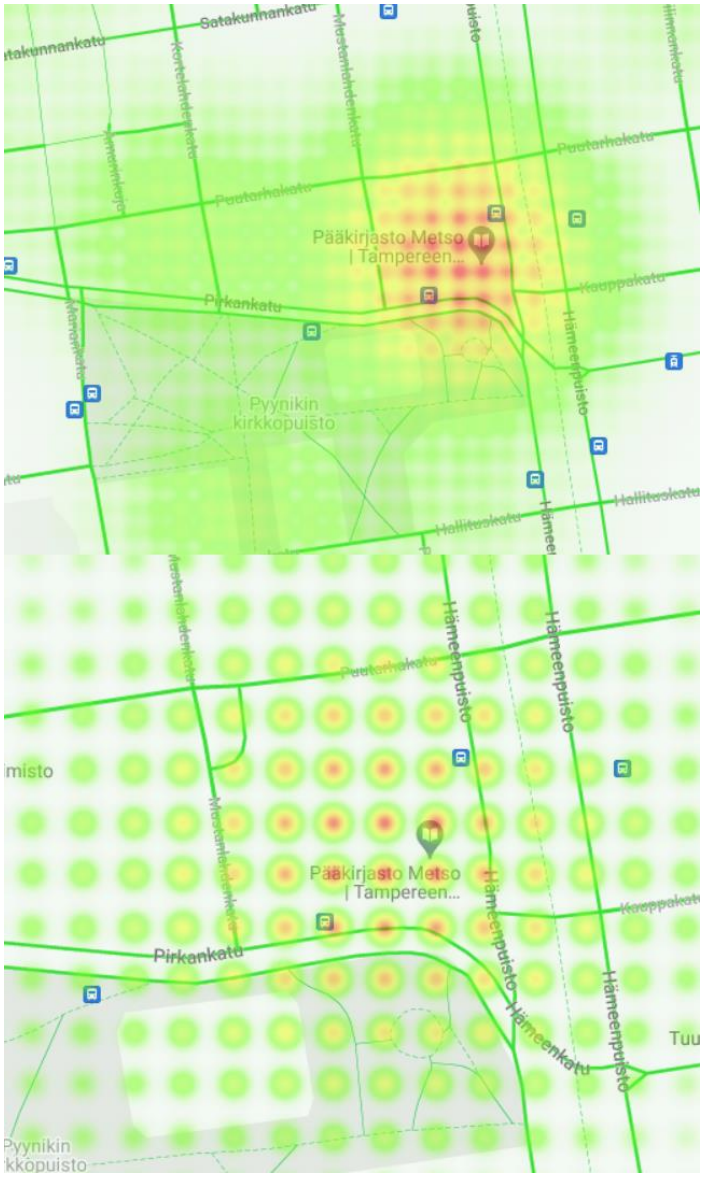
KUVA 16. Varkauskarttasivu internet-selaimessa. Kartan päällä pyörävarkauksien tiheyttä visualisoiva heatmap-kerros.

Muodostetuista heatmap-alueista näytti käyvän ilmi hyvin ne alueet, joille oli keskittynyt isompi määrä varkauksia pienemmälle alueelle, kun verrattiin karttamerkkien paikkoja heatmapiin (KUVA 17).



KUVA 17. Itse varkaustapahtumat varkauskartan heatmapin päällä punaisina merkkeinä.

Kun karttaa zoomasi tarpeeksi alas, havaittiin “rakeisuutta” heatmapin värityksessä (KUVA 18). Tämä johtuu muodostetun koordinaattimatriisin luonteesta, jossa pisteet ovat noin 20 metrin välein. Ongelma ei kuitenkaan ole kovin oleellinen, sillä kartta on tarpeeksi tarkka korkeammallakin zoomaustasolla, jolla rakeisuutta ei esiinny, ja karttaelementin zoom-tasoa voidaan rajata. Jos rakeisuutta haluttaisiin vähentää, se onnistuisi myös tihentämällä koordinaattimatriisia. Tämä kuitenkin lisäisi heatmapin latausaikaa.



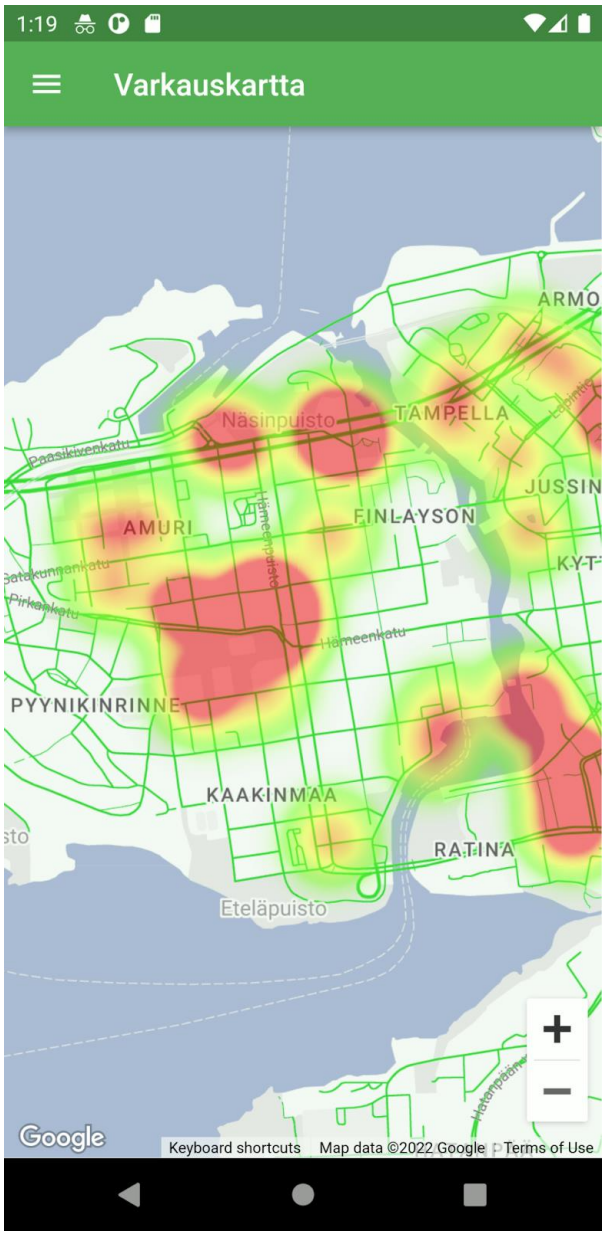
KUVA 18. Esimerkki heatmapin rakeisuudesta, kun varkauskarttaa zoomataan alemmas.

Ordobike-Frontendiin uusia koodirivejä tuli noin 150.

5.5 Ordobike-Mobile

Mobiilisovellukseen haluttiin toteuttaa samanlainen varkauskartta, kuin web-sovellukseenkin, sekä eräänlainen varkausindikaattori, joka kertoisi varkausasteen juuri mobiililaitteen sijainnin kohdalle.

Varkauskartan toteutuksessa päädyttiin käyttämään hyväksi Ordobike-Frontendiin toteutettua karttasivua, koska sopivia karttaelementtikirjastoja ei ollut saatavilla Flutter-ympäristöön, joilla olisi pystynyt toteuttamaan luontevasti heatmap-toiminnallisuuden. Näin ollen mobiilisovellukseen lisättiin vain oma näkymänsä, joka pitää sisällään koko näkymän kokoisen web-selainelementin, joka lataa Ordobike-Frontend -websivun varkauskartan (KUVA 19).



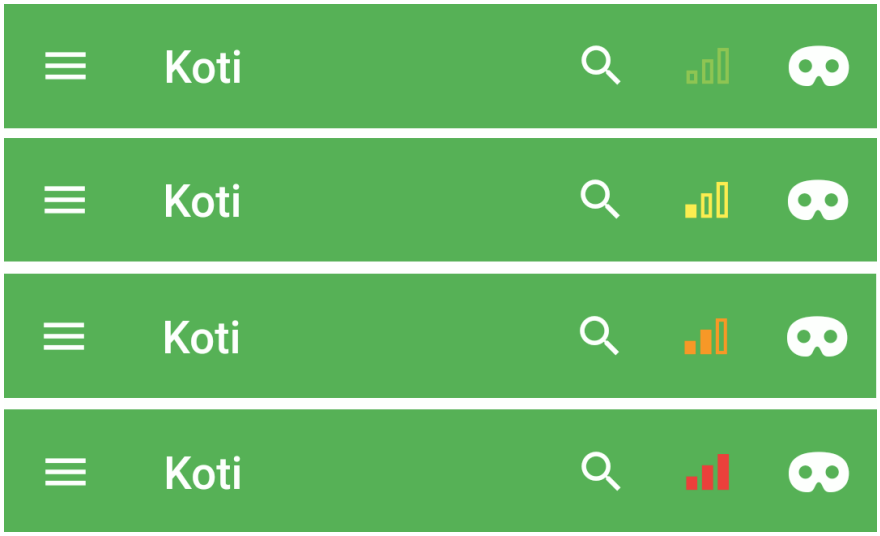
KUVA 19. Varkauskartta Ordobike-mobiilisovelluksessa.

Varkausindikaattoritoiminnallisuutta varten lisättiin mobiilisovelluksen yläpalkkiin, hakutoimintonapin viereen, kaksi erillistä ikonia (KUVA 20). Ensimmäisenä iko-

nina on väritykseltään ja täyttöasteeltaan vaihteleva tasopalkistoikoni. Mitä korkeampi varkauspisteytys mobiililaitteen sijainnissa on, sitä enemmän palkkeja on täynnä, ja ikoni on värjättynä vihreästä, keltaisen ja oranssin kautta, punaiseen. Ikonin palkkien määrä ja väritys määräytyvät seuraavalla pisteytysasteikolla:

- pisteytys 0 – 0.249999: nolla täyttä palkkia, vihreä väritys
- pisteytys 0.25 – 0.499999: yksi täysi palkki, keltainen väritys
- pisteytys 0.5 – 0.749999: kaksi täyttä palkkia, oranssi väritys
- pisteytys 0.75 – 1: kolme täyttä palkkia, punainen väritys

Indikaattoria päivitetään taustalla automaattisesti, lähettämällä Ordobike-Backendin */theftscore*-endpointiin pyyntö sijainnin viimeisimmästä pisteytyksestä, jonka parametreiksi liitetään laitteen sijaintikoordinaatit (KUVA 21 ja KUVA 22). Indikaattorin vieressä on maski-ikoni, jonka katsottiin kehitysvaiheessa saatavilla olevista ikoneista kuvastavan parhaiten varkautta. Molemmat maski-ikoni, sekä indikaattori ovat painettavia, ja painettaessa ne vievät varkauskarttanäkymään.



KUVA 20. Varkausindikaattori (palkisto), eri varkausyleisyyteen perustuvilla täytöillä ja värityksellä, mobiilisovelluksen Koti-näkymän yläpalkissa.

```
Future<double> getTheftScore(
    double latitude, double longitude) {
  var url = apiUrl + '/theftscore?lat=$latitude&lon=$longitude';

  return http.get(Uri.parse(url)).then((response) {
    var responseMap = json.decode(response.body) as Map<String, dynamic>;

    if (responseMap['success']) {
      return responseMap['data'];
    } else {
      throw Exception(responseMap['message']);
    }
  }).catchError((error) {
    print(error.toString());
    throw error;
  });
}
```

KUVA 21. Flutter-koodiin lisätty rajapintakutsu Ordobike-Backendille, annetun sijainnin varkauspisteytyksen hakuun.

```
@override
void initState() {
  Future.delayed(Duration.zero, () async {
    this.loadAssets();
  });

  super.initState();

  this.theftScoreUpdateTimer = Timer(Duration(seconds: 5), () => this.updateTheftScore());
}

Future<void> updateTheftScore() async {
  var location =
          await Provider.of<LocationProvider>(context, listen: false)
              .getLocation();

  this.currentTheftScore = await Provider.of<AssetsProvider>(context, listen: false)
              .getTheftScore(location.latitude, location.longitude);
}
```

KUVA 22. Ote koodista, jossa kotisivun latauduttua käynnistetään varkauspisteytystä päivittävä ajastin, joka kutsuu *updateTheftScore()*-funktiota, jossa päivitetään päätelaitteen sijaintikoordinaatit, sekä tehdään pisteytyshaun rajapintakutsu.

Uusia koodirivejä Ordobike-Mobile-projektiin tuli näiden toiminnallisuuksien myötä noin 250.

6 POHDINTA

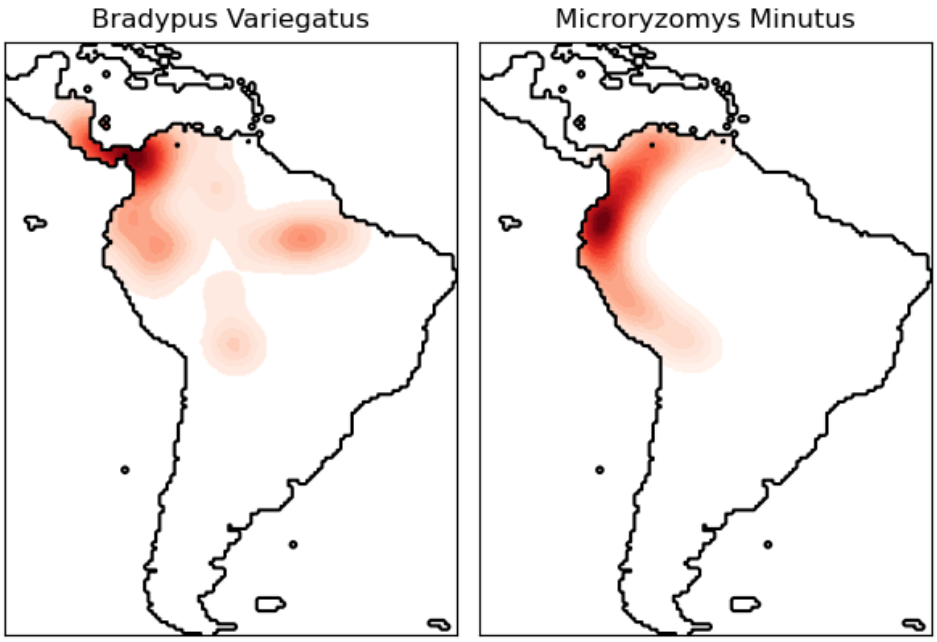
6.1 Datasta

Jo opinnäytetyön alkuvaiheessa kävi nopeasti ilmi, että Ordobike-palvelun ei sisältänyt tarpeeksi dataa, jotta sitä olisi voinut hyödyntää tekoälyominaisuudessa varteenotettavasti. Tämä ei onneksi estänyt työn toteutumista, sillä testidatan luominen ei ollut vaikea tehtävä. Toki oli pieni pettymys, että uutta toiminnallisuutta ei päästy heti hyödyntämään tosielämän tarkoituksiin.

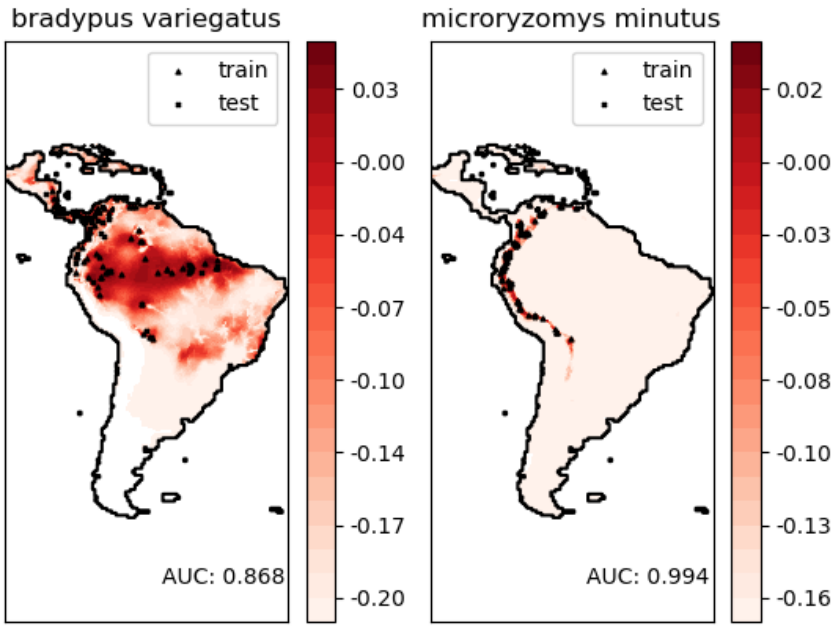
Itse palvelun data paljastui nopeasti sellaiseksi, josta voitaisiin eriyttää juuri sellaista sijaintidataa, jota työn varhaisessa suunnitteluvaiheessa haluttiin. Jatkokehitystä ajatellen datassa on sijaintitietojen lisäksi myös muita ominaisuuksia, joita koneoppimisessa on mahdollista hyödyntää, kuten esimerkiksi pyörävarkauden tapahtuma-aika, josta voitaisiin eriyttää kellonaika päivästä -tyyppinen tieto, ja käyttää sitä yhtenä koneoppimisdatan luokittelutietona.

6.2 Koneoppimisalgoritmeista

Ehkäpä suurin työn toteutuksen aikana syntynyt pohdinnan aihe liittyi työn koneoppimispuoleen. Ensimmäinen kysymys liittyi valitun Kernel Density -ydinestimoinnin validiteettiin: onko se oikeasti “puhdas” koneoppimisalgoritmi? Tämä kysymys syntyi, koska sen käyttö ei sisällä opintojen aikana keskeiseksi koneoppimisvaiheeksi tullutta mallin opetusta ja testausta eritellyillä opetus- ja testidatoilla. Työn loppuvaiheessa, lisätutkintaa tehdessä, paljastui toinen samankaltainen Scikit-Learn -koodikirjaston esimerkkiartikkeli (Prettenhofer & Vanderplas, n. d.), jossa oltiin käytetty opetus- ja testidata -jakauman sisältävää koneoppimisalgoritmia. Käytetty testidata oli sama, ja tässäkin esimerkissä tulostettiin samanlainen havainnollistava kartta, joten lopputuloksia oli kahden algoritmin välillä helppo verrata.



KUVA 23. Scikit-learn -koodikirjaston Kernel Density -ydinestimoinnin tiheysarviointi-esimerkkiartikkelin (n. d.) havainnointikuva eläinlajien esiintymistiheydestä.



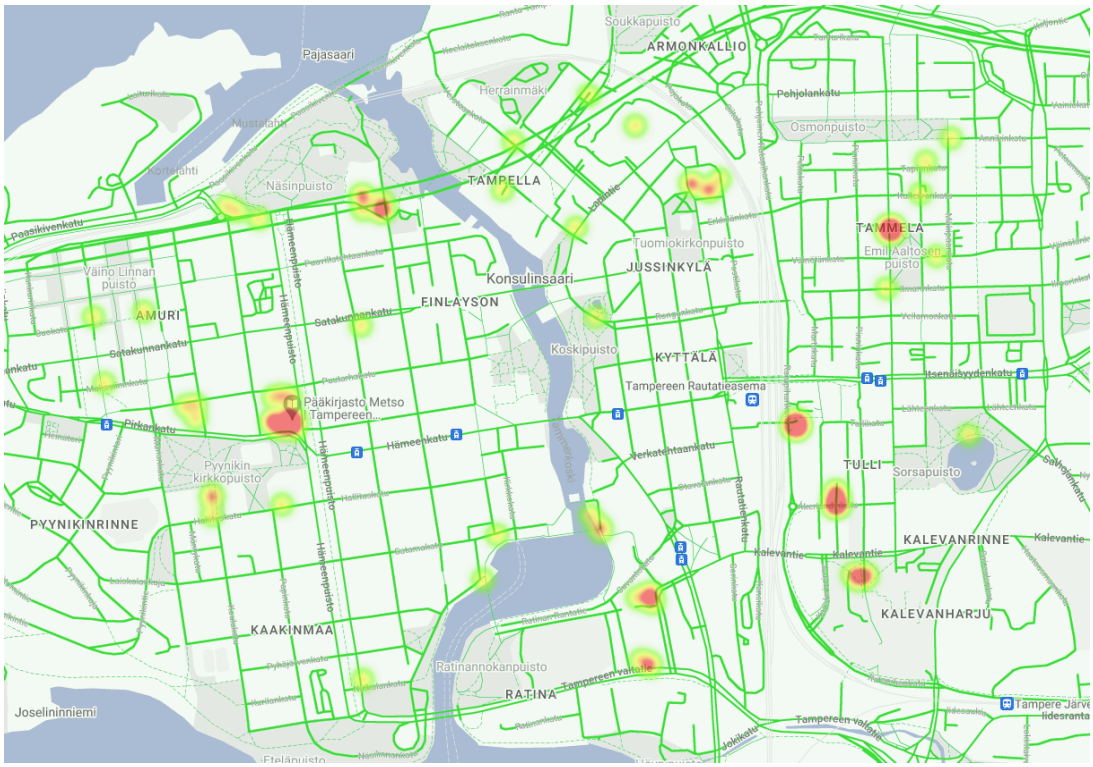
KUVA 24. Saman eläinlajidatan tiheysarviointi-esimerkki OneClassSVM-koneoppimisalgoritmilla (kuva Scikit-learn, n. d.).

Ensimmäinen kuva (KUVA 23) on tähän työhön valitun Kernel Density -ydinestimoinnin havainnollistamiskuva sen Scikit Learn -esimerkkiartikkelista, ja toinen kuva (KUVA 24) on myöhemmin löydetyn OneClassSVM-algoritmin esimerkkiartikkelista. Algoritmien merkkaamissa alueissa voidaan huomata samankaltaisuuksia, mutta myös selkeitä eroja. Ydinestimointi näyttää luovan “suljetumpia” ja selkeämpiä alueita, kun taas OneClassSVM on esimerkiksi toisen eläinlajin tapauksessa värjännyt koko mantereen, joskin hyvin lievän tason arvolla. Tämän perusteella OneClassSVM ei välttämättä soveltuisi pyörävarkauskartan heatmap-alueiden luontiin niin hyvin, koska se voisi merkitä myös sellaiset alueet, jotka ovat kaukana pyörävarkauksien tapahtumapaikoista, matalan varkausmäärien alueiksi. Tiheimpien keskittymien indikointiin algoritmi taas vaikuttaa soveltuvan. Olisi ollut kiintoisaa myös opinnäytetyön puitteissa toisena vaihtoehtona implementoida OneClassSVM pyörävarkausdatan kanssa käytettäväksi, mutta valitettavasti sen löytäminen tapahtui niin myöhäisessä vaiheessa työn kulkua, että aikataulullisten syiden johdosta näin ei päätetty tehdä. Tulevaisuutta ajatellen, itse algoritmin vaihtaminen olisi suhteellisen helppoa, koska se vaatisi muokkausta vain Pyhton-ML -skriptiin. Toisaalta, jo valitun Kernel Density -ydinestimoinnin kautta saadut tulokset eivät näytä oleellisesti huonommilta, joten kriittistä tarvetta algoritmien vaihtamiselle ei tässä vaiheessa nähdä.

Mitä tulee alkuperäiseen pohdintakysymykseen “oliko valittu ydinestimointi puhdasta koneoppimista?”, Kernel Densityn käyttämä Ball Tree -algoritmi luetaan luokittelualgoritmien alle, jotka taas ovat osa koneoppimisalgoritmeja. Teknisesti tapa siis oletettavasti on koneoppimista, vaikka siinä ei perinteistä mallin opetusvaihetta olekaan. Scikit Learnin dokumentaatio (Vanderplas, n. d.) ydinestimoinnista toteaakin, että se on sekoitus ohjaamatonta oppimista (unsupervised learning), ominaisuuksien suunnitteluprosessia (feature engineering), ja tietomallinnusta (data modeling).

Toiseksi pohdinnan aiheeksi heräsi myös kysymys valitun koneoppimistavan hyödyllisyydestä, toisin sanoen olisiko kaiken koneoppimistoiminnallisuuden kehittämisen voinut sivuuttaa, ja syöttää pyörävarkaussijainnit vain suoraan heatmap-komponentille, ja saatu suurin piirtein sama tulos? Tämän kartoittamiseksi tehtiin väliaikainen muutos varkauskarttaelementtiin, jossa varkaussijainnit syötettiin sen heatmap-komponentille suoraan (KUVA 25). Tulosta voi luonnehtia

suuntaa-antavan samankaltaiseksi, mutta siinä on oleellisia eroja koneoppimistulosten tiheyskarttaan. Tiheimmistä esiintymistä saa kyllä indikaation punaisten kohtien perusteella, mutta ne ovat paljon pienempiä alueita kuin koneoppimistulosten maalaamat. Voisi argumentoida, että pyörävarkaat eivät liiku eksakteilla sijainneilla, esimerkiksi varasta aina vain samassa pyörätolpassa kiinni olevia pyöriä, vaan varkauksia tehtaillaan ennemminkin tietyllä alueella, esimerkiksi kaupunkikorttelissa. Tällöin olisi myös perusteltua väittää, että koneoppimistulosten tiheyskartan laajemmat punaiset alueet antavat parempaa osviittaa siitä, mikä alue yleisesti on suosittua varkaiden keskuudessa.



KUVA 25. Esimerkkidatan varkaussijainnit syötettynä suoraan heatmapille.

6.3 Työn laajuudesta, lisäideoista, tuotteistamisesta, ja lopputuloksista

Opinnäytetyön tulosten tuotteistaminen oli keskeinen asia, joka työn laajuudesta oli valitettavaa jättää pois. Työn alkuvaiheessa kävi kuitenkin selväksi, että tuotteistaminen pitäisi sisällään laajojen, paljon työtä vaativien ratkaisujen, toteuttamisia, jotka olisivat liian suuria opinnäytetyöhön. Itse koodirivejä ei loppujen lopuksi komponentteihin tullut, mutta selvitystyö niiden aikaansaamiseksi oli suuri.

Itse laajemmat ominaisuudet, joita tuotteistetussa toiminnallisuudessa voisi olla, olisi mm. 6.1 -kappaleessa mainittu varkauksien aikatiedon tuonti mukaan koneoppimisen parametriksi, ja tämän myötä ehkä myös itse algoritmin vaihto sopivampaan.

Tuotteistamisessa ehkä isoin miettimisen ja työn kohde olisi, miten toteuttaa toiminnallisuus laajalla skaalalla. Tämän työn PoC-toteutuksessa alue pidettiin pienenä, Tampereen ydinkeskustan kokoisena, jolloin testidata ja myöskin koneoppimismallista ulos saadun datan määrä pysyivät siedettävän kokoisena, joka mahdollisti esimerkiksi sen, että palvelimelta voitiin kysyä suoraan koko karttaalueen varkaustasopisteytykset, eikä tämä vaikuttanut kriittisesti palvelun nopeuteen. Kun skaalassa noustaan koko Suomen, tai vaikkapa koko maapallon, laajuuteen, kasvaa myös välitettävän datan määrä, jolloin tarvitsee tuoda mukaan myös resurssien, eli laskentatehon ja siirrettävän datan määrän optimointia. Keinoja näihin olisi esimerkiksi koneoppisajojen tekeminen eri skaalan tuloksille, esimerkiksi maa-alueen laajuuden perusteella. Näin karttaan voitaisiin ladata dataa dynaamisesti sen zoomaustason perusteella. Koneoppimisajoja voitaisiin tehdä myös osissa pienemmille maa-alueille kerrallaan, jolloin ajo ei käyttäisi muistia ja laskentatehoa yhdellä kertaa isoa määrää.

Loppujen lopuksi, vaikka opinnäytetyö sisälsikin pettymyksiä, sai siitä hyvän kokonaislaatuisen käytännön kokemuksen, millaista on ottaa datankäsittely ja koneoppiminen osaksi ohjelmistoa, kaikkine eri vaiheineen.

LÄHTEET

Liikenne- ja viestintäministeriö. 2021. Kunnille 28,5 miljoonaa euroa valtionavustusta kävely- ja pyöräilyinfran parantamiseen. Verkkosivu. Viitattu 14.10.2022. https://www.lvm.fi/-/kunnille-28-5-miljoonaa-euroa-valtionavustusta-kavely-ja-pyorailyinfran-parantamiseen-1387000

Palmgren, 2021. Pyörävarkaudet mittaushistorian ennätykseen – menopelejä vietiin 11 miljoonan euron edestä. Finanssiala 21.6.2021. Viitattu 14.10.2022. https://www.finanssiala.fi/uutiset/pyoravarkaudet-mittaushistorian-ennatykseen-menopeleja-vietiin-11-miljoonan-euron-edesta/

Von Bell, 2020. Poliisi: Polkupyörävarkaudet ovat usein osa huumausainerikollisuutta. Autotoday 15.8.2020. Viitattu 14.10.2022. https://autotoday.fi/poliisi-polkupyoravarkaudet-ovat-usein-osa-huumausainerikollisuutta/

Anzer & Bauer, 2021. 2021. A Goal Scoring Probability Model for Shots Based on Synchronized Positional and Event Data in Football (Soccer). Frontiers. Verkkosivu. Viitattu 15.8.2022. https://www.frontiersin.org/articles/10.3389/fspor.2021.624475/full

Kokonaisturvallisuuden sanasto (TSK 50). 2017. Paikkatieto. Luettu 28.8.2022.

Polaris Intelligence, 2020. What is Location Intelligence? Verkkosivu. Viitattu 3.9.2022. https://www.polarisintelligence.com/what-is-location-intelligence/

Esri, n. d. Location Intelligence. Verkkosivu. Viitattu 3.9.2022. https://www.esri.com/en-us/location-intelligence

Krueger & Han, 2019. Bird's-Eye - Large-Scale Visual Analytics of City Dynamics using Social Location Data. Eurographics Conference on Visualization (EuroVis) 2019, Volume 38 (2019), Number 3

Polaris Intelligence, n. d. COVID-19 Hotspots. Verkkosivu. Viitattu 3.9.2022. https://www.polarisintelligence.com/covid-19-hotspots-finding-high-risk-populations-using-polaris/

Texas State University, n. d. Overview of Geographic Profiling. Verkkosivu. Viitattu 3.9.2022. https://www.txst.edu/gii/geographic-profiling/overview.html

Vanderplas, n. d. Kernel Density Estimate of Species Distributions. Scikit Learn. Verkkosivu. Viitattu 3.9.2022. https://scikit-learn.org/stable/auto_examples/neighbors/plot_species_kde.html

Marius, 2020. Tree Algorithms Explained: Ball Tree Algorithm vs. KD Tree vs. Brute Force. Towards Data Science. Verkkosivu. Viitattu 10.11.2022. https://towardsdatascience.com/tree-algorithms-explained-ball-tree-algorithm-vs-kd-tree-vs-brute-force-9746debcd940

Vanderplas, n. d. Nearest Neighbors. Scikit Learn. Verkkosivu. Viitattu 10.11.2022. https://scikit-learn.org/stable/modules/neighbors.html#neighbors

Geeks for Geeks, 2022. Haversine formula to find distance between two points on a sphere. Verkkosivu. Viitattu 3.9.2022. https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/

Tilastokeskus, n. d. Tiheyspintakartta esiintymistiheyden kuvaajana. Verkkosivu. Viitattu 27.7.2022. https://tilastokoulu.stat.fi/verkkokoulu_v2.xql?page_type=esim&course_id=tkoulu_teemak&lesson_id=7&subject_id=2&example_id=1

GIS Wiki, 2011. Decimal degrees. Verkkosivu. Viitattu 31.10.2022. http://wiki.gis.com/wiki/index.php/Decimal_degrees

Prettenhofer & Vanderplas, n. d. Species distribution modeling. Scikit Learn. Verkkosivu. Viitattu 25.10.2022. https://scikit-learn.org/stable/auto_examples/applications/plot_species_distribution_modeling.html

Vanderplas, n. d. Density Estimation. Scikit Learn. Verkkosivu. Viitattu 10.11.2022. https://scikit-learn.org/stable/modules/density.html