Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Henri Juppi

Koneoppimiskehitys ML.NET-sovelluskehyksellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

7.5.2019

Tekijä Otsikko Sivumäärä Aika	Henri Juppi Koneoppimiskehitys ML.NET-sovelluskehyksellä 43 sivua + 2 liitettä 7.5.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	pelisovellukset
Ohjaaja	lehtori Miikka Mäki-Uuro

Insinöörityöprojektin yhtenä tarkoituksena oli kehittää koneoppimisratkaisuja projektin teettäneen yrityksen datalla. Kehittämisen työkaluksi valittiin ML.NET-sovelluskehys, joka on ilmainen avoimen lähdekoodin koneoppimissovelluskehys. Insinöörityössä tutkittiin, miten ML.NET-sovelluskehystä käytetään ja mitä se tarjoaa.

Insinöörityöprojektissa oli tarkoituksena kehittää keinoja auttaa käyttäjiä dokumenttien luokittelussa ja samankaltaisten dokumenttien löytämisessä projektin teettäneen yrityksen .NET-sovelluksissa. Näitä ongelmia ratkaistiin ja toteutettiin ML.NET-sovelluskehyksellä. Insinöörityöprojektissa kehitettiin työkalu, joka helpottaa uusien koneoppimisratkaisujen kehittämistä. Työkalusta kehitettiin konsolisovellus, jonka toimintaa voitiin säätää työkaluun laadituilla asetuksilla.

Kehitystyö aloitettiin dokumenttien luokittelulla, johon keskityttiin insinöörityön aikana eniten. Dokumenttien luokittelulla tavoiteltiin luokittelun automatisointia ilman, että käyttäjien tarvitsisi luokitella dokumentteja itse. Mittaustuloksien perusteella tavoitetta ei saavutettu kokonaan. Mittauksilla kuitenkin päästiin sellaiseen tulokseen, että dokumenttien luokittelun automatisointia voidaan yrittää tuotantosovelluksissa.

Dokumenttien luokittelun jälkeen tehtiin suomen ja englannin kielen luokittelija. Tämän kielten luokittelijan mittaustulokset olivat lähes täydelliset. Luokittelijaa voi käyttää samankielisten dokumenttien etsimisessä tai datan esikäsittelyssä suomen- ja englanninkielisissä teksteissä.

Lopuksi kehitystyössä tehtiin dokumenttien ryhmittelyä, jonka avulla pyrittiin löytämään samankaltaisia dokumentteja, joita voitaisiin ehdottaa käyttäjille. Dokumenttien ryhmittelyllä onnistuttiin luomaan selkeitä ryhmiä datasta. Samankaltaisilla dokumenteilla voidaan yrittää mitata ryhmittelyn vaikutuksia käyttäjien katseluhistoriaan tuotantosovelluksissa.

Avainsanat	ML.NET, koneoppiminen, luokittelu, ryhmittely

Author	Henri Juppi
Title	Machine Learning Development Using ML.NET Framework
Number of Pages	43 pages + 2 appendices
Date	7 May 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Game Applications
Instructor	Miikka Mäki-Uuro, Senior Lecturer

The purpose of this thesis project was to develop machine learning solutions using data provided by a company that initiated the project. The chosen tool for developing machine learning solutions was ML.NET framework which is free and open-source machine learning framework.

The purpose of developing machine learning solutions was to develop ways to help the company's .NET applications' users in classifying documents and finding similar documents. These machine learning problems were solved and implemented in ML.NET framework. A tool was made to help develop new machine learning solutions in the project.

The thesis project was started with a focus on document classification. The goal of document classification was to automate the whole classification of documents in production. The results of document classification showed that automation cannot be achieved totally, but the results were good enough so that the document classification could be tried in production.

Next in this thesis project, a language classification model was developed which could classify a text in a document as English or Finnish. The results of the language classification model were almost perfect. The language classification model could be used for searching documents with the same language or processing the data where documents have English or Finnish text.

The last machine learning solution was document clustering in the thesis project. The goal of the document clustering was to find similar documents and suggest the users unseen documents based on their historical data. The document clustering could successfully create distinct clusters from the data. The document clustering model could be tried in production, and its effect could be measured in the users' document viewing history.

Keywords	ML.NET, machine, learning, classification, clustering

Sisällys

Lyhenteet

PCA	*Principal Components Analysis*. Tärkeimpien komponenttien analyysi, jota käytetään datan visualisoinnissa tai esikäsittelyssä.
ANN	*Artificial Neural Network*. Keinotekoinen neuroverkko, joka on perusta syväoppimiselle.
CNN	*Convolutional Neural Network*. Konvoluutioneuroverkko, jota käytetään kuvien tunnistamisessa.
RNN	*Recurrent Neural Network*. Takaisinkytkeytyvä neuroverkko, jota käytetään puheen ja käsinkirjoitetun tekstin ymmärtämisessä.
NLP	*Natural Language Processing*. Luonnollisen kielen käsittely, jolla tarkoitetaan puhutun tai tekstimuotoisen kielen oppimista ja ymmärtämistä.
BoW	*Bag-of-Words*. Sanasäkki, joka on tyypillinen tapa esittää luonnollisen kielen käsittelyssä tekstejä tai dokumentteja vektoreina.
TF-IDF	*Term Frequency-Inverse Document Frequency*. Termien merkityksellisyyden pisteyttäminen dokumenttikokoelmassa.
CNTK	*Cognitive Toolkit*. Microsoftin avoimen lähdekoodin syväoppimisen kirjasto Python-, C#- tai C++-ohjelmille.
ONNX	*Open Neural Network Exchange*. Avoin neuroverkon vaihtoformaatti eli formaatti, jolla voi siirtää koneoppimismalleja sovelluskehyksistä toiseen.
JSON	*JavaScript Object Notation.* Yleinen tiedostomuotostandardi.
XML	*Extensible Markup Language.* Tiedostomuoto dokumenttien tallentamiseen ja datan välittämisessä.
IDE	*Integrated Development Environment*. Ohjelmointiympäristö, jossa on tekstieditori, ohjelmointikielen kääntäjä, debuggeri ja profiloija.

1 Johdanto

Koneoppimisessa on paljon konsepteja, joita on vaikea ymmärtää ja toteuttaa. Onneksi korkean tason ohjelmointirajapintojen ansiosta niistä on tehty mahdollisimman yksinkertaisia. Nykyään on useita koneoppimissovelluskehyksiä, jotka tarjoavat tällaisia mahdollisuuksia. Koneoppimissovelluskehyksistä saattaa olla vaikea päättää, mitä kannattaa käyttää ja mihin tarkoitukseen. Siksi on hyvä tietää ensin, mitä kaikkea on mahdollista tehdä koneoppimisella ja millaisella datalla, jotta tietää, mitä koneoppimissovelluskehystä käyttää.

Tässä insinöörityössä valittiin Microsoftin avoimen lähdekoodin ML.NET-koneoppimissovelluskehys koneoppimisratkaisujen kehittämiseen. Insinöörityössä tutkittiin, miten ML.NET-sovelluskehystä käytetään ja mitä se tarjoaa. ML.NET-sovelluskehyksellä kehitettiin insinöörityöprojektissa käytännön koneoppimisratkaisuja tuotantosovelluksiin. Kehitystyössä yksi tarkoitus oli tuottaa työkalu, joka helpottaisi useiden koneoppimisratkaisujen kehittämistä.

Kehitystyön aikana laadittiin erilaisia koneoppimisongelmien ratkaisuja projektin teettäneen yrityksen datalla. Koneoppimisratkaisuja tehtiin yritykselle, joka haluaa käyttää ja hyödyntää koneoppimista sovelluksissaan. Koneoppimisella pyritään helpottamaan ja auttamaan yrityksen sovelluksien käyttäjiä. Koneoppimisen tarkoitus on tuottaa lisäarvoa yrityksen sovelluksien käyttäjille.

Insinöörityössä keskityttiin koneoppimisratkaisuihin, joihin kuului dokumenttien luokittelu ja samankaltaisten dokumenttien löytäminen. Dokumenttien luokittelulla pyrittiin automatisoimaan uusien dokumenttien luokittelu kokonaan. Samankaltaisilla dokumenteilla pyrittiin löytämään käyttäjille uusia dokumentteja, jotka ovat samankaltaisia käyttäjien katseluhistorian perusteella. Kehitystyössä näihin pyrittiin selvittämään ratkaisuja ja toteutuksia, joista esitellään tuloksia ja päätelmiä kuvilla ja taulukoilla. Tuloksien perusteella pyrittiin selvittämään koneoppimisratkaisujen käytettävyyttä.

2 Koneoppiminen

Tässä luvussa selostetaan yleiskäsitteitä koneoppimisesta. Aluksi luvussa 2.1 käsitellään koneoppimisen opetusmenetelmiä, jossa selostetaan yleisesti, mitä koneoppimisessa on mahdollista tehdä ja minkälaisia koneoppimistapoja on käytettävissä. Luvussa 2.2 käsitellään, mitä syväoppiminen on ja mitä keinotekoiset neuroverkot ovat. Syväoppimisen luvussa esitellään muutamia yleisiä keinotekoisia neuroverkkoja ja niitten käyttötarkoituksia. Lopuksi luvussa 2.3 puhutaan luonnollisen kielen käsittelystä, joka ei varsinaisesti ole koneoppimisen osa-alue, mutta koneoppimisessa sitä voidaan käyttää hyödyksi tekstin tai puheen opettamisessa.

2.1 Koneoppimisen opetusmenetelmät

Koneoppiminen on tekoälyn alalaji, jossa tietokonetta pyritään opettamaan ja saamaan se toimimaan yhtä hyvin tai paremmin kuin ihmiset datan ja havaintojen perusteella. Tietokonetta pyritään opettamaan siten, ettei opetettavaa ilmiötä varsinaisesti ohjelmoida tietokoneelle. Koneoppimisessa on tarkoitus löytää datasta säännöksiä, joita ihmisten on vaikea löytää tai nähdä. Opettaminen tapahtuu siten, että tietokoneelle syötetään dataa ja informaatiota, jossa oppimisalgoritmi pyrkii tekemään yleistettyjä ennustuksia tai analyyseja syötetyn datan ja informaation perusteella. [1.]

Oppimisalgoritmit määrittävät, mihin ongelmaan ne soveltuvat parhaiten, ja siksi niitä ryhmitellään yleensä opetusmenetelmien mukaan. Opetusmenetelmiin kuuluvat ohjattu oppiminen, ohjaamaton oppiminen ja puoliohjattu oppiminen. Oppimisalgoritmeja voidaan ryhmitellä myös samankaltaisen toiminnallisuuden mukaan, joita ovat esimerkiksi luokittelu, regressio, ryhmittely tai syväoppiminen. [1.]

Ohjattu oppiminen

Ohjattu oppiminen on opetusmenetelmä, jolla pyritään opettamaan tietokoneelle opetettavan datan sisältö. Oppimisalgoritmi tekee datasta itsenäisesti arvauksia, joiden virheitä minimoidaan. [2.] Opetustilannetta voidaan ajatella tilanteena, jossa opettaja seuraa oppimista ja oikaisee virheitä [3]. Opettaminen loppuu, kun oppimisalgoritmi saavuttaa hyväksyttävän tason [3]. Opettaminen tuottaa mallin, joka pystyy tekemään

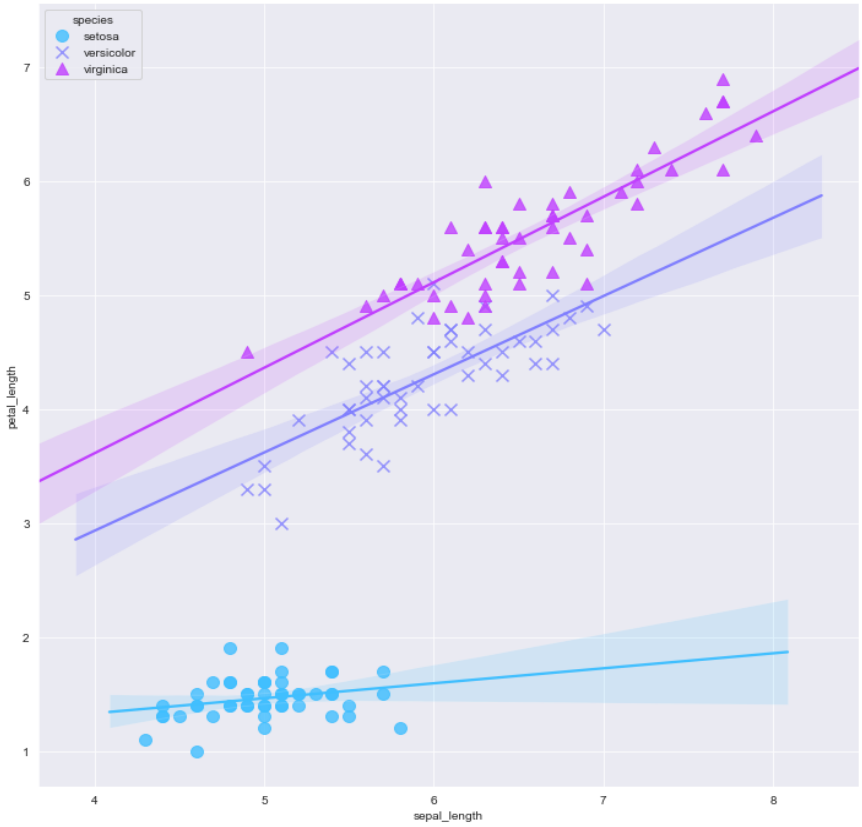
ennustuksia ja arvauksia aikaisemman opetusdatan perusteella uusista sisällöistä, joita malli ei ole ennen nähnyt. Ohjatun oppimisen ongelmat voidaan jakaa kahteen alakategoriaan, luokitteluun ja regressioon. [2.]

Luokittelu

Luokittelulla tarkoitetaan sellaista ohjattua oppimista, jossa ennustetaan jotain laadullista tai kategorista luokkaa. Luokitteluun soveltuu esimerkiksi ihmisten sukupuolien tai sairauksien ennustaminen. [4, s. 28.] Luokittelu vaatii, että opetettavassa datassa on kaksi tai useampia luokkia. Opettamalla pyritään ennustamaan kategorian luokittelu uudelle syötteelle. Luokittelua kutsutaan *binääriluokitteluksi* (engl. binary classification), jos mahdollisia luokkia on vain kaksi. Tähän voidaan ajatella ongelmia, jotka pyrkivät selvittämään esimerkiksi ”kyllä”- tai ”ei”-tuloksia. Luokittelua kutsutaan *monen luokan luokitteluksi* (engl. multi-class classification), jos mahdollisia luokkia on enemmän kuin kaksi. Monen luokan luokittelussa pyritään ennustamaan esimerkiksi hedelmän nimi kaikista mahdollisista hedelmistä. Viimeiseksi luokittelua kutsutaan *monen tuloksen luokitteluksi* (engl. multi-label classification), jos syötteet voivat kuulua useaan luokkaan samanaikaisesti. [5.]

Regressio

Regressiolla tarkoitetaan sellaista ohjattua oppimista, jossa ennustetaan jotain määrällistä tai jatkuvaa arvoa. Regressiota voidaan käyttää ennustettaessa jonkin asian kokoa, määrää tai pituutta. Regressiolla voidaan esimerkiksi ennustaa ihmisten pituutta, ikää tai tuloja. Regressiolla voidaan myös ennustaa esimerkiksi talojen arvo ja tai osakkeiden hintaa historiallisen datan perusteella. [4, s. 28.] Joskus regressio voidaan ajatella luokitteluksi. Tämä tapahtuu niin, että määritellään jollekin jatkuvalle arvolle raja, joka määrittää, mihin luokkaan kaikki arvot rajan yläpuolella ja alapuolella kuuluvat. Esimerkiksi ihmisten pituudet voidaan määrittää kolmeen luokkaan, joissa alle 150 cm pitkät ihmiset ovat lyhyitä, yli 200 cm pitkät ihmiset ovat pitkiä ja ihmiset rajojen välissä ovat keskipituisia. [5.] Kuvassa 1 havainnollistetaan lineaarista regressiota, jossa iiriskukan verholehden ja terälehden perusteella voidaan luokitella iiriskukan laji.



Kuva 1. Lineaarinen regressio iiriskukkadatajoukolla, jossa on 150 datapistettä ja kolmen iiriskukkalajin verholehden pituus terälehden pituuden funktiona [6].

Ohjaamaton oppiminen

Ohjaamaton oppiminen on opetusmenetelmä, jossa opetettava data sisältää syötteitä ja niistä ei tiedetä vastauksia. Ohjaamaton oppiminen on yleensä paljon haastavampaa, koska lopputuloksia ei tiedetä samalla tavalla kuin ohjatussa oppimisessa. Ohjaamattomassa oppimisessa tarkoituksena on löytää datasta yhteyksiä, rakenteita ja kaavoja. Datasta voidaan pyrkiä etsimään ryhmittelyllä esimerkiksi samankaltaisia henkilöitä tai samoista tuotteista kiinnostuneita henkilöitä heidän ostohistoriansa perusteella. Ohjaamatonta oppimista voidaan käyttää datan esikäsittelyssä ohjattua oppimista varten tai datan visualisoinnissa, johon tärkeimpien komponenttien analyysi soveltuu. [4, s. 373–374.]

Tärkeimpien komponenttien analyysi

Tärkeimpien komponenttien analyysi (engl. principal components analysis, PCA) tarkoittaa sitä prosessia, miten tärkeimmät komponentit lasketaan ja miten komponentteja käytetään datan ymmärtämisessä. Komponenteilla tarkoitetaan datasta löytyviä tärkeimpiä ominaisuuksia. Komponenttien on tarkoituksena löytää datasta matalan ulottuvuuden kuvaus, joka sisältää mahdollisimman paljon informaatiota alkuperäisestä datasta. Toisin sanoen datasta karsitaan mahdollisia turhia ominaisuuksia pois. PCA on ohjaamattoman oppimisen keino tuottaa ohjattuun oppimiseen tarkoitettuja arvoja tai visualisoida dataa. Esimerkiksi kahden ulottuvuuden datan kuvausta voidaan visualisoida kuvaajassa. [4, s. 375.]

PCA etsii pienen määrän ulottuvuuksia, jotka ovat mahdollisimman tärkeitä. Tärkeyttä mitataan siten, miten paljon datassa ominaisuudet vaihtelevat. Jokainen tärkein komponentti on datan ominaisuuksien lineaarinen yhdistelmä, jossa kaikista tärkeimmällä komponentilla on suurin varianssi. Varianssilla mitataan ominaisuuksien hajontaa ja vaihtelevuutta. Jokainen seuraavaksi tärkein komponentti löydetään laskemalla ominaisuuksista lineaarinen yhdistelmä, jolla on seuraavaksi suurin varianssi, mutta ne eivät ole korrelaatiossa toisten komponenttien kanssa. [4, s. 375–376.]

Ryhmittely

Ryhmittely (engl. clustering) on ohjaamattoman oppimisen keino analysoida datasta selkeitä ominaisuuksia, jotka ovat samankaltaisia tai erilaisia keskenään ja siten sijoittuvat omiin ryhmiin. Esimerkiksi datasta voidaan tunnistaa ihmisryhmiä, jotka todennäköisemmin ostavat jotain tiettyä tuotetta. Ryhmiteltävästä datasta täytyy olla tuntemusta, jotta tietää, miksi jotkin datan havainnot ovat samanlaisia tai erilaisia. Ryhmittelyllä yritetään etsiä datasta rakennetta, joka parhaiten kuvaa datan yhtenäisiä ryhmiä, kuten kuvassa 2 on havainnollistettu. Molemmat, ryhmittely ja PCA, pyrkivät yksinkertaistamaan dataa, vaikka ne toimivat eri tavalla. [4, s. 385–386.]

Kuva 2. Simuloidun datajoukon ryhmittely KMeans-algoritmilla, jossa on 200 datapistettä kahdessa ulottuvuudessa [4, s. 386].

Puoliohjattu oppiminen

Puoliohjattu oppiminen on opetusmenetelmä, jossa opetettavan datan syötteistä osa sisältää vastaukset ja osa ei. Tällainen tilanne voi syntyä esimerkiksi silloin, jos syötteitä on helppo saada, mutta tuloksia on paljon vaikeampi kerätä [4, s. 28]. Puoliohjatussa oppimisessa hyödynnetään ohjatun ja ohjaamattoman oppimisen keinoja. Siksi tällaisesta datasta voidaan tehdä oletuksia, joista perusoletukset ovat *tasaiset* ja *ryhmäoletukset* (engl. smoothness and cluster assumptions). Tasaisessa oletuksessa oletetaan, että syötteet, jotka sijoittuvat lähekkäin data-avaruudessa, jakavat todennäköisemmin saman luokan. Ryhmä oletuksessa oletetaan, että syötteet, jotka kuuluvat samaan luokkaan, kuuluvat todennäköisemmin samaan ryhmään. [7.] Näiden oletuksien perusteella data voidaan ensin ryhmitellä, minkä jälkeen lähekkäisiä datapisteitä voidaan tutkia ja luokitella.

Toinen puoliohjatun oppimisen tapa on opettaa mallia iteratiivisesti. Tällä tarkoitetaan, että ensin pyritään opettamaan ohjatun oppimisen malli luokitetulla datalla, josta tiedetään vastaukset. Malli pyrkii sitten luokittelemaan luokittelematonta dataa, joista ei

tiedetä vastauksia. Sellaisiin luokittelemattomiin datapisteisiin asetetaan mallin ennustamat luokat, mihin malli on saanut tarpeeksi korkealla varmuudella vastauksen. Tämän jälkeen malli opetetaan uudestaan samalla luokitetulla datalla, johon on lisätty korkealla varmuudella luokitellut datapisteet. Sitten pyritään uudestaan luokittelemaan jäljelle jäänyttä luokittelematonta dataa. Tätä suoritetaan niin monta kertaa, kunnes malli saavuttaa hyväksyttävän tason tai luokittelematonta dataa ei ole enää jäljellä. [7.]

Vahvistusoppiminen

Vahvistusoppiminen (engl. reinforcement learning) on opetusmenetelmä, jossa tekoälyagentti oppii olemalla vuorovaikutuksessa ympäristön kanssa. Tekoälyagentille määritetään mahdolliset toiminnot, joita se voi suorittaa ympäristössä ja siten oppia toimintojen seurauksista. Jokaisesta toiminnosta tekoälyagenttia palkitaan joko positiivisesti tai negatiivisesti. Tekoälyagentin tarkoituksena on pyrkiä maksimoimaan positiiviset palkinnot ajan kuluessa ja siten oppia optimaalinen tapa suoriutua ympäristössä. Tekoälyagentin ympäristönä voivat toimia esimerkiksi pelit, simulaatio tai oikea maailma. [8.]

2.2 Syväoppiminen

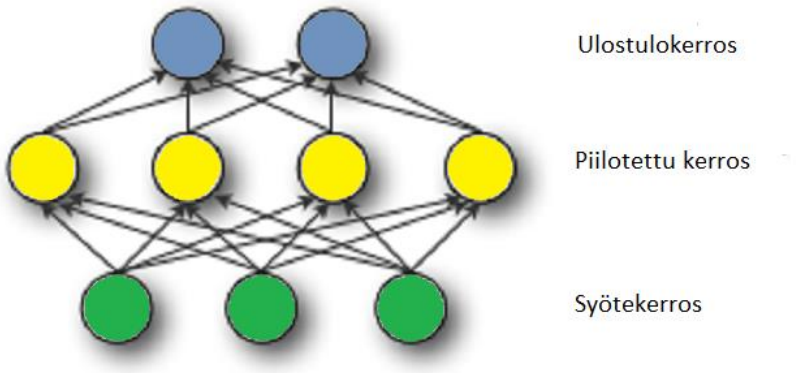
Syväoppimisella (engl. deep learning) tarkoitetaan pääasiassa koneoppimista keinotekoisilla neuroverkoilla (engl. artificial neural network, ANN), jotka on rakennettu biologisten aivojen inspiroimana [9, s. 13]. Keinotekoisilla neuroverkoilla voidaan opettaa tietokonetta oppimaan asioita, jotka ovat ihmisille helppoja ja päivänselviä, mutta tietokoneille vaikeita. Perinteiset koneoppimisalgoritmit eivät ole pystyneet ratkaisemaan tällaisia keskeisiä ongelmia, joihin kuuluu esimerkiksi *konenäkö* (engl. computer vision). Konenäössä tietokone pyrkii tunnistamaan kuvista objekteja, ihmisten naamoja tai käsinkirjoitettua tekstiä. Tällaiset koneoppimisongelmat ovat motivoineet keinotekoisten neuroverkkojen kehityksen. [9, s. 152.]

Keinotekoiset neuroverkot

Keinotekoiset neuroverkot (ANN) koostuvat kerroksista neuroneja, jotka on yhdistetty toisiinsa kuvan 3 mukaisesti. Kerros koostuu samalla tasolla olevista neuroneista ja

yksittäinen neuroni on laskennan tapahtumapaikka. ANN koostuu *syöte-* ja *ulostulokerroksista* (engl. input and output layer), joissa syötekerros ottaa vastaan informaatiota ja ulostulokerros tuottaa informaatiota ANN:stä. [10.] Syöte- ja ulostulokerroksen välissä voi olla kerroksia, joita kutsutaan *piilotetuiksi kerroksiksi* (engl. hidden layer). Kerroksia kutsutaan piilotetuiksi kerroksiksi, koska ANN:lle ei määritetä, mitä sen pitäisi tehdä piilotetuilla kerroksilla. ANN:n täytyy itse päätellä, kuinka piilotettuja kerroksia käytetään parhaan lopputuloksen saamiseksi [9, s. 165].

Neuroverkkoja kutsutaan *mataliksi neuroverkoiksi* (engl. shallow neural network), jos syöte- ja ulostulokerroksen välissä on joko yksi tai ei yhtäkään piilotettua kerrosta, mitä on kuvassa 3 visualisoitu. Neuroverkkoja kutsutaan *syviksi neuroverkoiksi* (engl. deep neural network), jos syöte- ja ulostulokerroksen välissä on enemmän kuin yksi piilotettu kerros, mistä myös syväoppiminen on saanut nimensä [9, s.165]. Neuroverkot muistuttavat hyvin paljon ihmisten hermoverkkoja, koska neuronit aktivoituvat samantapaisesti riittävällä määrällä ärsykkeitä [10].



Kuva 3. Visualisoitu matala neuroverkko, jossa on yksi piilotettu kerros [10].

Konvoluutioneuroverkko

Konvoluutioneuroverkko (engl. convolutional neural network, CNN) on syväneuroverkko, jonka tarkoitus on käsitellä *tensoridataa*. Tensoreilla tarkoitetaan taulukoita, jotka ovat useissa ulottuvuuksissa. Esimerkiksi kahden ulottuvuuden tensori on matriisi, mutta kolmen ulottuvuuden tensori on matriiseja koottuna kuutioksi. Toisin sanoen tensorit ovat taulukkoja asetettuna sisäkkäin, missä sisäkkäin asetettujen taulukoitten määrä voi kasvaa jatkuvasti. [11.] Konvoluutioneuroverkko saa nimensä siitä, että se käyttää matemaattista *konvoluutio-operaatiota* matriisikertolaskun sijaan vähintään yhdessä

neuroverkon kerroksessa [9, s. 326]. Konvoluutiolla tarkoitetaan sellaista integraalia, joka mittaa, kuinka paljon kaksi funktiota ovat päällekkäin jollain ajan hetkellä [11].

Konvoluutioneuroverkkoja käytetään pääasiassa *konenäössä*, koska se antaa mahdollisuuden prosessoida kuvia. Konenäössä tietokone oppii esimerkiksi tunnistamaan objekteja, käsinkirjoitettuja kirjaimia tai numeroita kuvista. Opettaminen tapahtuu luokittelemalla kuvia, ja konvoluutioneuroverkko yrittää nimetä, mitä se havaitsee kuvista. Koska tietokoneet eivät näe kuin ihmiset, kuvat muokataan neliulotteisiksi tensoreiksi. Neliulotteisissa tensoreissa tiedetään kuvien leveydet, pituudet ja värikoodit. Tensorin viimeiseen ulottuvuuteen käytetään konvoluutiota, jolla lasketaan kuvien ominaisuuksia. Kuvien ominaisuuksilla tarkoitetaan kuvien yksityiskohtia, kuten esimerkiksi muotoja tai viivoja. [11.]

Takaisinkytkeytyvä neuroverkko

Takaisinkytkeytyvä neuroverkko (engl. recurrent neural network, RNN) on erikoistettu käsittelemään jaksottaista dataa. Se on tehokas neuroverkko esimerkiksi käsinkirjoitetun tai puhutun kielen käsittelyssä. Takaisinkytkeytyvä neuroverkko palauttaa ulostulojaan takaisinpäin neuroverkossa, mikä vaikuttaa uusiin tuloksiin. Tällä keinolla takaisinkytkeytyvä neuroverkko pystyy tekemään esimerkiksi ennustuksia sanoista vain käsittelemällä sanojen ensimmäiset kirjaimet. Takaisinkytkeytyvää neuroverkkoa voidaan hyödyntää myös ryhmittelyssä, koska se tunnistaa samankaltaisuuksia mittaamalla havaintojen etäisyyksiä toisistaan vektoriavaruudessa. [12.]

2.3 Luonnollisen kielen käsittely

Luonnollisen kielen käsittely (engl. natural language processing, NLP) on tekoälyn alalaji, jossa tietokoneet pyrkivät ymmärtämään ihmisten kieltä ja tekstiä. Kielen ja tekstin ymmärtäminen on haastavaa tietokoneille, koska ne ovat jäsentymätöntä dataa ja niissä ei ole johdonmukaisia sääntöjä. Kieli ja teksti täytyy ensin muokata opetettavaan ja yksinkertaisempaan muotoon, jotta tietokoneet voivat oppia ymmärtämään niitä. [13.] *Tekstin normalisointi* (engl. text normalization) on tekniikka luonnollisen kielen käsittelyssä muokata ja valmistella tekstiä, sanoja ja dokumentteja jatkokäsittelyä varten. Tekstin normalisointiin kuuluu esimerkiksi *perusmuotoistaminen* (engl. lemmatization),

kantasanoittaminen (engl. stemming), *symbolisoiminen* (engl. tokenization) ja *pysäytyssanojen poistaminen* (engl. stop word removal). [14.]

Perusmuotoistamisella tarkoitetaan sanojen muokkaamista perusmuotoihin, ja tähän tyypillisesti tarvitaan hakutaulukko, josta voidaan hakea sanojen perusmuotoja. Kantasanoittaminen on samantyyppinen tekniikka kuin perusmuotoistaminen, mutta kantasanoittaminen löytää sanojen juuren tai kannan. Kantasanoittamisessa ei siis tiedetä, mikä sanan perusmuoto on, vaan pyritään löytämään sanasta kannan, joka ei välttämättä ole mikään oikea sana, mutta kuvaa mahdollisimman monta sanaa. Näitä tekniikoita käytetään siksi, että tietokoneet eivät pysty ymmärtämään sanojen taivutusmuotoja samaksi sanaksi ja siksi sanat täytyy muokata yksinkertaisempaan muotoon. [14.]

Symbolisoiminen on lauseitten pilkkomistekniikka, jossa jokaisessa tekstin välilyönnissä sanat erotetaan lauseista, jolloin jäljelle jää lista sanoja. Jos teksteistä ei eroteltaisi sanoja, tietokoneet käsittelisivät tekstit tai dokumentit yhtenä merkkijonona, jolloin ne eivät oppisi todennäköisesti mitään. Symbolisointi on suoritettava ennen perusmuotoistamista ja kantasanoittamista, jotta jokaista sanaa pystytään käsittelemään erikseen. [14.]

Pysäytyssanat ovat kielissä täytesanoja, jotka esiintyvät usein eivätkä sisällä tärkeää merkitystä tietokoneille. Esimerkiksi englannin kielessä ”a”, ”and” ja ”the” sanat ovat täytesanoja, jotka eivät ole teksteissä tärkeää informaatiota tietokoneille. Pysäytyssanoihin tarvitaan myös tyypillisesti jokaista kieltä kohden oma hakutaulukko, josta poistettavia sanoja voidaan hakea. Pysäytyssanat voidaan poistaa tekstistä symbolisoinnin, perusmuotoistamisen tai kantasanoittamisen aikana. [14.]

Sanasäkki

Sanasäkki (engl. bag-of-words, BoW) on tyypillinen tapa esittää tekstejä tai dokumentteja luonnollisen kielen käsittelyssä. Jokainen teksti tai dokumentti esitetään vektorina. Vektorin pituus on vakio, ja se määritetään jokaisen tekstin tai dokumentin sanavaraston koolla. Jokaisen vektorin arvot ovat indeksoitu sanavaraston sanoilla. Indeksoitujen sanojen arvot vastaavat jokaisen sanan esiintymistä ja määrää tekstissä tai dokumentissa. Tällä tarkoitetaan sitä, että sanat, jotka esiintyvät tekstissä tai

dokumentissa useasti, saavat korkeamman arvon, ja sanat, jotka eivät esiinny koskaan, saavat arvon nolla. Jos täytesanoja ei poisteta sanavarastosta, ne saattavat saada korkeat arvot vektoreissa, koska ne esiintyvät yleensä usein teksteissä. Sanasäkki tuottaa siis monen ulottuvuuden dataa, jossa data on lista teksteistä tai dokumenteista koostettuja vektoreita. Sanasäkkiä voidaan käyttää esimerkiksi tekstien ja dokumenttien ryhmittelyssä tai luokittelussa, jossa oppimisalgoritmit joutuvat keksimään, mitkä vektorit ovat merkitykseltään samanlaisia. [15.]

Termitiheys ja käänteinen dokumenttitiheys

Termitiheys ja käänteinen dokumenttitiheys (engl. term frequency-inverse document frequency, TF-IDF) on tekniikkojen yhdistelmä, jossa termitiheydellä tarkoitetaan jokaisen dokumentin termin pisteyttämistä sen esiintymismäärällä. Sanasäkki on luonnollisen kielen käsittelyssä tällainen tekniikka. Käänteinen dokumenttitiheys pisteyttää puolestaan termien merkityksellisyyttä dokumenttikokoelmassa. Tällä tarkoitetaan sitä, että jos termi esiintyy vain vähässä määrässä dokumenteista, silloin termin täytyy olla merkityksellinen niissä dokumenteissa. Jos termi esiintyy useassa dokumentissa, silloin sen täytyy olla merkityksetön. [16, s. 117–119.]

TF-IDF yhdistää nämä pisteytystekniikat, joilla etsitään dokumenttikokoelmasta merkityksellisiä termejä. Tämä tarkoittaa, että termi saa pisteytyksen seuraavasti:

- Pisteytys on korkeimmillaan, jos termi esiintyy usein pienessä määrässä dokumentteja.
- Pisteytys on matala, jos termi esiintyy vähän pienessä määrässä dokumentteja tai paljon isoissa määrissä dokumentteja.
- Pisteytys on matalimmillaan, jos termi esiintyy lähes kaikissa dokumenteissa.

Tällä tavoin jokaisen dokumentin sanat voidaan TF-IDF-pisteyttää, jolloin dokumenttikokoelmasta voidaan hakea dokumentteja hakukonemaisesti termien perusteella. [16, s. 119.]

3 Työkalut ja kehitysympäristö

Koneoppimiseen on paljon työkaluja, jotka tarjoavat monipuolisesti mahdollisuuksia kehittää koneoppimisratkaisuja. Jotkin työkalut on kohdennettu tiettyihin osa-alueisiin koneoppimisessa, ja siksi saattaa olla haastavaa päättää, mitä työkaluja käyttää ja mihin tarkoitukseen. Tämän takia on tärkeää tietää, mitä koneoppimisella on mahdollista tehdä ja mitä työkaluja on olemassa sekä millaista dataa on saatavilla. Tässä luvussa esitellään ne työkalut, joita kehitystyössä käytettiin. Työkalut valittiin sen perusteella, että ne täyttivät projektin teettäneen yrityksen tarpeita. Yrityksen tarpeisiin kuului esimerkiksi se, että työkaluilla voi kehittää monipuolisia koneoppimisratkaisuja ja niitä on helppo integroida ja käyttää yrityksen kehittämissä .NET-sovelluksissa.

3.1 ML.NET-sovelluskehys

.NET on kehitysympäristö, joka tukee useita ohjelmointikieltä. Se toteuttaa yhteisen kielen infrastruktuurin (engl. common language infrastructure), mikä tarkoittaa, että millä tahansa .NET-ohjelmointikielellä voi koota ja rakentaa sovelluksia ja palveluita .NET-ympäristössä. Nämä .NET-ohjelmointikielet ovat C#, F# ja Visual Basic, joita Microsoft kehittää ja tukee aktiivisesti. [17.]

.NET Core on avoimen lähdekoodin kehitysympäristö, jota ylläpitävät Microsoft ja .NET-yhteisö [18]. .NET Core tukee useita alustoja, kuten Windowsia, Mac OS:ää ja Linuxia. .NET Core toteuttaa *.NET Standardin* ohjelmoitavan rajapinnan, joka on standardisoitu määrittely .NET-ympäristön ohjelmoitavista rajapinnoista. Ohjelmoitavat rajapinnat on tarkoitettu saatavaksi kaikkiin .NET-toteutuksiin [18; 19]. .NET Core tukee myös kaikkia .NET-ohjelmointikieliä [18].

ML.NET-sovelluskehys on ilmainen ja avoimen lähdekoodin koneoppimissovelluskehys, jossa voi rakentaa ja kehittää koneoppimisratkaisuja .NET-ympäristössä ja siten myös liittää niitä .NET-sovelluksiin [20]. ML.NET-sovelluskehys toimii käyttäen .NET Core -ympäristöä, ja siten se myös tukee samoja alustoja kuin .NET Core. Sitä voi myös käyttää .NET Framework -ympäristössä, joka toimii vain Windows-alustalla. [21.]

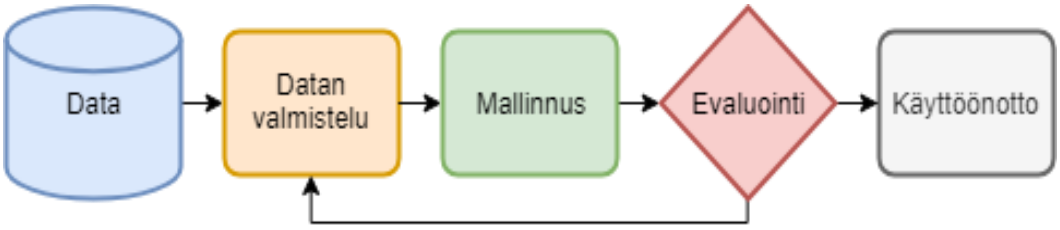
ML.NET-sovelluskehys tuo ensimmäiset .NET-ohjelmoitavat rajapinnat koneoppimismallien opettamiseen ja ennustuksien tekemiseen [21]. ML.NET-sovelluskehys tarjoaa useita oppimisalgoritmeja ohjattuun ja ohjaamattomaan oppimiseen sekä perinteisiin koneoppimisongelmiin, kuten esimerkiksi binääriluokitteluun, monen luokan luokitteluun, regressioon ja ryhmittelyyn. ML.NET-sovelluskehys tarjoaa myös sijoitus-, suosittelu- ja säännöttömyysongelmiin muutamia oppimisalgoritmeja, jotka perustuvat ohjattuun ja ohjaamattomaan oppimiseen. [22.]

Syväoppimisen ohjelmoitavia rajapintoja ML.NET-sovelluskehyksessä ei ole, mutta se tukee ulkopuolisia sovelluskehyksiä. Esimerkiksi ML.NET-sovelluskehys tukee Googlen TensorFlow- ja Microsoftin Cognitive Toolkit (CNTK) -kirjastoja [23]. ML.NET-sovelluskehys tukee myös avointa neuroverkon vaihtoformaattia (engl. open neural network exchange, ONNX). ONNX-formaatti antaa mahdollisuuden kehittää koneoppimismalleja eri sovelluskehyksissä ja siirtää koneoppimismalleja sovelluskehyksien välillä. Siirtäminen onnistuu vain, jos sovelluskehykset tukevat ONNX-formaattia. [24.]

Kun sovelluskehyksiä ja työkaluja etsittiin ja valittiin, ML.NET-sovelluskehys oli kaikista lupaavin, koska .NET-ympäristö oli entuudestaan tuttu. ML.NET-sovelluskehys antaa mahdollisuuden integroida helposti koneoppimisratkaisuja olemassa oleviin .NET-sovelluksiin, ja se myös tukee ONNX-formaattia. Tämä tarkoittaa, että koneoppimisratkaisuja ei ole sidottu pelkästään .NET-ympäristöön.

.NET-ohjelmointikielistä otettiin käyttöön C#, koska se oli entuudestaan tuttu ohjelmointikieli. Se madalsi kynnystä ML.NET-sovelluskehyksen käytössä, koska C#-ohjelmointikielessä on paljon valmiita työkaluja, jotka auttoivat datan käsittelyssä. Siihen myös löytyi eniten valmiita esimerkkejä ML.NET-sovelluskehyksen GitHub-sivuilta.

ML.NET-sovelluskehyksessä koneoppimismallien kehittäminen on korkean tason pääkomponentteilla suoraviivaista. Kuvassa 4 ovat tyypilliset koneoppimistyövaiheet vaiheittain, mitä kutsutaan *putkeksi* (engl. pipeline). Putki sisältää kaikki ne operaatiot, joita tarvitaan koneoppimismallin luomiseen [25]. Putken ensimmäinen vaihe on muokata ja valmistella tuotu data mallinnusta varten. Datasta kerätään halutut *ominaisuudet* (engl. features), joita halutaan mallintaa. Ominaisuuksilla tarkoitetaan havaintojen tai ilmiön mitattavia arvoja, joita ollaan sillä hetkellä mittaamassa [25].



Kuva 4. Tyypilliset koneoppimistyövaiheet vaiheittain [26].

Data yleensä pilkotaan kahteen osaan, koulutus- ja testidataan, minkä jälkeen koulutusdata syötetään oppimisalgoritmille. Oppimisalgoritmi tuottaa mallin, joka lopuksi evaluoidaan. Evaluoinnissa mallin laatua ja tehokkuutta testataan testidatalla. Evaluoinnin jälkeen malli voidaan ottaa käyttöön. Jos se ei ole tarpeeksi laadukas tai tehokas, palataan takaisin datan valmisteluun, jossa datasta kerätään uusia ominaisuuksia tai niitä muokataan. [26.]

Koneoppimiskonteksti

Koneoppimiskonteksti on aloituskomponentti ML.NET-sovelluskehyksessä, joka sisältää aloituspisteet kaikille toiminnoille ML.NET-sovelluskehyksessä. Koneoppimis-kontekstista luodaan aina aluksi instanssi, jota esimerkkikoodi 1 kuvastaa. Sitä tarvitaan arviointien, evaluointien ja ennustuksien tekemisessä. Se tarjoaa myös toimintoja poikkeuksien seuraamiseen ja lokittamiseen. Koneoppimiskontekstilla voidaan asettaa oppimiseen satunnaisuutta, joka annetaan parametrina esimerkkikoodissa 1 konstruktorille. [27.]

```
var mlContext = new MLContext(seed: 0);
```

Esimerkkikoodi 1. Ote C#-ohjelmasta, jossa luodaan koneoppimiskonteksti-instanssi. Koneoppimiskonteksti on aloituspiste kaikille toiminnoille ML.NET-sovelluskehyksessä [27].

Datanäkymä

Datanäkymä (engl. data view) on rajapinta, joka kuvastaa ML.NET-sovelluskehyksessä dataa [28]. Putken ensimmäisessä vaiheessa putkelle tuotu data on muokattava datanäkymäksi. Datanäkymärajapinta on yksi putken tärkeimmistä komponenteista, koska suurin osa ML.NET-sovelluskehyksen koodikannasta kuluttaa sitä [28].

Datanäkymät muistuttavat SQL-näkymiä [29], ja ne on suunniteltu toimimaan sulavasti ja tehokkaasti monen ulottuvuuden datan ja suuren datamäärän kanssa [28]. Datanäkymät ovat saaneet inspiraationsa tietokantamaailmasta, jossa tietokantataulut ovat muuttuvaa ja näkymät ovat muuttumatonta dataa. Datanäkymät ovat kyselyiden ja hakujen tulos tietokantatauluista ja -näkymistä. [28.]

Datanäkymät ovat kaavamaisia ja koottavia tauluja dataa. Kaavamaisuudella tarkoitetaan, että datanäkymät koostuvat sarakkeista, jotka sisältävät informaatiota. Jokaisella sarakkeella on nimi, indeksi ja datatyyppi. Koottavuudella tarkoitetaan, että datanäkymiin voidaan koota sarakkeita toisista näkymistä kopioimalla. Datanäkymät ovat muuttumattomia, koska datanäkymät ei itsessään sisällä arvoja, vaan informaatio lasketaan alkuperäisistä näkymistä tai tauluista, mikä tarkoittaa, että informaatiota ei voi datanäkymissä muuttaa. [28; 29.]

Datanäkymät ovat *kursorin* (engl. cursor) tuottama näkymä. Kursori on siis osoitin senhetkiseen riviin datassa. Kursorin tehtävä on iteroida dataa rivi kerrallaan ja näyttää datanäkymissä informaatiota.. Kursoreita voi olla useampia samanaikaisesti, ja kursorit ovat itsenäisiä, koska datanäkymät ovat muuttumattomia. Vain osaa datanäkymästä käytetään kerrallaan, ja siksi datanäkymät ovat *laiskoja* (engl. lazy). Laiskuudella tarkoitetaan, että käyttämättömiä rivejä ja sarakkeita ei tarvitse laskea. Datanäkymiin lasketaan vain ne rivit ja sarakkeet, joita kursorit käyttävät kullakin hetkellä. [28; 29.]

Datanäkymät ovat ML.NET-sovelluskehyksessä koneisto datan käsittelyssä ja isossa roolissa kaikissa koneoppimistyövaiheissa. Suurin osa ML.NET-sovelluskehyksen pääkomponenteista kuluttavat datanäkymärajapintaa, jossa esimerkiksi lataus-, tallennus-, opetus- ja evaluointikomponentit kuluttavat sitä. [28.]

Datalukija

Datalukija (engl. data reader) on ML.NET-sovelluskehyskomponentti, jolla luetaan dataa esimerkiksi tekstitiedostoista datanäkymiin. Datalukija on ”laiska”, kuten datanäkymät, ja sen pystyy määrittelemään etukäteen, ennen kuin mitään dataa luetaan. Esimerkkikoodissa 2 on määritelty datalukijakomponentti, jossa määritellään tekstitiedoston kaava. Kaavassa ovat sarakkeiden nimet, datatyypit ja järjestys [27].

Datalukijan ”Read”-metodia kutsuessa tekstitiedostoa ei lueta. Vasta kun kursoria pyydetään saatuun dataan, lukeminen tapahtuu. [29.]

```
var reader = mlContext.Data.CreateTextLoader(
      columns: new TextLoader.Column[]
      {
            new TextLoader.Column(“Label”, DataKind.TXT, 0),
            new TextLoader.Column(“Text”, DataKind.TXT, 1)
      },
      hasHeader: true
);

var dataView = reader.Read(dataPath);
```

Esimerkkikoodi 2. Ote C#-ohjelmasta, jossa luodaan datalukijan kaava. Datalukija lukee tekstitiedoston ”laiskasti” kaavan mukaisesti datanäkymään. [27.]

Dataa on myös mahdollista lukea datanäkymiin tyypitetyillä luokilla. Tällöin tyypitetyn luokan muuttujat määrittävät sarakkeiden nimet ja datatyypin. Esimerkkikoodissa 3 numeroitavasta datasta luodaan datanäkymä, jossa numeroitava data voidaan lukea esimerkiksi SQL-tietokannasta tai JSON- ja XML-tiedostoista .NET-ympäristön omilla työkaluilla. [27.]

```
var dataView = mlContext.Data.ReadFromEnumerable<Observation>(observations);
```

Esimerkkikoodi 3. Ote C#-ohjelmasta, jossa numeroituvasta datasta luodaan datanäkymä, jossa ”Observation” on tyypitetty luokka [27].

Muuntaja

Muuntaja (engl. transformer) on rajapinta, joka ML.NET-sovelluskehyksessä muuntaa dataa olomuodosta toiseen. Kaikki datanäkymien data täytyy muuntaa liukuluvuiksi, koska ML.NET-sovelluskehyksessä oppimisalgoritmit odottavat saavansa ominaisuudet liukulukuvektoreina [30]. Tätä toimenpidettä varten tarvitaan muuntajia, koska data voi olla esimerkiksi teksti-, merkkijono- tai kokonaislukumuodossa.

Muuntaja on komponentti, joka yleensä vastaanottaa datanäkymän sarakkeen datan ja tuottaa datanäkymään uuden sarakkeen, jossa on muunnettu data. Muuntajat pystyvät myös tuottamaan uuden sarakkeen vanhan tilalle. [29.] Vanha sarake jää niin sanotuksi piilotetuksi sarakkeeksi, koska datanäkymät ovat muuttumattomia [28]. Näitä molempia tapoja on kuvastettu esimerkkikoodissa 4, jossa ensimmäiseksi sarake ”Text” normalisoidaan ja sitten muutetaan sanasäkiksi sarakkeeseen ”BoW”.

Useita muuntajia on mahdollista ketjuttaa, jotta useampia muunnoksia voi suorittaa peräkkäin. Ne siten muodostavat uuden muuntajan, jossa ketjutetut muuntajat on yhdistetty. Esimerkkikoodissa 4 tätä on myös kuvastettu, kun kaksi toimenpidettä on ketjutettu toisiinsa. Tyypillisesti opetettu malli on yhdistelmä ketjutettuja muuntajia ja siksi itse malli on myös muuntaja. [29.]

```
var mlContext = new MLContext(seed: 0);
var transformer = mlContext.Transforms.Text.NormalizeText("Text")
      .Append(mlContext.Transforms.Text.ProduceWordBags("BoW", "Text"));
```

Esimerkkikoodi 4. Ote C#-ohjelmasta, jossa normalisoitu teksti muutetaan sanasäkiksi [29].

Kun muuntaja luodaan, mitään dataa ei vielä muunneta, vaan muuntajissa määritellään, miten dataa muunnetaan. Jokainen muuntaja tuottaa ”laiskasti” uuden datanäkymän, kun muuntajan ”Transform”-metodia kutsutaan. Metodi ottaa parametrikseen esimerkkikoodin 5 mukaisesti datanäkymän, jonka dataa muunnetaan muuntajan määrittelemällä tavalla. [29.]

```
IDataView transformedData = transformer.Transform(someData);
```

Esimerkkikoodi 5. Ote C#-ohjelmasta, jossa muuntaja tuottaa uuden datanäkymän toisesta datanäkymästä ML.NET-sovelluskehyksessä [29].

Muuntajat ovat datanäkymien ja datalukijan tapaisesti ”laiskoja”. Vaikka muuntajan ”Transform”-metodia kutsutaan, ei mitään muunnoksia vielä tapahdu. Vasta sitten kun uudesta datanäkymästä saadaan kursori ja sen arvoja ruvetaan käyttämään, muunnos tapahtuu niissä sarakkeissa, joita tarvitaan ja käytetään. Tämä johtuu siitä, että datanäkymissä ei välttämättä ole dataa saatavilla, kun muuntajan ”Transform”-metodia kutsutaan. [29.]

Arvioija

Arvioija (engl. estimator) on ML.NET-sovelluskehyksessä rajapinta ja komponentti, jonka tarkoitus on oppia datasta ja tuottaa muuntaja, jota kutsutaan malliksi. Oppimisalgoritmit ovat arvioijia ML.NET-sovelluskehyksessä. Arvioijat eivät ole ”laiskoja”, koska arvioijan ”Fit”-metodia kutsuessa oppiminen käynnistyy, mikä saattaa olla aikaa vievä toiminto riippuen arvioijasta. [29.]

Arvioijaa on esimerkkikoodissa 6 kuvastettu putkessa. Putken alussa ”Label”-sarakkeen arvot muunnetaan numeraalisiksi avaimiksi ja lopuksi ne palautetaan takaisin alkuperäiseksi ”PredictedLabel”-sarakkeessa. ML.NET-sovelluskehys tarjoaa korkean tason ”FeaturizeText”-metodin, joka normalisoi tekstin. Normalisointi poistaa tekstistä välimerkit ja numerot sekä muuttaa kaikki isot kirjaimet pieniksi kirjaimiksi [30]. Se poistaa vakiona englanninkieliset pysäytyssanat ja tekee *n-grammeja* peräkkäisistä sanoista, millä tarkoitetaan peräkkäisten sanojen muuntamista numeraalisiksi arvoiksi [25]. Lopuksi se pisteyttää n-grammit vakiona termitiheydellä [30]. ”FeaturizeText”-metodia käytetään ”Text”-sarakkeeseen.

Koska kyseessä on monen luokan luokittelu, putkeen on määritetty monen luokan luokittelija Naive Bayes -algoritmi, joka on tämän putken arvioija. Arvioijalle on määritetty ”Label”- ja ”Features”-sarakkeet. ”Label”-sarake sisältää luokittelun oikeat vastaukset ja ”Features”-sarake datan ominaisuudet. Seuraavaksi opetettava data haetaan ja muutetaan datanäkymäksi. Datanäkymä jaetaan kahteen satunnaiseen osaan, joiden suhdeluku määritetään parametrina ”TrainTestSplit”-metodissa. Arvioija opetetaan koulutusdatalla, joka tuottaa mallin. Malli muuntaa testidatan evaluoitavaksi, ja tämä tuottaa mallin metriikkaa. Metriikka kuvaa mallin tehokkuutta testidataan. [29.]

```
var mlContext = new MLContext(seed: 0);
var pipeline = mlContext.Transforms.Conversion.MapValueToKey(“Label”)
      .Append(mlContext.Transforms.Text.FeaturizeText(“Features”, “Text”))
      .Append(mlContext.MulticlassClassification.Trainers.NaiveBayes(
            “Label”,
            “Features”))
      .Append(mlContext.Transforms.Conversion.MapKeyToValue(“PredictedLabel”));

var data = GetData(dataPath);
var dataView = mlContext.Data.ReadFromEnumerable(data);

var (trainData, testData) = mlContext.MulticlassClassification.TrainTestSplit(
      dataView,
      testFraction: 0.2);

var model = pipeline.Fit(trainData);
var predictions = model.Transform(testData);
var metrics = mlContext.MulticlassClassification.Evaluate(predictions);
```

Esimerkkikoodi 6. Ote C#-ohjelmasta, jossa putkeen määritellään datan muunnos ja monen luokan luokitteluarvioija. Arvioija opetetaan koulutusdatalla ja malli evaluoidaan testidatalla. [29.]

Näillä pääkomponenteilla pystyy tekemään kaikki putken työvaiheet ML.NET-sovelluskehyksessä. Esimerkkikoodissa 6 putki suoritetaan lähes kokonaan, ja tämä onnistuu suhteellisen vähällä koodilla. Nämä esimerkit käyttivät ML.NET-

sovelluskehyksen dynaamista rajapintaa, mutta sille on olemassa myös vahvasti tyypitetty staattinen rajapinta, jota ei käytetty tässä insinöörityössä [27].

3.2 Visual Studio

Visual Studio on Microsoftin ohjelmointiympäristö (engl. integrated development environment, IDE), jossa on standardin IDE:n ominaisuuksien lisäksi paljon työkaluja. Visual studio IDE tukee kaikkia .NET-ohjelmointikieliä, mutta myös C++-, Python- ja JavaScript-ohjelmointikieliä. Visual Studio IDE:tä on mahdollista käyttää Windowsilla tai Mac-käyttöjärjestelmällä. [31.]

Suosituimpia ominaisuuksia Visual Studio IDE:ssä ovat ne ominaisuudet, jotka helpottavat ja auttavat kehitystyötä, kuten esimerkiksi koodin muotoilutyökalut, jolla koodista tehdään helpommin ylläpidettävä ja luettavampaa ilman, että sen toiminta muuttuu. IntelliSense on toinen Visual Studio IDE:n ominaisuus, joka näyttää informaatiota koodista kirjoittaessa tekstieditorissa. Se osaa ehdottaa ja näyttää metodeja ja muuttujia, joita on käytettävissä. [31.]

NuGet

NuGet on Microsoftin tukema .NET-paketinhallintatyökalu, jolla voi jakaa valmiiksi käännettyä koodia kehittäjille. Kehittäjät voivat tehdä omia paketteja ja jakaa niitä joko julkisiin tai yksityisiin palvelimiin. Kuluttajat voivat sitten vetää itselleen .NET-sovelluksiin palvelimilta NuGet-paketteja omaan käyttöön ja siten kuluttaa jaettua koodia. Koska NuGet-paketteja on mahdollista jakaa yksityisiin palvelimiin, niitä on mahdollista jakaa vain työryhmän tai organisaation välillä. [32.]

Visual Studio IDE:een on sisäänrakennettu NuGet-paketinhallintatyökalu, jolla voi helposti ladata, poistaa ja päivittää NuGet-paketteja [33]. Visual Studio IDE:llä voi myös julkaista NuGet-paketteja, jolloin on syytä kohdentaa sovellus .NET Standardiin, jotta se toimii kaikissa .NET-sovelluksissa [32]. NuGet-paketteja voi myös hallita komentorivityökalujen kautta [32].

Visual Studio IDE oli ykkösvalinta ohjelmointiympäristöksi, koska ML.NET-sovelluskehys on jakelussa NuGet-pakettina, joka on helposti saatavilla Visual Studio IDE:n sisäänrakennetussa NuGet-paketinhallintatyökalusta. Visual studio IDE sisältää myös kaiken tarvittavan insinöörityössä tehtyjen koneoppimisratkaisujen tekemisessä, kuten .NET-ympäristön ja C#-ohjelmointikielen, jotka oli kehitystyöhön valittu.

4 Koneoppimisratkaisujen kehitystyö

Tässä luvussa esitellään insinöörityössä kehitettyjä koneoppimisratkaisuja ja niiden tuloksia. Aluksi kuitenkin selostetaan luvussa 4.1 kehitystyön ja koneoppimisratkaisujen tarkoitusta ja tavoitteita. Luvussa 4.2 esitellään, miten ML.NET-mallien kehitystyökalu tehtiin. Lopuissa luvuissa selostetaan niitä koneoppimisratkaisuja, joita kehitystyössä tehtiin. Näihin kuuluu luvussa 4.3 selostettu dokumenttien luokittelu, luvussa 4.4 selostettu kielten luokittelu ja luvussa 4.5 selostettu dokumenttien ryhmittely. Jokaisesta tehdystä koneoppimisratkaisusta esitellään niiden toteutusta esimerkeillä ja mittaustuloksia sekä selostetaan niihin liittyviä ongelmia.

4.1 Kehitystyön tarkoitus ja tavoitteet

Kehitystyön yhtenä tarkoituksena oli tutkia, mitä koneoppimisella on mahdollista tehdä ja mitä koneoppimistyökaluja on käytettävissä, jotta tiedettiin, mitä koneoppimisratkaisuja lähdettiin kehittämään. Koneoppimistyökaluksi valittiin ML.NET-sovelluskehys, jonka perusteella kehitettiin käytännön koneoppimisratkaisuja ja tarkasteltiin niiden laatua ja käytettävyyttä oikeisiin tuotantotason sovelluksiin. Koneoppimisratkaisujen opettamiseen käytettiin projektin teettäneen yrityksen dataa, jolla määritettiin mahdollisia opetettavia ongelmia. Käytettävässä datassa oli neljä eri datasettiä, jotka ovat alakohtaisia ja toistensa poissulkevia.

Yksi kehitystyön tavoite oli kehittää ML.NET-sovelluskehyksellä työkalu, jolla pystyi kehittämään, kouluttamaan ja testaamaan koneoppimismalleja helposti ja nopeasti. Koneoppimisratkaisuista kehitystyössä otettiin tavoitteeksi luokitella dokumentteja. Dokumenttien luokittelussa tavoitteena oli löytää paras mahdollinen malli, jolla pystyttäisiin automatisoimaan dokumenttien luokittelu. Tämä tarkoittaa, että mallin täytyy

oppia luokittelemaan dokumentteja lähes aina oikein käytetyllä koulutusdatalla. Dokumenttien luokitteluun keskityttiin eniten, koska siitä oli helpointa aloittaa.

Kehitystyön toiseksi tavoitteeksi asetettiin kehittää koneoppimisratkaisuja, jotka auttaisivat käyttäjiä löytämään samankaltaisia ja kiinnostavia dokumentteja. Samankaltaisten dokumenttien löytämiseen käytettiin ohjaamattoman oppimisen tekniikoita. Tavoitteena oli luoda ryhmittelymalli, jonka avulla pystyttäisiin tutkimaan olemassa olevista dokumenteista samankaltaisuuksia.

4.2 ML.NET-mallien kehitystyökalu

Ensimmäinen askel ML.NET-mallien kehitystyökalussa oli aloittaa luomalla uusi konsolisovellus Visual Studiossa. Konsolisovellukseen asennettiin Visual Studion sisään rakennetun NuGet-paketinhallintatyökalun kautta seuraavat NuGet-paketit:

- Microsoft.ML 0.10.0
- OxyPlot.Core 1.0.0.

NuGet-paketit sisältävät ML.NET-sovelluskehyksen ja kaiken tarvittavan kehitystyössä tehtyjen koneoppimisratkaisujen tekemiseen. Paketit sisältävät OxyPlot-kirjaston datan visualisointia varten. NuGet-paketit eivät sisällä syväoppimisen kirjastoja, koska kehitystyössä ei niitä käytetty, mutta nekin ovat saatavilla NuGet-paketteina.

Koska kehitystyö oli kokeiluprojekti ja ML.NET-sovelluskehyksestä ei ollut aikaisempaa kokemusta, kehitystyön alussa lähdettiin kehittämään yksinkertaista dokumenttien luokittelua esimerkkien avulla. Kun ML.NET-sovelluskehyksestä sai näppituntumaa, putken yleinen toiminnallisuus pystyttiin muuttamaan liitteen 1 mukaiseen putkirajapintaan, mikä oli mahdollista vain sen takia, että data esitetään datanäkymässä, joka voi sisältää mitä tahansa dataa [28]. Liitteen 1 putkirajapinta hoitaa ja määrittelee työkalussa ML.NET-mallien opettamisen, evaluoinnin ja tallentamisen.

Koneoppimisratkaisuissa niitä putkia, jotka kuluttavat liitteen 1 putkirajapintaa, kutsutaan työkalussa dataprosesseiksi, koska ne määrittävät, miten dataa käsitellään, muunnetaan ja arvioidaan. Dataprosessit siis päättävät, miten data ladataan ja muunnetaan opettamista varten. Jokainen dataprosessi määrittää myös sen, mitä ongelmaa ne

ratkovat. Putkirajapinta ja koneoppimiskonteksti jaettiin jokaiselle dataprosessille parametrina niiden konstruktoreissa. Työkaluun tarvittiin vielä keino, jolla voitiin määrittää asetuksia dataprosesseille. Tähän käytettiin hyväksi konsoliparametreja. Konsoliparametreja tehtiin kehitystyön edetessä lisää, mutta taulukossa 1 näytetään kaikki kehitystyön aikana tehdyt asetukset.

Taulukko 1. ML.NET-mallien kehitystyökalussa käytetyt konsoliparametrit.

Parametri	**Selite**
Target	Suoritettavan dataprosessin nimi
Datapath	Polku datakansioon
TrainDataFiles	Datakansion koulutusdatatiedostot
TestDataFiles	Datakansion testidatatiedostot
PredictionDataFiles	Datakansion ennustusdatatiedostot
SavePath	Opetetun mallin tallennuspolku
TestFraction	Liukuluku, jolla jaetaan koulutusdata kahteen osaan, jos testidatatiedostoja ei määritetä
MinTrainingData	Luokittelussa minimiraja positiivisia datapisteitä opetettavalle luokalle
MinTrainingRatio	Luokittelussa käytetty liukuluku, jolla rajoitetaan positiivisten ja negatiivisten datapisteiden epätasapainoa
Prune	Luokkakohtaisessa binääriluokittelussa käytettävän luokkapuun datan karsinta aliluokille
OmitRoot	Luokkakohtaisessa binääriluokittelussa juuritason pois jättäminen luokkapuusta
Depth	Luokkakohtaisessa binääriluokittelussa luokkapuun opetussyvyys
KClusters	Ryhmittelyssä käytettävä ryhmien määrä
Trainer	Käytettävän oppimisalgoritmin nimi

Työkalulla pystyi tähän asti pelkästään opettamaan putkia. Työkaluun tehtiin lisäksi pelkkien päätelmien tekemiseen komponentti, jota kutsutaan päätelmämoottoriksi työkalussa. Komponentti pystyy lataamaan opetetun mallin kiintolevyltä ja tehdä ennustuksia dataan ladatulla mallilla. Päätelmämoottoria pystyttäisiin käyttämään työkalun ulkopuolella muissa .NET-sovelluksissa, ja sitä käytetäänkin myös työkalussa dataprosessien osana.

4.3 Dokumenttien luokittelu

Dokumenttien luokittelu oli kehitystyössä ensimmäinen dataprosessi, jota lähdettiin toteuttamaan. Aluksi päätettiin, mitä ominaisuuksia dokumenttien luokittelussa haluttiin käyttää. Dokumenteille luotiin alustava putki, jossa dokumenttien kuvausta, avainsanoja ja otsikkoa käytettiin ominaisuuksina. Datassa jokaisella dokumentilla oli luokka, jota käytettiin tuloksena putkessa.

Opetuksessa käytettävä data täytyi tuoda tietokannasta johonkin formaattiin. Kokeilemalla parhaaksi tiedostomuodoksi havaittiin JSON, koska JSON-tiedostomuotoon oli tietokannasta helpointa tuoda dataa. Data tuotiin aluksi tekstitiedostoon, mutta datalukijakomponentti ei onnistunut lukemaan dokumenttien rivivaihtoja yhdeksi kuvaukseksi. JSON-tiedostomuodossa tässä ei ollut ongelmaa, ja siksi sitä käytettiin myös jatkossa kaikissa dataprosesseissa.

Dokumentteja luokiteltiin aluksi monen luokan luokitteluna ja datasetillä 1, joka sisälsi noin 600 dokumenttia. Datasettiä 1 tutkimalla havaittiin, että data sisälsi sekaisin suomen- ja englanninkielisiä dokumentteja, ja osasta datasta puuttui avainsanat. Aluksi ominaisuuksiksi valittiin vain dokumenttien otsikko ja kuvaus.

Esimerkkikoodissa 7 on dokumenttien luokitteluun käytetty putki, jossa aluksi muutetaan ”Classification”-sarakkeen arvot numeraalisiksi avaimiksi ”Label”-sarakkeeseen. ”Description”- ja ”Title”-sarakkeet muutetaan ominaisuuksiksi ”FeaturizeText”-metodilla, ja ne yhdistetään ”Features”-sarakkeeseen. Tämän jälkeen putkeen lisätään arvioija, joka saadaan ”GetMulticlassClassificationTrainer”-metodista. Metodi palauttaa konsoliparametreissa määritetyn nimen perusteella vastaavan arvioijan, jolle on vakiona määritetty, että ”Label”-sarakkeessa on tulokset ja ”Features”-sarakkeessa on ominaisuudet.

```
var dataProcessPipeline = _mlContext.Transforms.Conversion.MapValueToKey(
          “Label”, “Classification”)
     .Append(_mlContext.Transforms.Text.FeaturizeText(“Features”,
          new [] {“Description”, “Title”)})
     .Append(GetMulticlassClassificationTrainer(settings.Trainer))
     .Append(_mlContext.Transforms.Conversion.MapKeyToValue(
          “PredictedLabel”));
```

Esimerkkikoodi 7. Ote tehdystä C#-ohjelmasta, jossa määritetään dataprosessin putki [30].

Kun dataprosessi oli määritetty, voitiin opetusdata ladata JSON-tiedostosta ja siten muuttaa se datanäkymäksi. Ladattu datanäkymä jaettiin satunnaisesti koulutus- ja testidataksi, joita esitetään esimerkkikoodissa 8. Satunnaisen koulutus- ja testidatan jaettavaa osuutta voidaan säätää konsoliparametrilla, joka on prosenttiosuus kokonaisdatasta. Tätä prosenttiosuutta pidettiin lähes aina 20 %:ssa.

```
var data = JSONUtility.DeserializeFromJsonFiles<Document>(
     settings.DataPath,
     settings.TrainDataFiles);
var dataView = _mlContext.Data.ReadFromEnumerable(data);
var (trainData,testData) = _mlContext.MulticlassClassification.TrainTestSplit(
     dataView,
     testFraction: settings.TestFraction);
```

Esimerkkikoodi 8. Ote tehdystä C#-ohjelmasta, jossa luetaan dokumenttidataa JSON-tiedostosta. Ladattu data muutetaan datanäkymäksi, joka jaetaan konsoliparametrin osuudella koulutus- ja testidataksi.

Mallin opetus tapahtuu yksinkertaisesti hyödyntämällä liitteen 1 putkirajapintaa. Esimerkkikoodissa 9 kulutetaan putkirajapintaa, jossa ensin opetetaan malli koulutusdatalla ja sitten malli evaluoidaan testidatalla. Lopuksi malli tallennetaan zip-tiedostoksi haluttuun polkuun.

```
_pipeline.AddDataProcessPipeline(dataProcessPipeline);
_pipeline.Train(trainData);
var metrics = _pipeline.EvaluateMulticlassClassificationModel(testData);
_pipeline.Save(settings.SavePath);
```

Esimerkkikoodi 9. Ote tehdystä C#-ohjelmasta, jossa opetetaan, evaluoidaan ja tallennetaan malli kuluttamalla työkalussa tehtyä putkirajapintaa.

Datasettiin 1 kokeiltiin ottaa ominaisuudeksi avainsanatkin. Niihin dokumentteihin, joilla ei ollut avainsanoja, määritettiin tyhjät merkkijonot avainsanoiksi. Esimerkkikoodin 7 putkea muutettiin siten, että avainsanat eroteltiin toisistaan symbolisoimalla ja sitten avainsanat muutettiin ominaisuuksiksi ”FeaturizeText”-metodilla. Lopuksi avainsanat lisättiin kuvaus- ja otsikkosarakkeiden tapaisesti ominaisuudet-sarakkeeseen.

Datasettiä 1 kokeiltiin vielä esikäsitellä tarkemmin. Datasetissä 1 oli sekaisin suomen- ja englanninkielisiä dokumentteja. Suomenkielisiä dokumentteja oli selvästi vähemmän, ja siksi ne poistettiin datasetistä. Suomenkieliset dokumentit poistettiin hyödyntämällä suomen ja englannin kielten luokittelijaa, jota selostetaan luvussa 4.4. Ajatuksena oli, että erikieliset dokumentit saattaisivat vaikuttaa mallin laatuun.

Jokaisen datasetin 1 eri putkea evaluoitiin. Evaluointi tuottaa mallista metriikoita, joista mallin *tarkkuutta* (engl. accuracy) haluttiin tutkia erityisesti. Tarkkuudella tarkoitetaan luokittelussa mallin ennustamien oikeiden tulosten määrää jaettuna koko testidatan määrällä. Tämä saa arvoja väliltä nolla ja yksi, ja arvo nolla on kaikista epätarkin ja arvo yksi on kaikista tarkin [25]. Tarkkuus voidaan laskea yhtälöllä 1.

$$Accuracy = \frac{\mathrm{TP+TN}}{\mathrm{N+P}} \quad (1)$$

TP on oikeiden positiivisten ennustuksien määrä.
TN on oikeiden negatiivisten ennustuksien määrä.
N on negatiivisten datapisteiden määrä.
P on positiivisten datapisteiden määrä. [34.]

Taulukosta 2 nähdään, että tarkkuus on parhaimmillaan datasetin 1 kanssa avainsanoilla. Se on silti mallille liian alhainen tulos, koska joka toinen dokumentti ennustetaan väärin. Toisaalta datasetissä 1 on suhteellisen vähän dataa, mikä on todennäköisin syy mallin huonoihin tarkkuuksiin. Kaikista huonoin tulos saatiin, kun datasetistä 1 poistettiin suomenkieliset dokumentit. Syyksi voisi ajatella, että erikieliset tekstit ovat selkeästi erilaisia ja ne saattoivat kuulua myös eri luokkiin, mikä saattoi auttaa dokumenttien luokittelussa.

Taulukko 2. Datasetin 1 tarkkuudet eri ominaisuuksilla.

Datasetin 1 ominaisuudet	**Tarkkuus**
Otsikko ja kuvaus	0,455
Otsikko, kuvaus ja avainsanat	0,546
Otsikko, kuvaus ja avainsanat ilman suomenkielisiä dokumentteja	0,346

Dokumenttien monen luokan luokitteluun ei tämän jälkeen enää dataprosessissa keskitytty, koska projektin teettäneen yrityksen muissa dataseteissä luokkia on mahdollista olla useita samanaikaisesti. Usean mahdollisen luokan luokittelua kutsutaan siis monen tuloksen luokitteluksi, jota ei kehitystyötä tehtäessä ollut mahdollista ML.NET-sovelluskehyksessä tehdä. Ongelmaan keksittiin ratkaisuksi käyttää binääriluokittelua jokaista luokkaa kohden, mutta tällöin jokaista luokkaa kohden luodaan oma malli. Binääriluokittelu oli kuitenkin kompromissi, jolla monen tuloksen luokittelua pystyi opettamaan.

Alkuperäistä monen luokan luokitteludataprosessia ei tarvinnut paljoa muokata. Putken ei tarvinnut enää muuntaa luokittelun nimeä numeraaliseksi, koska binääriluokittelussa tulos sarakkeen arvot ovat boolean-tyyppisiä. Tämä kuitenkin vaati sen, että jokaista datasettiä kohden täytyi myös tietää kaikki mahdolliset luokat, joihin dokumentit voivat kuulua. Dataprosessi käytti samoja ominaisuuksia kuin aikaisemminkin, eli dokumentin otsikkoa, kuvausta ja avainsanoja. Dataprosessissa käytettiin datasettiä 2, jossa kaikki dokumentit ovat englanninkielisiä ja sisältää noin 3 800 dokumenttia.

Opetus tehdään siis siten, että mallit opetetaan silmukassa, jossa datasetin 2 dokumentit täytyy jokaisessa silmukan kierroksessa päivittää tulossarakkeen arvot uudestaan. Tämä tarkoittaa, että jos opetetaan luokkaa ”x”, niihin kaikkiin dokumentteihin merkitään tulokseksi epätosi, jotka eivät sisällä luokkaa ”x”. Niihin dokumentteihin merkitään tulokseksi tosi, jotka sisältävät luokan ”x”.

Tällä tavoin jokainen luokka pystyttiin opettamaan yhdellä kerralla, mutta se osoittautui nopeasti käyttökelvottomaksi, koska datasetissä 2 oli yli 80 mahdollista luokkaa ja useisiin luokkiin kuului todella vähän dokumentteja. Mahdolliset luokat olivat puurakenteessa, ja suurin osa luokista kuuluivat jonkin pääluokan alle. Luokkapuuta varten luotiin JSON-tiedosto, joka sisälsi kaikki datasetin 2 luokat sellaisessa rakenteessa, kuin ne olivat oikeasti tietokannassa. Opetettaessa otettiinkin huomioon vain pääluokat. Jos jonkin dokumentin luokka kuului opetettavan pääluokan aliluokkiin, sen tulokseksi merkittiin tosi. Jos dokumentin luokka ei kuulunut mihinkään opetettavan pääluokan aliluokkaan, sen tulokseksi merkittiin epätosi.

Datasetin 2 opetettavat luokat kutistuivat alle 15 pääluokkaan, ja ne tuottivat hyviä tarkkuustuloksia. Dataprosessiin käytettiin vielä konsoliparametria, jolla voidaan asettaa minimiraja positiivisiin datapisteisiin, ja rajan voi asettaa myös prosentuaaliseksi. Sillä hetkellä opetettavaa pääluokkaa ja sen aliluokkia täytyi löytyä datasta yli minimirajan verran, jotta pääluokka opetetaan. Tällä estetään opettamasta epätasapainossa olevia pääluokkia, koska sellaisia pääluokkia ei todennäköisesti pystytä opettamaan.

Pääluokan luokittelua testattiin erilaisilla algoritmeilla, ja niiden keskiarvotarkkuuksia näytetään taulukossa 3. Kaikilla binääriluokittelun algoritmeilla päästiin yli 0,80:n keskiarvotarkkuuden yläpuolelle, mikä on jo kiitettävä tulos malleille. Tämä tarkoittaa, että mallit tekevät keskiarviolta virheen joka viidennessä dokumentissa. Kaikissa

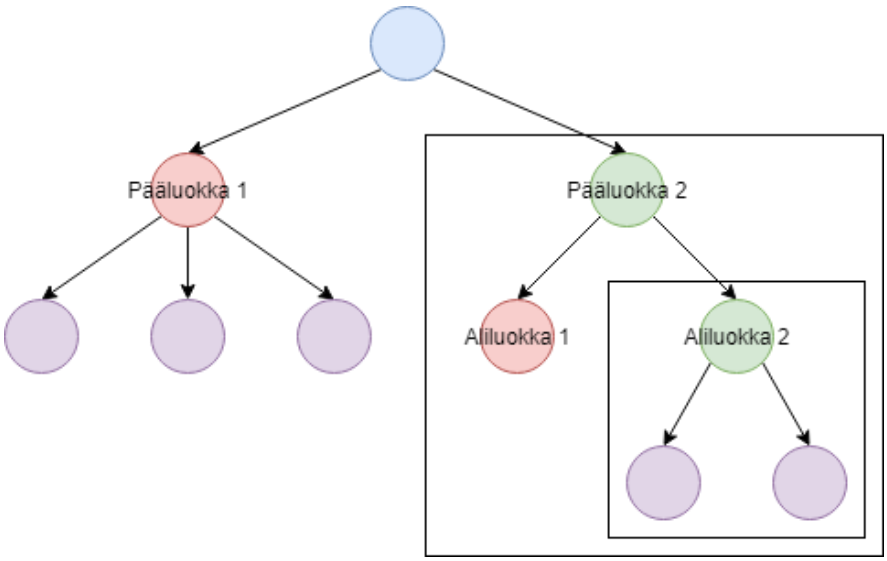
testeissä pidettiin minimirajana 100 dokumenttia. Minimiraja estää siis pääluokkien opettamisen, joihin kuului alle 100 dokumenttia.

Taulukko 3. Binääriluokittelun keskiarvotarkkuus datasetin 2 pääluokille. Luokitteluputkessa käytettiin eri algoritmeja.

Algoritmi	**Keskiarvotarkkuus**	**Minimiraja**
Fast Tree (FT)	0,839	100 dokumenttia
Fast Forest (FF)	0,826	100 dokumenttia
Linear Support Vector Machines (LSVM)	0,813	100 dokumenttia
Stochastic Dual Coordinate Ascent (SDCA)	0,839	100 dokumenttia
Averaged Perceptron (AP)	0,84	100 dokumenttia
Stochastic Gradient Descent (SGD)	0,839	100 dokumenttia

Binääriluokittelun dataprosessissa oli vielä yksi ongelma. Yrityksen dataseteissä oli myös mahdollista, että jokainen dokumentti kuuluu yhteen ja ainoaan pääluokkaan ja pääluokkien aliluokilla pystyi olemaan omia aliluokkia. Ongelmaa varten käytettiin rekursiota. Luokkapuu opetetaan rekursiivisesti, niin että luokkapuun haarat opetetaan kerrallaan. Rekursiota varten tehtiin muutamia konsoliparametreja, joilla pystyi vaikuttamaan luokkapuun opetukseen. Esimerkiksi opettamisen syvyyttä voitiin säätää, juuritaso voitiin jättää opettamatta ja dataa voitiin karsia luokkapuun haarakohtaisesti.

Karsiminen tehdään siten, että kun opetetaan kuvan 5 pääluokkaa 2, kaikki pääluokan 2 aliluokkiin kuuluvat dokumentit merkitään positiivisiksi datapisteiksi ja kaikki pääluokan 1 aliluokkiin kuuluvat dokumentit merkitään negatiivisiksi datapisteiksi. Kun opetetaan kuvan 5 aliluokkaa 2, positiivisiksi datapisteiksi valitaan kaikki aliluokan 2 aliluokkiin kuuluvat dokumentit ja negatiivisiksi aliluokan 1 aliluokkiin kuuluvat datapisteet, mutta pääluokan 1 aliluokkiin kuuluvat dokumentit jätetään kokonaan pois. Tämän tarkoitus oli, etteivät negatiiviset ja positiiviset datapisteet olisi suuressa epätasapainossa, eli toisiin päähaaroihin kuuluvaa dataa ei sisällytetä opetettavan haaran sisäisten luokkien opettamiseen.



Kuva 5. Opetettava luokkapuurakenne, joka havainnollistaa luokkapuun karsintaa rekursiivisesti eri syvyyksissä taso kerrallaan.

Karsimalla luokkapuun oksia pystytään myös luokittelemaan samanaiheisia dokumentteja eri luokkiin. Kuvitellaan, että kuvan 5 pääluokan 2 luokka olisi nimeltään esimerkiksi urheilu ja sen aliluokka 1 nimeltään jalkapallo ja aliluokka 2 nimeltään jääkiekko. Karsimalla luokkapuusta muunaiheiset haarat pois koulutusdatasta, binääriluokittelija pakotetaan oppimaan erilaisuudet jääkiekosta ja jalkapallosta, vaikka molemmat aiheet kuuluvat urheiluun.

Dataa karsimalla tuotetaan myös omia ongelmia, koska mitä syvempi luokkapuurakenne on, sitä vähemmän dataa on käytössä opetettaville syville aliluokille. Tähän ei myöskään enää pelkkä tarkkuusmetriikka riitä kertomaan mallin tehokkuudesta, koska se ei kerro mallin koko totuutta. Tarkkuus kertoi siis, kuinka monta ennustusta malli ennusti oikein suhteessa koko testidataan, mutta se ei kerro, kuinka hyvin se ennusti positiivisia datapisteitä oikein. Tämä tarkoittaa, että malli voi ennustaa lähes kaikki ennustukset oikein ja saa siten korkean tarkkuusarvon, mutta koulutusdata saattaa olla epätasapainossa ja siten sisältää vain muutamia positiivisia datapisteitä, jolloin malli saattaa ennustaa väärin vain testidatan positiivisiin datapisteisiin.

Tätä ongelmaa varten mallin metriikoita tutkittiin tarkemmin. Tarkkuuden lisäksi tutkittiin mallin *muistikykyä* (engl. recall). Muistikyvyllä tarkoitetaan luokittelussa oikein menneiden ennustuksien määrää jaettuna sillä kokonaismäärällä, joka oli testidatassa oikeasti oikein [25]. Muistikyky voidaan laskea yhtälöllä 2.

$$Recall = \frac{TP}{TP+FN} \quad (2)$$

TP on oikeiden positiivisten ennustuksien määrä.
FN on väärien negatiivisten ennustuksien määrä. [34.]

Muistikyvyn lisäksi voidaan laskea mallin *täsmällisyys* (engl. precision). Täsmällisyydellä tarkoitetaan luokittelussa luokkaan tehtyjen oikeiden ennustuksien määrää jaettuna luokkaan tehtyjen ennustuksien kokonaismäärällä [25]. Mallista nähdään siis, kuinka monta oikeaa positiivisista ennustusta malli tekee verrattuna mallin tekemiin vääriin positiivisiin ennustuksiin. Täsmällisyyttä voidaan laskea kaavalla 3.

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

TP on oikeiden positiivisten ennustuksien määrä.
FP on väärien positiivisten ennustuksien määrä. [34.]

Lopuksi muistikyvyn ja täsmällisyyden avulla voidaan laskea mallille *F1-pisteytys* (engl. F1-score). Se on luokittelussa muistikyvyn ja täsmällisyyden tasapainotettu keskiarvo [25]. F1-pisteytyksellä voidaan tutkia, kuinka hyvin malli sai pelkät positiiviset tai negatiiviset datapisteet ennustettua oikein. F1-pisteytys lasketaan kaavalla 4.

$$F1Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (4)$$

Precision on täsmällisyys.
Recall on muistikyky. [34.]

Binääriluokittelun dataprosessia kokeiltiin neljään eri datasettiin, ja näihin datasetteihin tehtiin testimittauksia, joissa käytettiin seuraavanlaisia asetuksia:

- Koulutus- ja testidatan jako on 20 %.
- Minimisuhdeluku on 15 %.
- Opetussyvyys on luokkapuun maksimisyvyys.

Datasetteihin 2 ja 3 kokeiltiin myös datan karsintaasetusta, jota ei voinut käyttää datasetteihin 1 ja 4, koska näissä dataseteissä pääluokilla ei ollut ollenkaan aliluokkia. Juuritaso jätettiin opettamatta niissä dataseteissä, joissa kaikki dokumentit kuuluivat

yhden juuren alle. Kaikkiin datasetteihin testattiin seitsemää eri algoritmia, joista mitattiin keskiarvometriikat.

Taulukossa 4 on testimittausten tulokset, ja siitä nähdään opetettujen luokkien määrä ja se, onko dataa karsittu datasetissä. Taulukosta nähdään myös jokaisen mittauksen keskiarvotarkkuus ja F1-pisteet. Jokaisessa mittaustuloksessa keskiarvotarkkuus on 0,80:n tienoilla, minkä perusteella voitaisiin ajatella, että tulokset näyttävät hyviltä. F1-pisteet toisaalta näyttävät, että mallit eivät ole saaneet kovin hyvin positiivisia datapisteitä ennustettua oikein.

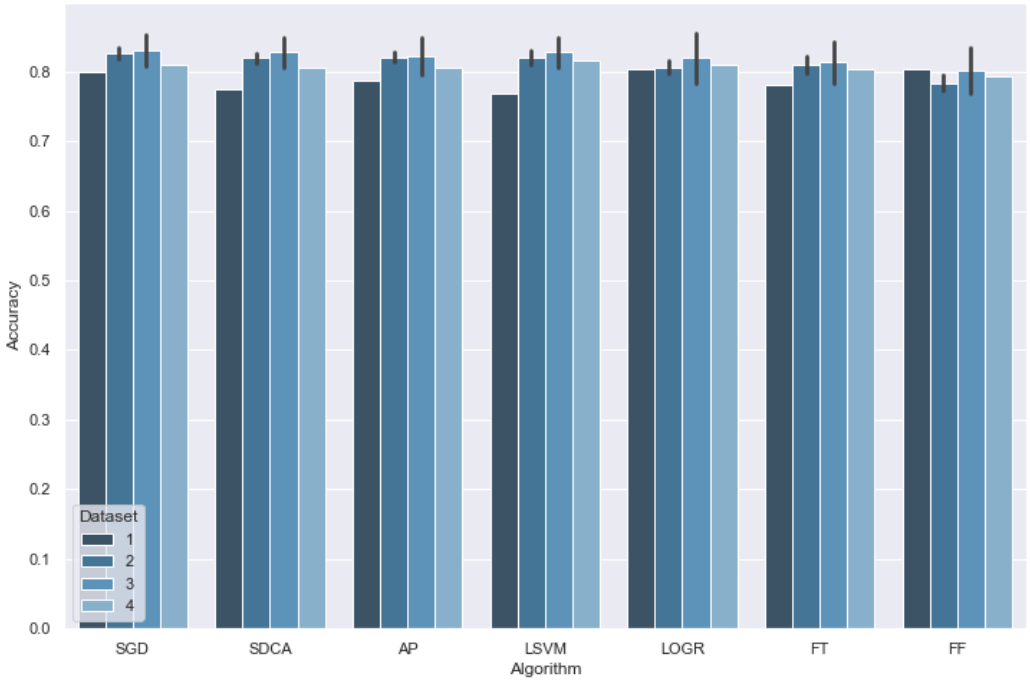
Parhaiten mittaustuloksissa olivat pärjänneet datasetit 2 ja 3, joilla on korkeimmat tarkkuudet ja F1-pisteet. Taulukosta 4 huomataan myös, että kun datasettien 2 ja 3 data oli karsittu, saatiin enemmän luokkia opetettua. Tämä johtui siitä, että opetustilanteeseen oli asetettu 15 % minimisuhdeluku, joka rajoittaa positiivisten ja negatiivisten datapisteiden epätasapainoa. Kun datasettien dataa ei ollut karsittu, positiivisia datapisteitä ei ollut suhteessa tarpeeksi verrattuna negatiivisiin datapisteisiin.

Taulukko 4. Testin mittaustulokset neljään eri datasettiin, joissa on eri määrä dokumentteja ja luokkia. Mittaustulokset ovat keskiarvo seitsemän eri algoritmin tuottamista metriikoista.

Datasetti	Karsittu	Luokkien määrä	Tarkkuus	F1-pisteet
1	Ei	3	0,789	0,485
2	Ei	6	0,822	0,694
2	Kyllä	12	0,80	0,723
3	Ei	3	0,793	0,662
3	Kyllä	13	0,848	0,670
4	Ei	5	0,807	0,565

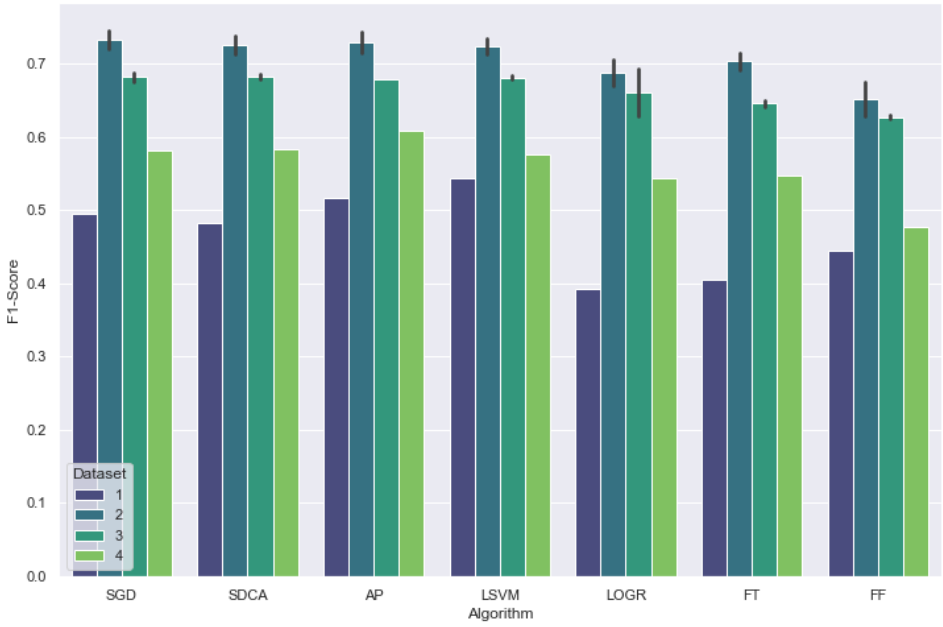
Koska luokkapuun opetetuista malleista täytyi kerätä jotenkin kaikki metriikat, puurakenteen opetuksen jälkeen tulostettiin JSON-tiedosto. Se sisälsi saman puurakenteen kuin opetettaessa, ja jokaisessa puun luokassa oli tallennettu luokan metriikat. Puurakenteeseen ei kuitenkaan tallennettu sellaisia luokkia, joita ei opetettu lainkaan. Metriikoitten lisäksi tallennettiin jokaisen opetetun luokan koulutus- ja testidatan määrä. Tulostetun puurakenteen juuressa olivat koko puun keskiarvometriikat jokaisesta luokasta ja opetettujen luokkien määrä sekä kaikki opetuksessa käytetyt asetukset. Tästä JSON-tiedostosta oli helppo katsoa tuloksia ja nähdä, missä luokissa mallit eivät suoriutuneet, sekä mahdollisia syitä mallien huonoon laatuun.

Mittaustuloksia tarkasteltiin ja havainnollistettiin vielä tarkemmin kuvassa 6. Kuvasta nähdään jokaisen algoritmin tuottama keskiarvotarkkuus jokaiselle datasetille. Algoritmit ovat saaneet jokaiseen datasettiin suurin piirtein samat tarkkuudet, minkä takia pelkkä tarkkuus ei riitä kuvaamaan mallien tehokkuutta. Datasetissä 1 on hieman eroavaisuuksia verrattuna muihin datasetteihin.



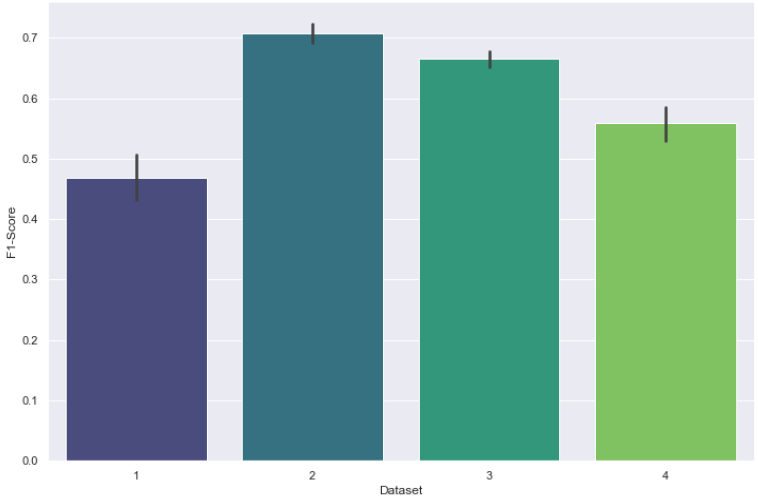
Kuva 6. Havainnollistus mitatuista keskiarvotarkkuuksista jokaisessa algoritmissa. Värikoodit vastaavat jokaista testissä käytettyä datasettiä.

Kuvassa 7 ovat jokaisen algoritmin tuottamat F1-pisteet jokaiselle datasetille. Kuvasta nähdään jo selkeämpiä eroavaisuuksia algoritmien välillä. Datasettiin 1 tehdyt mittaukset ovat selkeästi alhaisempia, ja se ei siten ole kovin hyvä malli käytettäväksi. Datasetin 1 mallit eivät ole oppineet luokittelemaan tarpeeksi hyvin positiivisia datapisteitä. Datasetin 2 F1-pisteet ovat selkeästi parhaat kuvassa, mutta datasettiin 3 tehdyt mittaukset eivät ole kovin kaukana perässä. Jokaisessa mallissa olisi vielä parannettavaa.



Kuva 7. Havainnollistus F1-pisteistä seitsemässä algoritmissa. Värikoodit vastaavat jokaista datasettiä, johon mittaus tehtiin.

Kuvassa 8 ovat keskiarvolliset mitatut F1-pisteet jokaiseen datasettiin. Kuvasta nähdään selkeämmin, että datasettiin 2 tehdyt mittaukset ovat parhaat ja sitä vasten opetetut mallit ovat oppineet parhaiten luokittelemaan. Datasettejä 2 ja 3 vasten opetettuja malleja on mahdollista testata oikeissa sovelluksissa ja katsoa, kuinka ne pärjäävät, vaikka näissä malleissa on silti vielä parannettavaa. Datasettien 1 ja 4 mittaustulokset ovat sen verran alhaiset, ettei niitä kannata testata oikeissa olosuhteissa.



Kuva 8. Havainnollistus mitatuista keskiarvollisista F1-pisteistä jokaisessa datasetissä.

4.4 Kielten luokittelu

Kielten luokittelu oli kaikista yksinkertaisin dataprosessi. Siinä luokiteltiin suomen- ja englanninkielisiä tekstejä. Kielten luokitteluun käytettiin datana Wikimedian englannin- ja suomenkielisiä jaettuja dokumentteja. Opetusdata jaettiin kahteen tekstitiedostoon, joista toisessa olivat suomenkieliset dokumentit ja toisessa englanninkieliset dokumentit. Yhden dokumentin teksti oli laitettu yhdelle riville, jolloin data pystyttiin lukemaan ML.NET-sovelluskehyksen datalukijalla. Datalukija määritetään esimerkkikoodissa 10, joka lukee ensimmäiseksi sarakkeeksi ”Id” ja toiseksi ”Text”. Datalukijalla luettiin molemmat tiedostot, joissa molemmissa oli 10 000 riviä dokumentteja.

```
var textLoader = _mlContext.Data.CreateTextLoader(new TextLoader.Arguments
{
      HasHeader = false,
      Column = new[]
      {
            new TextLoader.Column(“Id”, DataKind.I4, 0),
            new TextLoader.Column(“Text”, DataKind.TXT, 1)
      }
});
```

Esimerkkikoodi 10. Ote laaditusta datalukijasta C#-ohjelmointikielellä.

Kielten luokittelu suoritettiin monen luokan luokittelulla, vaikka dataprosessissa käytettiinkin vain kahta kieltä. Ajatuksena oli, että dataprosessiin on helposti lisättävissä lisää opetettavia kieliä tarvittaessa. Kielten nimet oli määritetty tekstitiedoston nimeksi, ja sillä tavoin jokaisen dokumentin tekstin tulossarakkeeseen laitettiin tekstitiedoston nimi. Kielten luokitteluun käytetty putki muistuttaa paljon ensimmäistä versiota dokumenttien luokittelusta, mutta kielten luokittelussa käytetään ominaisuutena vain dokumenttien tekstiä. Kielten nimet muutettiin ensiksi numeraalisiksi ja dokumenttien tekstiin käytettiin ”FeaturizeText”-metodia. Lopuksi putkeen lisättiin monen luokan luokittelija.

Taulukossa 5 on tehty mittauksia kielten luokittelijasta. Siinä on eri algoritmien tuloksia. Mittauksissa on mitattu algoritmien häviötä, jota *logaritminen häviö* (engl. logarithmic loss) kuvastaa. Logaritminen häviö rankaisee luokittelijan tekemiä vääriä luokitteluita. Logaritmista häviötä pyritään minimoimaan samaan tapaan kuin tarkkuutta maksimoimaan. [35.] Jokainen algoritmi onnistuu luokittelemaan lähes kaiken oikein, mutta Naive Bayes -algoritmin logaritminen häviö on kaikista suurin, ja siten se on myös

huonoin algoritmi tähän luokitteluun. Parhaiten kielten luokittelusta suoriutui Averaged Perceptron -algoritmi, koska sillä on suurin tarkkuus ja pienin logaritminen häviö.

Taulukko 5. Kielten luokittelun tarkkuudet ja logaritminen häviö algoritmikohtaisesti.

Algoritmi	**Tarkkuus**	**Logaritminen häviö**
Stochastic Gradient Descent (SGD)	0,994	0,0179
Stochastic Dual Coordinate Ascent (SDCA)	0,997	0,0129
Linear Support Vector Machines (LSVM)	0,998	0,0051
Naive Bayes (NB)	0,996	34,5388
Averaged Perceptron (AP)	0,999	0,0044
Logistic Regression (LOGR)	0,997	0,0192
Fast Tree (FT)	0,998	0,0059
Fast Forest (FF)	0,983	0,0587

Kielten luokitteludataprosessin tarkkuutta testattiin datasetin 1 dokumentteihin, koska datasetissä oli suomen ja englannin kielen dokumentteja sekaisin. Kielten luokittelijalla haluttiin nähdä, onnistuuko se oikeasti luokittelemaan dokumentteja suomen ja englannin kielen mukaan. Tätä varten käytettiin työkaluun tehtyä päätelmämoottoria, jolla voidaan tehdä ennustuksia dataan. Päätelmämoottoria käytetään esimerkkikoodin 11 mukaisesti, missä ladataan tallennettu malli polusta ja tehdään yksittäisiä ennustuksia dokumenttien tekstiin. Lopuksi ennustuksien tulokset tallennetaan listaan.

```
var results = new List<LanguageResult>();
var inferEngine = new InferEngine<LanguageObservation, LanguagePrediction>(
      _mlContext,
      savePath);

foreach(var observation in observations)
{
      var result = inferEngine.PredictSingle(observation);

      results.Add(new LanguageResult
      {
            Language = result.PredictedLabel,
            Text = observation.Text,
            Probabilities = result.Score
      };
}
```

Esimerkkikoodi 11. Ote laaditusta C#-ohjelmasta, jossa päätelmämoottorilla tehdään ennustuksia dokumenttien tekstiin.

Datasettiin 1 tehdyistä kielten ennustuksista katsottiin silmämääräisesti, menikö ne oikein. Datasetin 1 dokumenteista ei tiedetty siis etukäteen, onko ne kirjoitettu englanniksi vai suomeksi. Tuloksia tutkimalla datasettiin 1 tehdyt ennustukset lähes kaikki menivät oikein. Suomen ja englannin kielten luokittelua käytettiin datasetin 1 esikäsittelyssä, mistä mainittiin aikaisemmin dokumenttien luokittelussa. Kielten luokittelija onnistui hyvin, koska koulustusdataa oli paljon ja se oli tasapainossa.

4.5 Dokumenttien ryhmittely

Dokumenteista oli tarkoitus löytää samankaltaista dokumenttien sisältöä, jota voitaisiin esimerkiksi ehdottaa tai hakea käyttäjille katseluhistorian perusteella. Ryhmittelyllä pyritään löytämään datasta kaavamaisuuksia ja rakenteita. Dokumenttien ryhmittelyyn dataprosessissa käytetään KMeans-algoritmia ja PCA-projektiota havainnollistamaan dokumentteja OxyPlot-kirjastossa.

Aluksi dokumentit täytyi muuttaa ominaisuuksiksi samalla tavalla kuin luokitteluissakin. Putkeen määritettiin PCA-projektio ja KMeans-algoritmi esimerkkikoodin 12 tavalla. PCA-projektioon määritetään parametrilla, kuinka monessa ulottuvuudessa dataa halutaan kuvata. Datan kuvaus haluttiin vain kahdessa ulottuvuudessa, jotta dataa voi havainnollistaa kuvaajassa. KMeans-algoritmi ottaa vastaan PCA-projektion tulokset. KMeans-algoritmiin täytyi määrittää, kuinka monta ryhmää se luo, koska KMeans-algoritmi ei määritä optimaalisten ryhmien määrää. Ryhmien määrä annetaan säädettävän konsoliparametrin avulla.

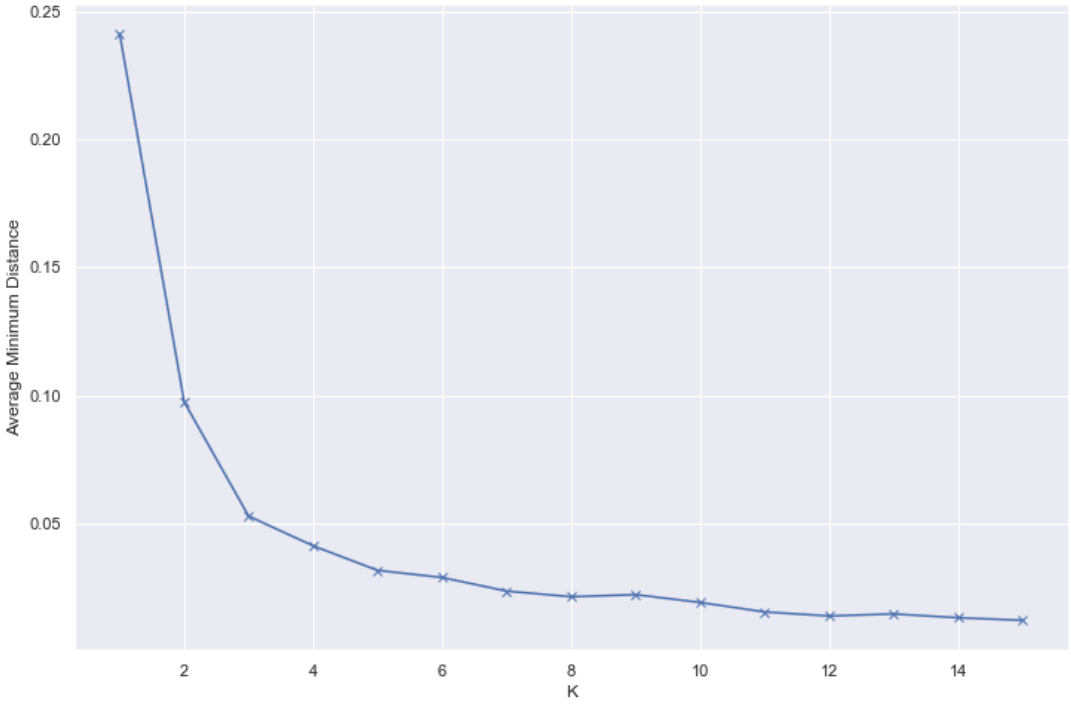
```
var dataProcessPipeline = _mlContext.Transforms.Text.TokenizeWords(
          “Keywords”, separators: new[] { '|' })
     .Append(_mlContext.Transforms.Text.FeaturizeText(“Features”,
          new[] { “Description”, “Title”, “Keywords” })
     .Append(_mlContext.Transforms.Projection.ProjectToPrincipalComponents(
          “PCAFeatures”, “Features”, rank: 2)
     .Append(_mlContext.Clustering.Trainers.KMeans(“PCAFeatures”,
          clustersCount: settings.KClusters);
```

Esimerkkikoodi 12. Laadittu ryhmittelyputki C#-ohjelmointikielellä. Ensin dokumentit muunnetaan ominaisuuksiksi. PCA-projektiota käytetään kaksiulotteisen datan kuvaamiseen ja KMeans-algoritmia ryhmittelyyn.

Optimaalisen ryhmien määrän löytämiseen on olemassa muutamia tekniikoita. Dataprosessissa käytettiin *kyynärpäämetodia* (engl. elbow method). Sen tarkoituksena

on kokeilla eri *K-arvoja* ja löytää sellainen *K-arvo*, jossa datapisteiden *keskiarvominimietäisyydet* (engl. average minimum distance) keskuksista eivät muutu rajusti korkeimmilla K-arvoilla. K-arvoilla tarkoitetaan siis ryhmien määrää, ja keskiarvominimietäisyydellä tarkoitetaan ryhmien tiiviyttä [36]. Ryhmien tiiviys lasketaan laskemalla keskiarvo jokaisen datapisteen lyhyimmästä etäisyydestä lähimmän ryhmän keskustaan. Kyynärpäämetodissa keskiarvominimietäisyydet keskuksista piirretään kuvaajaan, ja jos kuvaaja näyttää kädeltä, käden kyynärpää on optimaalinen K-arvo [37].

ML.NET-sovelluskehyksessä ryhmittelymallia voidaan evaluoida. Evaluointi palauttaa metriikoita, joista keskiarvominimietäisyyksistä oli kiinnostuttu kyynärpäämetodin takia. Datasettiin 1 kokeiltiin K-arvoiksi arvoja 1–15, ja näiden keskiarvominimietäisyyksiä on havainnollistettu kuvassa 9. Kuvasta nähdään, että kyynärpäämetodilla optimaalinen K-arvo on 5 datasetille 1.

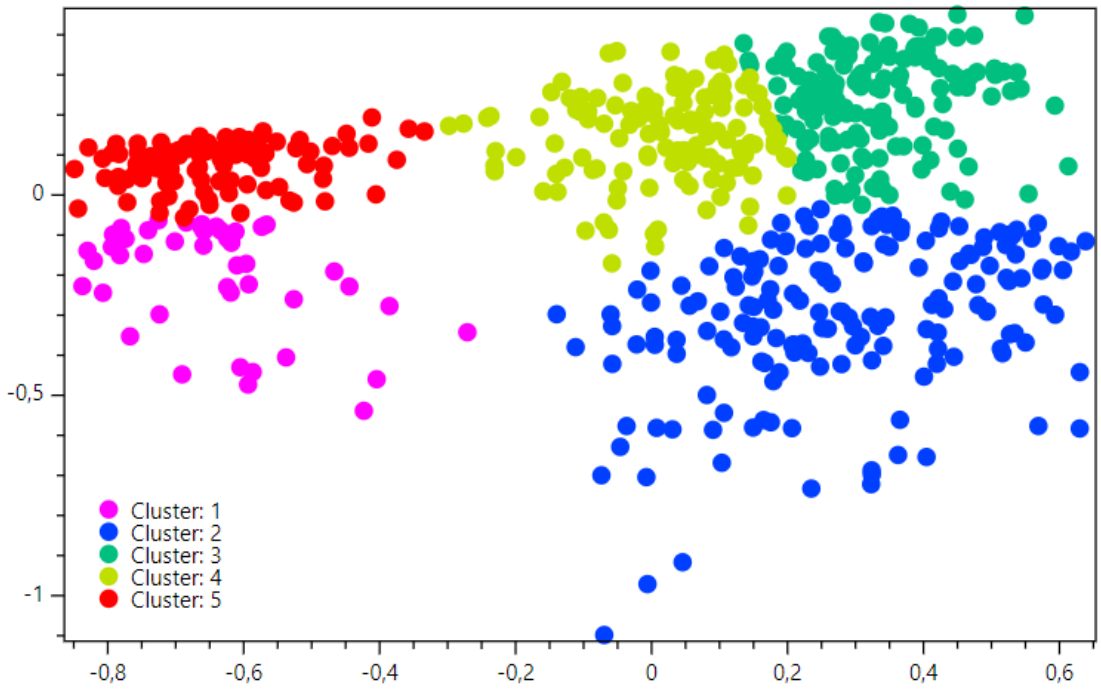


Kuva 9. KMeans-algoritmin datasetillä 1 evaluoidut keskiarvominimietäisyydet eri K-arvoilla.

Ryhmittelyn tulokset kannattaa piirtää kuvaajaan, josta näkee nopeasti, kuinka hyvin ryhmittely onnistui. Kuvaajien piirtämistä varten käytettiin OxyPlot-kirjastoa, jolla piirrettiin kaikki PCA-projektion datapisteet 2D-kuvaajaan. Datapisteet väritetään ryhmittelyn tuloksilla. Liitteen 2 Oxyplot-piirrosmetodilla voitiin piirtää datasetille 1

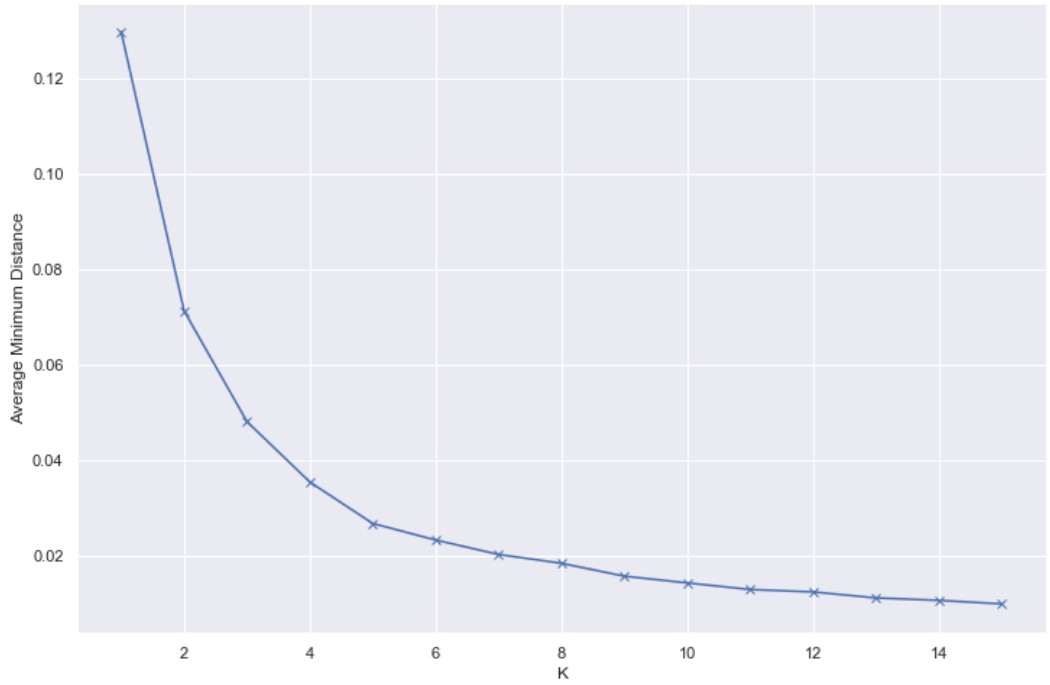
hajotuskuvaaja. Hajotuskuvaajan lopputulosta näytetään kuvassa 10, jossa käytettiin kyynärpäämetodilla määritettyä ryhmien määrää 5.

Kuvan 10 mukaan ryhmien 1 ja 5 dokumentit ovat selkeästi erilaisia kuin loput dokumentit. Jokaiseen dokumenttiin merkittiin ryhmittelystä tuotettu ryhmänumero, ja niitä tutkimalla huomattiin, että ryhmien 1 ja 5 dokumentit ovat kaikki suomenkielisiä ja ryhmissä 2, 3 ja 4 kaikki dokumentit ovat englanninkielisiä. Datasetin 1 ryhmittelyllä pystyttiin ymmärtämään, miksi dokumenttien luokittelussa datasetistä 1 poistetut suomenkieliset dokumentit pärjäsivät huonoiten monen luokan luokittelussa.



Kuva 10. Datasetin 1 hajotuskuvaaja PCA-projektioista, jossa on viisi ryhmää. Värit kuvastavat KMeans-algoritmin ryhmittelytuloksien eri ryhmiä.

Ryhmittelyä kokeiltiin vielä datasettiin 3 ja kyynärpäämetodilla selvitettiin myös optimaalinen K-arvo. Kuvassa 11 on havainnollistettu datasetin 3 keskiarvominimietäisyydet jokaiseen kokeiltuun K-arvoon. Kuvasta nähdään, ettei datasetillä 3 ole selkeää kyynärpäätä ja optimaalinen K-arvo voisi olla 5 tai 6. Datasettiin 3 käytettiin kuitenkin K-arvoa 6. Korkeampiakin K-arvoja olisi voitu käyttää, varsinkin kun datasetissä 3 on enemmän dataa kuin datasetissä 1.

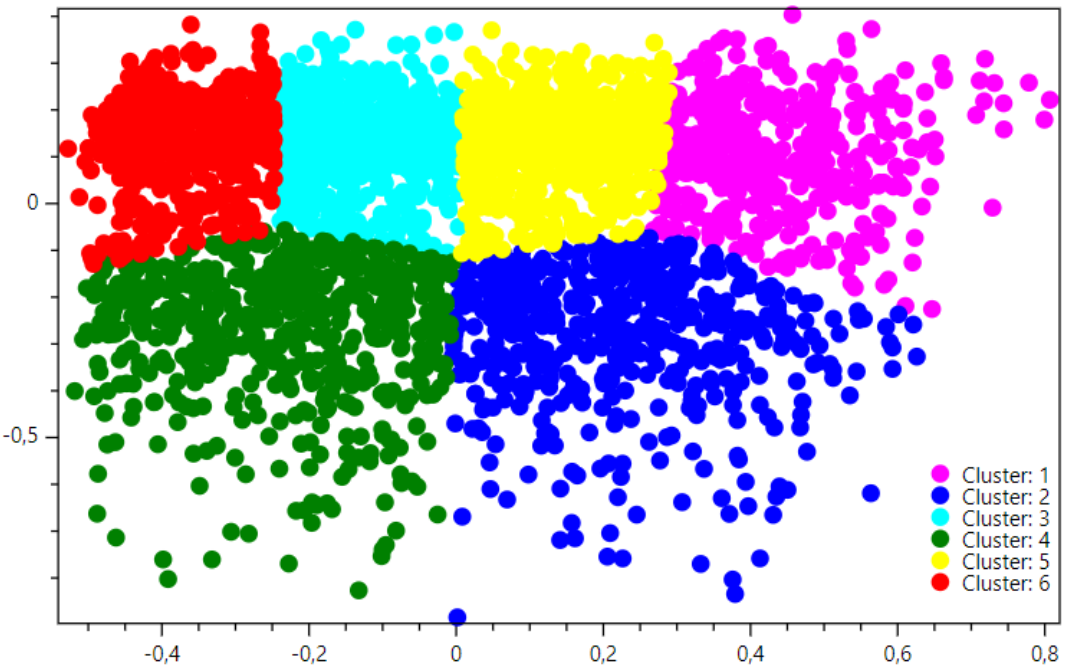


Kuva 11. KMeans-algoritmin datasettiin 3 mitatut keskiarvominimietäisyydet eri K-arvoilla.

Kuvassa 12 ovat datasettiin 3 tehdyn ryhmittelyn tulokset, jotka piirrettiin liitteen 2 OxyPlot-piirrosmetodilla. Dokumenteista on silmämääräisesti tutkimalla vaikea sanoa, ovatko saman ryhmän dokumentit samankaltaisia. Dokumenttien samankaltaisuutta voitaisiin mitata muutamilla tekniikoilla, joista yksinkertaisimpia ovat *Jaccard-* ja *kosinisamanlaisuus* (engl. Jaccard and cosine similarity) [38].

Jaccard-samanlaisuudessa kahden tekstin samanlaisuutta mitataan summaamalla teksteissä esiintyneet samat sanat yhteen ja jakamalla tämä summa molempien tekstien sanojen määrän summalla. Kosinisamanlaisuudessa mitataan kahden dokumenttivektorin välisestä kulmasta kosini. Jos dokumenttivektorit ovat täysin samanlaisia, niiden välinen kulma on 0°. Tämä tarkoittaa, että kosini saa silloin arvokseen 1. [38.]

Ryhmistä voitaisiin mitata dokumenttien samankaltaisuutta kosinisamanlaisuudella, koska ryhmittelyssä dokumentit ovat jo vektorimuodossa. Samankaltaisia dokumentteja voitaisiin siten ehdottaa käyttäjille ja tutkia käyttäjien katseluhistorian perusteella, ovatko käyttäjät kiinnostuneita ehdotuksista. Kehitystyön aikana ei kuitenkaan tehty käyttäjätason testiä.



Kuva 12. Datasetin 3 hajotuskuvaaja kuudella ryhmällä PCA-projektioista.

5 Johtopäätökset

Tässä luvussa tutkitaan ja arvioidaan dataprosesseihin liittyviä tuloksia ja ongelmia. Tuloksilla pyritään selvittämään, miten koneoppimisratkaisuja voidaan hyödyntää ja käyttää ja kuinka käyttökelpoisia ne ovat. Luvussa 5.1 käsitellään dataprosessien ongelmia ja selostetaan, mitä mahdollisuuksia on vaikuttaa niihin. Luvussa 5.1 kerrotaan, kuinka hyvin koneoppimisratkaisujen tavoitteet saavutettiin ja mitä niissä voisi tehdä toisin. Lopuksi selostetaan kehitettyjen koneoppimisratkaisujen tulevaisuuden näkymistä luvussa 5.2.

5.1 Dataprosessien analyysi

Kehitystyö alkoi siitä, että ML.NET-sovelluskehyksellä kehitettiin työkalu, jolla pystyttiin helposti ja nopeasti kehittämään useita dataprosesseja. ML.NET-sovelluskehys vaikutti ympäristönsä takia tarpeeksi hyvältä sovelluskehykseltä kehittää koneoppimisratkaisuja. Työkalusta suunniteltiin mahdollista palvelua, joka automatisoisi koneoppimismallien

opettamisen. Jotta työkalun voisi automatisoida, täytyy tehtyjen dataprosessien olla tarpeeksi laadukkaita.

Dokumenttien luokitteluun keskityttiin koneoppimisratkaisuista eniten, koska siinä oli selkeä tavoiteltava määränpää. Dokumenttien luokittelu koki kehitystyön aikana useita eri muutoksia, joilla pyrittiin tuottamaan parasta mahdollista dokumenttien luokittelumallia. Dataprosessilla pyrittiin siis luomaan malli, joka pystyisi automatisoimaan dokumenttien luokittelun tuotantosovelluksissa. Käytössä olleiden datasettien perusteella kuitenkin huomattiin, että dokumenttien luokittelussa on vielä parannettavaa. Dokumenttien luokittelumalli oli kuitenkin laadultaan suhteellisen hyvä eri dataseteillä, kun se teki parhaimmillaan virheen joka viidennessä dokumentissa. Siksi sitä voitaisiin kokeilla oikeassa ympäristössä ja mitata, kuinka hyvin se onnistuu luokittelemaan.

Dokumenttien luokittelumalliin voitaisiin vaikuttaa muokkaamalla ja käsittelemällä dataa tarkemmin, varsinkin kun tekstimuotoista dataa on haastavampaa opettaa tietokoneille. Esimerkiksi datasta voitaisiin poistaa sellaisia dokumentteja, joissa on vain vähän tekstiä. Samoin kantasanoittaminen tai perusmuotoistaminen voisi auttaa oppimisessa, mutta näistä ei ollut toteutusta ML.NET-sovelluskehyksessä kehitystyötä tehtäessä. Sekin on mahdollista, että osa dokumenteista on luokiteltu alkuperäisesti väärin, mikä saattaisi heikentää tuloksia. Tätä on kuitenkin vaikea selvittää datasta.

Dokumenttien luokittelumallilla ei täysin saavutettu koneoppimisratkaisulle asetettua tavoitetta. Luokittelumalli ei pystynyt tekemään tarpeeksi varmasti päätöksiä testattuun dataan, ja siksi sillä ei voida automatisoida dokumenttien luokittelua kokonaan. Tuloksista kuitenkin nähdään, että dokumenttien luokittelumalli tuotti hyvän perustan jatkokehitykselle.

Kielten luokittelu oli koneoppimisratkaisuista positiivinen yllätys tuloksien kannalta, mutta se ei ollut niin arvokas dataprosessi verrattuna dokumenttien luokitteluun. Kielten luokittelu oli kokeilukoneoppimisratkaisu, jolla voitaisiin etsiä samankielisiä dokumentteja sovelluksista. Kielten luokittelu oli laadultaan lähes täydellinen, ja tämä malli ei ollut sidoksissa käytettyihin datasetteihin, koska kielten luokittelija opetettiin Wikimedian datalla. Kielten luokittelijaa voitaisiin jatkossa hyödyntää osana datan esikäsittelyä, esimerkiksi sellaisissa tilanteissa, joissa tekstimuotoisesta datasta pitäisi poistaa tietyn

kielen pysäytyssanat ja kielten luokittelija pystyisi tunnistamaan tämän kielen ja siten valitsemaan oikeat pysäytyssanat poistettavaksi.

Samankaltaisten dokumenttien ryhmittelystä on vaikea sanoa, kuinka hyvin se onnistui. Kehitystyössä tuotettujen hajotuskuvaajien perusteella onnistuttiin tekemään selkeitä ryhmiä, ja samankaltaisuutta voitiin mitata kosinisamanlaisuudella. Kuitenkin ryhmittelyssä onnistuttiin erottelemaan erikielisiä dokumentteja toisistaan, koska erikieliset datapisteet ovat rakenteeltaankin erilaisia. Samankaltaisia dokumentteja voitaisiin ehdottaa käyttäjille ja katsoa, vaikuttaako se käyttäjien katseluhistoriaan.

Ryhmittelyssä voitaisiin yrittää ryhmitellä käyttäjät katseluhistorian mukaan, kun tiedetään, mitä dokumentteja käyttäjät ovat katsoneet. Tämän perusteella käyttäjät, jotka katsovat samoja dokumentteja, ryhmittyisivät samankaltaisiin ryhmiin. Silloin voitaisiin käyttäjäryhmistä ehdottaa dokumentteja, joita saman ryhmän käyttäjät eivät ole katselleet. Tällaista ryhmittelyä suunniteltiin koneoppimisratkaisuksi, mutta sitä ei toteutettu, koska ML.NET-sovelluskehyksessä katseluhistorian dataa ei ollut yksinkertaista muuttaa datanäkymäksi, jossa ovat jokaisen käyttäjän kaikki katsotut ja katselemattomat dokumentit.

Dataprosessien kehittäminen auttoi ymmärtämään paljon koneoppimisesta ja loi perustan tulevaisuuden koneoppimisratkaisuihin. Dataprosesseissa datan esikäsittelyssä, ominaisuuksien muokkaamisessa ja tulosten esilletuonnissa kului kehitystyössä kaikista eniten aikaa. Koneoppimisratkaisujen kehittämisestä tarvitaan vielä lisää kokemusta, jotta saadaan parempia tuloksia tulevaisuudessa.

5.2 Koneoppimisratkaisujen tulevaisuus

ML.NET-sovelluskehys oli kehitystyötä tehtäessä aktiivisessa kehityksessä, ja siksi siitä puuttuu vielä tarpeellisia toteutuksia, jotka helpottaisivat kehitystyötä. Koneoppimisratkaisuja kehitetään jatkossa, mutta tulevaisuudessa todennäköisesti hyödynnetään ONNX-formaattia, jolloin koneoppimismallien opettaminen tehtäisiin toisella koneoppimissovelluskehyksellä. Mahdollisuuksia on paljon, ja Python- ja R-ohjelmointikielelle näyttäisi olevan kehitystyötä tehtäessä eniten valmiita työkaluja.

Kehitystyössä tehtyä päätelmämoottoria käytetään ainakin ennustuksien tekemisessä ONNX-formaattia hyödyntäen, jotta koneoppimismalleja voidaan käyttää .NET-sovelluksissa. ONNX-formaatissa olevia malleja voidaan ladata ML.NET-sovelluskehyksessä ja siten tehdä ennustuksia havaintoihin tuotantosovelluksissa. Syväoppiminenkaan ei ole poissuljettu mahdollisuus. ML.NET-sovelluskehys tukee muutamia syväoppimiskirjastoja, joilla olisi mahdollista yrittää esimerkiksi dokumenttien luokittelua.

6 Yhteenveto

Insinöörityön tarkoitus oli perehtyä koneoppimisen perusteisiin ja mahdollisuuksiin ja oppia käyttämään ML.NET-sovelluskehystä. Insinöörityössä tehdyssä kehitystyössä tarkoituksena oli kehittää ja luoda käytännön koneoppimisratkaisuja ML.NET-sovelluskehyksellä projektin teettäneen yrityksen datalla. Koneoppimisratkaisuista dokumenttien luokitteluun keskityttiin eniten, mutta kehitystyössä tehtiin myös kielten luokittelua ja dokumenttien ryhmittelyä.

Dokumenttien luokittelu suoritettiin aluksi monen luokan luokitteluna, mutta kehitystyön edetessä huomattiin, että dokumentit pystyivät olemaan eri dataseteillä monessa luokassa samanaikaisesti. Dokumenttien luokittelussa tehtiin kompromissi, ja monen tuloksen luokittelu tehtiin binääriluokitteluna jokaista luokkaa kohden, koska monen tuloksen luokittelua ei ollut mahdollista tehdä ML.NET-sovelluskehyksessä kehitystyötä tehtäessä. Dokumenttien binääriluokittelu tuotti kokeilukelpoisia tuloksia tuotantotason sovelluksiin, mutta tavoitteeseen ei päästy. Dokumenttien luokittelulla tavoiteltiin automatisoitua dokumenttien luokittelua tuotantosovelluksissa.

Kielten luokittelu oli kehitystyössä suoraviivainen putki, jonka tulokset hipoivat täydellisyyttä. Kielten luokitteluun käytettiin Wikimedian dataa. Tällä datalla onnistuttiin opettamaan malli, joka onnistui luokittelemaan lähes täydellisesti suomen ja englannin kielen dokumentit. Kielten luokittelumallille pystyisi helposti opettamaan muitakin kieliä kehitystyössä tehdyllä toteutuksella.

Dokumenttien ryhmittelyssä pyrittiin löytämään samankaltaisia dokumentteja datasta, jotta voitaisiin auttaa käyttäjiä löytämään uusia samankaltaisia dokumentteja käyttäjien

katseluhistorian perusteella. Ryhmittelyssä onnistuttiin luomaan kuvaajien perusteella selkeitä ryhmiä. Samankaltaisilla dokumenteilla voidaan tulevaisuudessa ehdottaa käyttäjille uusia dokumentteja ja mitata sen vaikutuksia.

Insinöörityön avulla saatiin selville koneoppimisen mahdollisuuksia ja vaatimuksia. Insinöörityössä saatiin käytännön kokemusta ML.NET-sovelluskehyksestä ja opittiin koneoppimisen perusteista ja metriikoista. Insinöörityössä opittiin myös suhtautumaan kriittisesti saatuihin metriikoihin ja tuloksiin. ML.NET-sovelluskehyksessä huomattiin puutteita, koska se oli kehitystyötä tehtäessä kehitysvaiheessa. Tulevaisuudessa todennäköisesti käytetään jotain toista koneoppimissovelluskehystä koneoppimismallien opettamisessa, mutta ML.NET-sovelluskehystä käytetään havaintojen ennustamisessa .NET-sovelluksissa ONNX-formaattia hyödyntäen.

Insinöörityössä tehdyillä koneoppimisratkaisuilla onnistuttiin luomaan hyvä perusta jatkokehitykselle. Koneoppimiskehitystyöstä tarvitaan vielä lisäkokemusta, jotta tulevaisuudessa saadaan parempia tuloksia. Koneoppimisratkaisujen kehittämiseen on paljon mahdollisuuksia, ja mahdollisuudet pidetään avoimina.

Lähteet

1 Fagella, Daniel. 2019. What is Machine Learning? Verkkoaineisto. <https://emerj.com/ai-glossary-terms/what-is-machine-learning/>. 19.2.2019. Luettu 1.3.2019.

2 Badreesh, Shetty. 2018. Supervised Machine Learning: Classification. Verkkoaineisto. <https://towardsdatascience.com/supervised-machine-learning-classification-5e685fe18a6d>. 12.12.2018. Luettu 1.3.2019.

3 Brownlee, Jason. 2016. Supervised and Unsupervised Machine Learning algorithms. Verkkoaineisto. <https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/>. 16.3.2016. Luettu 1.3.2019.

4 James, Gareth; Witten, Daniela; Hastie, Trevor & Tibshirani, Robert. 2013. An Introduction to Statistical Learning. New York: Springer Science, Business Media.

5 Brownlee, Jason. 2017. Difference Between Classification and Regression in Machine Learning. Verkkoaineisto. <https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/>. 11.12.2017. Luettu 1.3.2019.

6 Iris Species. 2016. Verkkoaineisto. Alphabet Inc. <https://www.kaggle.com/uciml/iris>. 27.9.2016. Luettu 1.3.2019.

7 Peikari, Mohammad; L. Martel, Anne; Salama, Sherine & Nofech-Mozes, Sharon. 2018. A Cluster-then-label Semi-supervised Learning Approach for Pathology Image Classification. Verkkoaineisto. <https://www.nature.com/articles/s41598-018-24876-0>. 8.5.2018. Luettu 4.3.2019.

8 Oppermann, Artem. 2018. Self Learning AI-Agents Part I: Markov Decision Processes. Verkkoaineisto. <https://towardsdatascience.com/self-learning-ai-agents-part-i-markov-decision-processes-baf6b8fc4c5f>. 14.10.2018. Luettu 4.3.2019.

9 Goodfellow, Ian; Bengio, Yoshua & Courville, Aaron. 2016. Deep Learning. E-kirja. MIT Press.

10 A Beginner’s Guide to Neural Networks and Deep Learning. Verkkoaineisto. Skymind Inc. <https://skymind.ai/wiki/neural-network>. Luettu 5.3.2019.

11 A Beginner’s Guide to Convolutional Neural Networks (CNNs). Verkkoaineisto. Skymind Inc. <https://skymind.ai/wiki/convolutional-network>. Luettu 6.3.2019.

12 Recurrent Networks. Verkkoaineisto. Skymind Inc. <https://skymind.ai/wiki/recurrent-network-rnn>. Luettu 6.3.2019.

13 Geitgey, Adam. 2018. Natural Language Processing is Fun! Verkkoaineisto. <https://medium.com/@ageitgey/natural-language-processing-is-fun-9a0bff37854e>. 18.7.2018. Luettu 7.3.2019.

14 Jabeen, Hafsa. 2018. Stemming and Lemmatization in Python. Verkkoaineisto. <https://www.datacamp.com/community/tutorials/stemming-lemmatization-python>. 23.9.2018. Luettu 7.3.2019.

15 Hulstaert, Lars. 2017. LDA2vec: Word Embeddings in Topic Models. Verkkoaineisto. <https://www.datacamp.com/community/tutorials/lda2vec-topic-model>. 19.10.2017. Luettu 8.3.2019.

16 D. Manning, Christopher; Raghavan, Prabhakar & Schütze, Hinrich. 2008. Introduction to Information Retrieval. Cambridge University Press.

17 Carter, Phillip; Koritzinsky, Jeremy; Petrusha, Ron; Wenzel, Maira & Jones, Mike. 2017. Tour of .NET. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/standard/tour>. 22.5.2017. Luettu 11.3.2019.

18 Lander, Rich; Wenzel, Maira & Casey, Graig. 2018. About .NET Core. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/core/about>. 1.8.2018. Luettu 11.3.2019.

19 Wenzel, Maira & Latham, Luke. 2019. .NET Standard. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/standard/net-standard>. 25.2.2019. Luettu 11.3.2019.

20 Gronlund, C.J & Alexander, John. 2019. What is ML.NET and how do I understand Machine Learning basics? Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/machine-learning/what-is-mldotnet>. 1.3.2019. Luettu 8.3.2019.

21 Moseley, Dan; Madero, Santiago, Fernandez; Erhardt, Eric; Siddiqui, Zeeshan; Forkmann, Steffen; Harwell, Sam; Wenzel, Maira; Ormont, Justin; Nazirov, Gani; De la Torre, Cesar; Asthana, Ankit & Ahmed, Zeeshan. 2019. Machine Learning for .NET. Verkkoaineisto. <https://github.com/dotnet/machinelearning/blob/master/README.md>. 6.3.2019. Luettu 11.3.2019.

22 Alexander, John; Gronlund, C.J; Kulikov, Petr; Wenzel, Maira & Dugar, Aditi. 2019. Machine learning tasks in ML.NET. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/machine-learning/resources/tasks>. 15.1.2019. Luettu 8.3.2019.

23 What is ML.NET? Verkkoaineisto. Microsoft Corporation. <https://dotnet.microsoft.com/learn/machinelearning-ai/what-is-mldotnet>. Luettu 11.3.2019.

24 ONNX. 2019. Verkkoaineisto. Facebook Inc. <https://onnx.ai>. 23.1.2019. Luettu 11.3.2019.

25 Alexander, John; Wenzel, Maira; Kulikov, Petr; Wagner, Bill & Dugar, Aditi. 2019. Machine learning glossary of important terms. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/machine-learning/resources/glossary>. 5.3.2019. Luettu 12.3.2019.

26 Sethi, Nishant. 2018. A Machine Learning Classification Model in Python – Part I. Verkkoaineisto. <https://medium.com/@nsethi610/a-machine-learning-classification-model-in-python-e97dece85f09>. 28.8.2018. Luettu 12.3.2019.

27 Filipi, Senja; Chin, Wei-Shen; Luferenko, Pete; Pagnoni, Artidoro; Goswami, Abhishek; Shariq, Sabah & Moradi, Shahab. 2019. ML.NET Cookbook. Verkkoaineisto. <https://github.com/dotnet/machinelearning/blob/master/docs/code/MlNetCookBook.md>. 13.3.2019. Luettu 14.3.2019.

28 Finley, Tom; Wenzel, Maira & Erhardt, Eric. 2019. IDataView Design Principles. Verkkoaineisto. <https://github.com/dotnet/machinelearning/blob/master/docs/code/IDataViewDesignPrinciples.md>. 26.2.2019. Luettu 12.3.2019.

29 Alexander, John; Quintanilla, Luis & Gronlund, C.J. 2019. Basic concepts for model training in ML.NET. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/machine-learning/basic-concepts-model-training-in-mldotnet>. 5.3.2019. Luettu 12.3.2019.

30 Alexander, John; Schonning, Nick; Quintanilla, Luis & Wenzel, Maira. 2019. Apply feature engineering for machine learning model training on textual data with ML.NET. Verkkoaineisto. <https://docs.microsoft.com/en-us/dotnet/machine-learning/how-to-guides/train-model-textual-ml-net>. 5.3.2019. Luettu 14.3.2019.

31 Warren, Genevieve & G. Lee, Terry. 2019. Welcome to the Visual Studio IDE. Verkkoaineisto. <https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2017>. 21.2.2019. Luettu 12.3.2019.

32 Nandwani, Karan; Wenzel, Maira; Brockschmidt, Kraig & Myers, Alfred. 2018. An introduction to NuGet. Verkkoaineisto. <https://docs.microsoft.com/en-us/nuget/what-is-nuget>. 10.1.2018. Luettu 12.3.2019.

33 Nandwani, Karan; Wenzel, Maira; Brockschmidt, Kraig; Cenerelli, Ken; Matthijssen, Louis & McMaster, Nate. 2017. NuGet Package Manager UI.

Verkkoaineisto. <https://docs.microsoft.com/en-us/nuget/tools/package-manager-ui>. 8.12.2017. Luettu 12.3.2019.

34 Swalin, Alvira. 2018. Choosing the Right Metric for Evaluating Machine Learning Models – Part 2. Verkkoaineisto. <https://medium.com/usf-msds/choosing-the-right-metric-for-evaluating-machine-learning-models-part-2-86d5649a5428>. 2.5.2018. Luettu 21.3.2019.

35 B. Collier, Andrew. 2015. Making Sense of Logarithmic Loss. Verkkoaineisto. <https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/>. 14.12.2015. Luettu 22.3.2019.

36 ClusteringMetrics Class. 2019. Verkkoaineisto. Microsoft Corporation. <https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.data.clusteringmetrics?view=ml-dotnet>. Luettu 23.3.2019.

37 Alade, Tola. 2018. How to determine the optimal number of clusters for k-means clustering. Verkkoaineisto. <https://blog.cambridgespark.com/how-to-determine-the-optimal-number-of-clusters-for-k-means-clustering-14f27070048f>. 27.5.2018. Luettu 23.3.2019.

38 Sieg, Adrien. 2018. Text similarities: Estimate the degree of similarity between two texts. Verkkoaineisto. <https://medium.com/@adriensieg/text-similarities-da019229c894>. 5.7.2018. Luettu 17.4.2019.

Putkirajapinta

```
internal interface IPipeline
{
      void AddDataProcessPipeline(IEstimator<ITransformer> pipeline);

      void AddTrainer(IEstimator<ITransformer> trainer);

      ITransformer Train(IDataView trainingData);

      RegressionMetrics EvaluateRegressionModel(
            IDataView testData,
            string labelColumn = DefaultColumnNames.Label,
            string scoreColumn = DefaultColumnNames.Score);

      CalibratedBinaryClassificationMetrics EvaluateBinaryClassificationModel(
            IDataView testData,
            string labelColumn = DefaultColumnNames.Label,
            string scoreColumn = DefaultColumnNames.Score);

      MultiClassClassifierMetrics EvaluateMultiClassClassificationModel(
            IDataView testData,
            string labelColumn = DefaultColumnNames.Label,
            string scoreColumn = DefaultColumnNames.Score);

      ClusteringMetrics EvaluateClusteringModel(
            IDataView testData,
            string scoreColumn = DefaultColumnNames.Score,
            string featuresColumn = DefaultColumnNames.Features);

      void Save(string savePath);
}
```

OxyPlot-piirrosmetodi

```
public static void CreatePlotChart(
      IEnumerable<ClusterPrediction> predictions,
      string plotSavePath,
      string title,
      LegendPosition legendPosition)
{

      var plot = new PlotModel
      {
            Title = title,
            IsLegendVisible = true,
            LegendPosition = legendPosition,
            LegendLineSpacing = 2
      };

      clusters = predictions.Select(p => p.ClusterId)
            .Distinct()
            .OrderBy(x => x);

      foreach (var cluster in clusters)
      {
            var scatterSeries = new ScatterSeries
            {
                  MarkerType = MarkerType.Circle,
                  MarkerStrokeThickness = 1,
                  Title = $"Cluster: {cluster}",
                  RenderInLegend = true
            };

            var points = predictions
                  .Where(p => p.ClusterId == cluster)
                  .Select(p => new ScatterPoint(p.Location[0], p.Location[1]))
                  .ToArray();

            scatterSeries.Points.AddRange(points);
            plot.Series.Add(scatterSeries);
      }

      plot.DefaultColors = OxyPalettes.HueDistinct(plot.Series.Count).Colors;
      plotSavePath = Path.ChangeExtension(plotSavePath, ".svg");

      var exporter = new SvgExporter { Width = 600, Height = 400 };
      using (var fs = new FileStream(plotSavePath, FileMode.Create))
            exporter.Export(plot, fs);
}
```