NLP VUONNA 2021, AWS JA TEKSTIN LUOKITTELU

LAB-AMMATTIKORKEAKOULU
Insinööri (ylempi AMK)
Digitaaliset ratkaisut
Kevät 2021
Miika Jukakoski

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Jukakoski, Miika	Opinnäytetyö,ylempi AMK	Kevät 2021
	Sivumäärä	
	71	

Työn nimi

NLP vuonna 2021, AWS ja tekstin luokittelu

Tutkinto

Insinööri (ylempi AMK), Digitaaliset ratkaisut

Tiivistelmä

Tässä työssä tutkittiin luonnollisen kielen käsittelyn (engl. *natural language processing*) teknologista kypsyyttä vuonna 2021. Tarkempi tutkimuksen kohde oli suomen kieli ja NLP-tehtävistä tekstiaineiston luokittelu. Työssä tutustuttiin uusimpiin arkkitehtuureihin, joilla NLP-tehtävien suorituskykyä on saatu nostettua suurin harppauksin viime vuosina. Tehokkaat arkkitehtuurit ovat perustuneet poikkeuksetta neuroverkkoihin, mutta vuonna 2017 julkaistu Attention-malli on mahdollistanut erityisen suuren edistyksen NLP-tehtävissä.

Samaan aikaa neuroverkkojen kehityksen kanssa on myös teknologia kehittynyt. Kehitys on mahdollistanut mallien nopeamman koulutuksen, joka on edelleen nopeuttanut neuroverkkojen kehitystä. Suorituskykyiset laitteistot ovat jo kaikkien saatavilla, joilla voi kouluttaa malleja tai suorittaa päättelytehtäviä. Laitteistoa ei tarvitse enää itse omistaa, vaan markkinoilla on useita toimijoita, joilta on mahdollista vuokrata laskentakapasiteettia niin koulutukseen kuin hostaamiseenkin. Palveluntarjoajista tässä työssä keskityttiin Amazon Web Services-yrityksen palvelutarjontaan. AWS on tällä hetkellä markkinajohtaja pilvipalveluissa, joten heidän tarjontansa myös NLP-palveluissa on kattava.

NLP-mallit ovat tyypillisesti hyvin kookkaita, joten mallin kouluttaminen ja hostaaminen vaativat tehokkaan GPU:n ja riittävästi käyttömuistia. Suuri tehovaatimus varsinkin hostauksessa tarkoittaa sitä, että mallia varten on oltava jatkuvasti päällä oleva palvelin. Tässä työssä tutkittiin päättelytehtävien ajamista Serverless-ympäristössä, jolloin satunnaisella käytöllä olevan mallin kustannukset olisi mahdollista pitää kohtuullisina. Työssä toteutettiin Serverless-arkkitehtuurilla suomen kielen luokittelija, joka osaa luokitella uutisaineistoa kymmeneen eri luokkaan.

Asiasanat

NLP, Serverless, AWS, tekoäly, koneoppiminen

Abstract

Author(s)	Type of publication	Published
Jukakoski, Miika	Master's thesis	Spring 2021
	Number of pages	
	71	

Title of publication

NLP in 2021, AWS and text classification tasks

Name of Degree

Master of Engineering, Digital Solutions

Abstract

This thesis was about the maturity of natural language processing (NLP) in 2021. Subtopic in the vast field of NLP was text classification in Finnish language. Aim of this thesis was to explore the state-of-the-art practices for text classification tasks in Finnish and especially in cloud. Latest solutions have been based primarily on artificial neural networks and there have been huge steps regarding accuracy and performance in last years. For example, Attention-model released in 2017 by Google Research was a game changer in many ways.

At the same time with network architecture evolvement also hardware has been evolved a lot. Better hardware enables faster model training which leads to faster model development. This has led to positive feedback loop and boosted the entire domain to the next level. Lower price of hardware has also given the possibility to everyone to practice machine learning. You don't even need to own the hardware yourself as cloud providers like Google give the possibility to use their unused capacity for free. However, in this thesis the focus was to study the services that AWS has to offer for NLP.

NLP-models are typically huge in megabytes and training the model requires a powerful GPU with over 10GB of GPU RAM-memory. Also, the hosting needs resources, as the trained model has to be kept in memory to enable low latency in inference and the server must be running all the time. Despite of these constraints the possibility to use Serverless architecture for text classification was studied in this thesis. Serverless architecture is very cost wise if use of service is low or sporadic. As a part of this thesis an application was developed for Finnish text classification using Serverless Lambda functions in AWS cloud.

Keywords

NLP, Serverless, AWS, Artificial Intelligence, Machine learning

SISÄLLYS

KÄSITELUETTELO

Anaconda	Ohjelmisto, jolla voi luoda virtuaalisia ohjelmien suoritusympäristöjä. Data Scientist:it käyttävät virtualisointia, sillä ohjelmistokirjastot ovat usein riippuvaisia toisten kirjastojen tietyistä versioista.
API	Application Programming Interface. Rajapinta, jonka kautta käytetään esimerkiksi ohjelmistoa tai verkkopalvelua.
AWS	Amazon Web Services. Amazon yrityksen julkinen pilvipalvelu.
BERT	Bidirectional Encoder Representations from Transformers.
CBOW	Continuous Bag of Words. Menetelmä, jolla voidaan luoda word2vec-mallin koulutuksen yksi datapiste. Ympäröivien sanojen perustella pyritään arvaamaan puuttuva sana. Verkon tulokerroksella ovat ympäröivät sanat ja ulostulokerroksella puuttuva sana.
CLM	Causal Language Model
Colab	Googlen tarjoama ilmainen palvelu Python Notebookien ajamiseen. Palvelu luo käyttäjälle virtuaalisen ajoympäristön, jossa käytössä on oma keskeiset laiteresurssit sisältämä virtuaalikone. Varsinkin näytönohjaimet ovat erittäin tehokkaita.
Epoch	Yksikkö, joka kertoo sen, montako kertaa koulutukseen käytettävää tietojoukkoa käytetään mallin koulutukseen.
FaaS	Function as a Service. Pilviympäristössä ajettava funktio. Sovellus kutsuu pilvessä olevaa funktiota, joka ajetaan määritellyssä ajoympäristössä. Ajoympäristö on tyypillisesti Linux-pohjainen kontti, jossa ajetaan esimerkiksi Python – koodia.
Framework	Ohjelmistokirjasto, jolla on mahdollista luoda sovelluksia. Frameworkissa on yleensä joko jonkinlainen graafinen käyttöliittymä tai API-rajapinta.
GPU	Graphics Processing Unit. Näytönohjain.
JSON	Javascript Object Notation. Yleisesti käytössä oleva viestirakenteen notaatio. Suurin osa verkkoliikenteen on JSON-muotoisia viestejä.
Lambda	AWS:n termi pilviympäristössä ajettavasta funktiosta (FaaS).
MLM	Masked Language Model
NLP	Natural Language Processing. Luonnollisen kielen käsittelyä.
NLU	Natural Language Understanding. Luonnollisen kielen ymmärtämistä.
Perusmuotoistus	Sanat muutetaan niiden perusmuotoon. Esimerkiksi sanan ”kissojen” perusmuoto on ”kissa”.

PyTorch	Facebookin luoma Python Framework, jolla voi luoda tekoälysovelluksia.
RNN	Recurrent Neural Network. Neuroverkko, jolla voidaan käsitellä sekvenssimuotoista dataa. Esimerkiksi lauseet ovat sanojen sekvenssejä.
Skip-gram	Menetelmä, jolla voidaan luoda word2vec-mallin koulutuksen yksi datapiste. Lähdesanan perustella pyritään arvaamaan lähdesanan läheiset sanat. Verkon tulokerroksella on lähdesana ja ulostulokerroksella läheiset sanat.
TensorFlow	Googlen luoma Python Framework, jolla voi luoda tekoälysovelluksia.
Tensori	Moniulotteinen vektori.
TPU	Tensor Processing Unit. Prosessorityyppi, jolla on mahdollista laskea vektorilaskutoimituksia tehokkaasti.
tsv	Tab Separated Values. Tiedosto, jossa data on erotettu sarakkeisiin sarkainerottimella.
XLM	Cross-Lingual Model. Kielimalli, joka pystyy käsittelemään useita kieliä.

1 JOHDANTO

NLP on tällä hetkellä yksi nopeimmin kehittyvä alaluokka tekoälystä puhuttaessa. Viime vuosikymmenen aikana algoritmien, neuroverkkoarkkitehtuurien, teknologian ja pilviympäristöhän kehitys ovat mahdollistaneet tekoälyn tuomisen moniin arkipäiväisiin tehtäviin. Laskentakapasiteetti, joihin vielä vuosikymmen sitten vain suurimmilla teknologiayrityksillä oli varaa, on nyt käytännössä kaikkien kuluttajien saatavilla. Alle kahdentuhannen euron investoinnilla on mahdollista rakentaa erittäin suorituskykyinen laitteisto tekoälytehtävien suorittamiseen. Varsinkin kehitys GPU:issa on tuonut suuren harppauksen erityisesti neuroverkkopohjaisissa arkkitehtuureissa. Neuroverkoissa laskenta on käytännössä operaatioiden suoritusta isoilla matriiseilla, joista GPU:t suoriutuvat erittäin hyvin. Viime aikoina markkinoille on tullut myös matriisilaskentaan erityisesti suunniteltuja prosessoreita, eli TPU:ta (Tensor Processing Unit). Nämä yksiköt mahdollistavat äärimmäisen nopeaa mallien suoritusta.

Samaan aikaan, kun teknologia on kehittynyt ja laitteet muuttuneet halvemmiksi, on kilpailu julkisten pilvitarjoajien välillä kasvanut. Pilvet tarjoavat suorituskykyistä rautaa minuuttiveloituksella ja heidän kirjastoistaan löytyy runsaasti valmiita algoritmeja sekä malleja. Toimijat ovat kehittäneet myös graafisia kehitysympäristöjä (IDE), joiden avulla kynnys tekoälysovelluksen rakentamiseen on madaltunut huomattavasti.

Kun kaikki kehitys niin algoritmeissa, arkkitehtuureissa, teknologiassa kuin pilvissäkin tapahtuu samaan aikaan, kehityksestä perillä oleminen vaatii intensiivistä alan seuraamista. Muutamassa vuodessa hyvistä käytännöistä on tullut esihistoriallisia teknologisessa mielessä. Myös tämä opinnäyte tulee olemaan kuva siitä hetkestä, jota eletään vuonna 2021. Kahden vuoden päästä tämä työ tulee toimimaan vain yleiskuvana NLP-tehtävistä, sillä uudet mallit syrjäyttävät vanhat mallit sataprosenttisella varmuudella.

Tätä jatkuvassa myllerryksessä olevaa tekoälyn osa-aluetta, eli NLP:tä, voi olla vaikea lähestyä vain pintapuolisesti, sillä jo perusteiden ymmärtäminen vaatii melko laajan yleistietämyksen monelta osa-alueelta. Uudet mallit perustuvat poikkeuksetta erityyppisiin neuroverkkoihin ja keskivertoinsinöörille melko haastavaan matematiikkaan. Datan esikäsittely ja mallien opetus vaatii Python-ohjelmointikielen sekä siihen liittyvien kirjastojen ja ohjelmistokehyksien (framework) hallintaa. Tällaisia ovat esimerkiksi TensorFlow, PyTorch ja NumPy. Lisäksi tulee hallita mallien hallinta esimerkiksi jonkin pilvitarjoajan ympäristössä. Uusimpien Googlen ja Facebookin mallien taustalla on monia eri kerroksia, joilla on saatu luotua alan kehittyneimpiä malleja. Taustalla on valtava määrä tutkimusta, jotta mallit on saatu kehitettyä siihen missä ne tällä hetkellä ovat.

1.1 Työn rajaus

Tämä opinnäyte käsittelee NLP (Natural Language Processing) -kenttää vuonna 2021 ja erityisesti NLP-tehtävien implementointia AWS:n pilviympäristöön. NLP:n alaluokkana implementoinnissa keskitytään tekstin luokitteluun, sillä koko NLP-kentän tutkiminen yhdessä opinnäytetyössä ei olisi mielekästä.

Tässä opinnäytetyössä NLP:tä pyritään lähestymään käytännönläheisesti ”top to bottom” – lähestymistavalla. Aiheessa tosin liikutaan välillä myös sivusuunnassa sen useasta ulottuvuudesta johtuen. Ensin avataan mitä tehtäviä NLP:llä voidaan suorittaa. Sen jälkeen käydään läpi modernien mallien arkkitehtuuria, eli neuroverkkoja ja keskeisimpiä niihin liittyviä algoritmeja. Perusteiden ollessa hallussa, tutustutaan tarkemmin muutamaan keskeisimpään malliin ja vertaillaan niiden suorituskykyä sekä ominaisuuksia. Viimeiseksi tutustutaan AWS:n pilviympäristön tarjoamiin palveluihin ja tutkitaan millä kaikilla tavoilla valmista mallia voi ajaa heidän ympäristössään.

1.2 Tutkimuskysymykset ja tutkimusmenetelmät

Alan kehittyessä nopealla vauhdilla, ei kirjoissa oleva sisältö aina pysy perässä kehityksessä. Tästä syystä tässä työssä on käytetty poikkeuksellisen paljon lähteenä luotettavien sivustojen blogeja, vlogeja ja AWS:n materiaalia videomuodossa. Lähteinä on myös merkittävimpiä alan viimeaikaiseen kehitykseen vaikuttaneita artikkeleita.

Tutkimusotteena työssä on konstruktiivinen tutkimusote (Lukka 2001), jossa tavoite on luoda ymmärrys tämän hetken parhaista käytännöistä suorittaa NLP-tehtäviä AWS:n pilviympäristössä. NLP-tehtävistä tutkimus on rajattu tekstiaineiston luokitteluun, sillä useampien NLP-tehtävien arviointi tekisi työstä liian laajan. Pilviympäristöistä rajaus on tehty AWS:n tarjoamiin palveluihin, sillä useamman toimijan pilviratkaisujen arviointi laajentaisi työtä myös liikaa. Oletus on, että muut pilvipalveluiden tuottajat tarjoavat saman tyyppisiä palveluita kuin AWS. Tutkimuksessa erityiskiinnostus on toteuttaa tekstin luokittelua Serverless-arkkitehtuurilla, jota tässä työssä kutsutaan nimellä ServerlessAI.

Tutkimuskysymykset listattuna ovat seuraavat:

- Millä mallilla on paras tehdä suomen kielisen tekstin luokittelua tällä hetkellä?
- Kuinka Serverless-arkkitehtuurilla voidaan toteuttaa tekstin luokittelua?
- Mitä palveluita AWS tarjoaa tekstin luokitteluun?

2 NLP

Luonnollinen kieli niin puhuttuna kuin kirjoitettunakin on ihmeellistä ja ihmisen oppima taito kirjoittaa asioita muistiin on luonut hänelle supervoiman muihin maapallon eliöihin verrattuna. Paperille tai digitaaliseen muotoon tallennettu tieto on mahdollistanut muun muassa teknologisen kehityksen, kun ihmiset ympäri maailmaa ovat voineet jakaa tietoaan ja kehittää omaa osaamistaan muiden oppimien asioiden pohjalta. Kirjoitustaito kuitenkin on yllättävän tuore taito ihmiskunnan historiassa, sillä se on vasta 5000 vuotta vanha. Viimeisen viiden tuhannen vuoden aikana onkin tapahtunut todella paljon kehitystä, joka perimmäisenä edellytyksenä on kirjoitustaito. Ihmisen aivot ovat myös ihmeelliset, sillä ne pystyvät ymmärtämään kirjoitetusta tekstistä suuria asiakokonaisuuksia tai vaikkapa tunteita kirjoittajan pienien nyanssien pohjalta. Koneelle tekstin ymmärtäminen tavalla, jolla me sen ymmärrämme, ei ole aivan yksinkertainen tehtävä. Koneenkin pitää oppia säännöt, joiden avulla tulkita tekstiä ja jotka eivät aina ole aivan selkeitä. Sama pätee toki myös ihmisen oppimiseen, sillä ihmisenkin täytyy ymmärtää paljon ympäröivästä maailmasta, jotta pystyy esimerkiksi erottamaan ironian ja faktan poliittisessa satiirissa. Koneet ovat kuitenkin tulossa paremmiksi ja paremmiksi, sillä ihmiset kehittävät ahkerasti niitä osa-alueita, joita koneet tarvitsevat.

2.1 NLP:n määritelmä (Natural Language Processing)

Peter Jackson määrittelee NLP:n seuraavasti. NLP kuvaa sellaista tietoteknistä järjestelmää (laitteisto ja/tai ohjelmisto), jolla analysoidaan tai tuotetaan puhuttua tai kirjoitettua kieltä. ”Natural”, eli luonnollinen kieli tarkoittaa sellaista kieltä, jota ihminen tyypillisesti ymmärtää. Vastakohtia luonnolliselle kielelle ovat esimerkiksi matematiikka ja ohjelmointikielet. NLP:lle voidaan määritellä vielä alaluokaksi Natural Language Understanding (NLU), joka käsittelee nimensä mukaisesti kielen ymmärtämistä. (Jackson & Moulinier 2002, 2-3.)

2.1.1 Ongelmakenttä

NLP:ssä suuri haaste on saada kone ymmärtämään, mistä oikein on kysymys. Useilla sanoilla on monia merkityksiä ja myös sanan sijainti lauseessa vaikuttaa kielellisen viestin sisältöön merkittävästi. Esimerkiksi lause *‘John saw the man in the park with the telescope.’*, on koneelle erittäin hankala ymmärtää. Koneen on vaikea päätellä, kuuluuko kaukoputki Johnille, miehelle vai puistolle. (Jackson & Moulinier 2002, 3.)

NLP:n tehtävät kielen analysoinnissa voidaan jakaa kolmeen osaan (Jackson & Moulinier 2002, 5-7).

- Syntaksi, jolla tarkoitetaan sitä, miten kieli on muotoiltu. Kieliopin perusteella tunnistetaan lauserakenteita.
- Semantiikka, jolla tarkoitetaan sanojen eri merkitystä eri lauseyhteydessä.
- Pragmatiikka, jolla tarkoitetaan tilanteen vaikutusta merkitykseen. Esimerkiksi lause *”Olet minulle velkaa 5 euroa”*, vuoropuhelun yhteydessä on pyyntö, eikä toteamus.

2.1.2 NLP:n eri lähestymistavat

NLP:ssä on kaksi eri lähestymistapaa. Ensimmäinen tapa on lähestyä edellisessä luvussa esitettyjen ulottuvuuksien kautta ja luoda sääntöjä erilaisiin tilanteisiin. Esimerkiksi kielioppi voidaan määritellä sääntöjen kautta. Tätä ensimmäistä tapaa kutsutaan *symboliseksi* lähestymistavaksi. Tämä lähestymistapa kutsutaan termillä ”Good Old-Fashioned AI”. Symbolinen menetelmä lähestyykin NLP:tä top-down-menetelmällä, jossa ennalta tiedossa olevat säännöt ovat määrääviä. (Jackson & Moulinier 2002, 7.)

Toinen lähestymistapa lähestyy NLP:tä tilastollisesta näkökulmasta. Lähestymistavassa lähtökohtana on iso tekstiaineisto (korpus), jota analysoimalla voidaan ymmärtää säännöt kielen käyttäytymisestä. Tätä lähestymistapaa kutsutaan termillä empiirinen lähestymistapa. Empiirinen lähestymistapa onkin vastakohta symboliselle menetelmälle, eli NLP:tä lähestytään bottom-up-menetelmällä, sillä säännöt luodaan käsiteltävän korpuksen pohjalta. (Jackson & Moulinier 2002, 7.)

Lähestymisetapoja erottaa myös se, millä menetelmällä mallia voi päivittää, mikäli kieleen tulee uusia sääntöjä tai rakenteita. Symbolista mallia päivitetään muuttamalla tai luomalla uusia sääntöjä ja empiiristä mallia opettamalla mallia uudella datalla. (Jackson & Moulinier 2002, 7). Tällä hetkellä NLP-kenttä on kääntynyt voimakkaasti empiirisen lähestymistavan puolelle, sillä uusimmat ja tarkimmat mallit perustuvat pääsääntöisesti neuroverkkoihin, jotka on koulutettu valtavilla tietomäärillä.

2.2 NLP:n tehtäviä

Yksi tapa luokitella NLP:tä on määritellä tehtävät, joita NLP:llä voidaan suorittaa. Se mitä malli osaa tehdä, riippuu sen rakenteesta ja datasta, jota mallin opetukseen on käytetty. Modernit kielimallit kykenevät usein moniin eri tehtäviin, mutta malli pitää yleensä opettaa johonkin tiettyyn tehtävään. Jotta malli kykenee esimerkiksi kielen kääntämiseen, tulee opetusdatan olla monikielistä. Taulukossa 1 on esitetty tehtäviä, joita tässä työssä käsitellyillä malleilla voi suorittaa. Taulukossa ei ole esitetty kaikkia mahdollisia NLP-tehtäviä, sillä monia kielimalleja voi opettaa hyvin monipuolisesti aivan uudentyyppisiin tehtäviin, kuten esimerkiksi sentimenttianalyysiin.

Tehtävä/Malli	BERT	M-BERT	XLM-R	FinBERT	fastText	GPT-3
NMT (Neural Machine Translation) Konekäännös	-	x	x	-	-	x
Tekstin generointi	x	x	x	x	-	x
Luokittelu	x	x	x	x	x	x
NER (Named-entity recognition)	x	x	x	x	-	x
QnA	x	x	x	x	-	x
POS (Part of Speech tagging)	x	x	x	x	-	x

Taulukko 1. NLP-tehtäviä

2.2.1 Konekääntäminen

Konekääntäminen on tällä hetkellä varmasti yksi tunnetuimmista ja eniten käytetyistä kielimalleista, sillä Google Translatorista on tullut jo itsessään käsite monen arkikielessä. Käytännössä konekääntäminen tarkoittaa kielen kääntämistä lähdekielestä kohdekielelle. Konekääntämisen ja muiden monikielisten NLP-tehtävien tarkkuus ja nopeus ovat parantuneet viime vuosina huomattavasti uusien neuroverkkomallien myötä. Varsinkin Transformer-mallit ovat vieneet tätä osa-aluetta isoin harppauksin eteenpäin. Kääntämiseen tarvitaan kuitenkin vielä useita malleja, jotka on opetettu haluttujen kielien vastinpareilla, eli ei ole vielä olemassa yhtä mallia, joka hallitsee kaikki mahdolliset kielet. Esimerkiksi GPT-3

mallin opetusdatasta on 93 % englantia, joten suorituskyky ei ole täydellinen vähemmistökielillä (OpenAI 2020, 14).

2.2.2 Tekstin generointi

Tekstin generoinnilla tarkoitetaan sitä, että mallin avulla tuotetaan uutta tekstiä. Prosessi toimii niin, että mallille annetaan ensin jokin sana, jota malli jatkaa uudella tekstillä mallin mukaisella metodilla. Tekstin generointia voi tehdä monen tyyppisellä neuroverkkomallilla, ja yksinkertaisimpia on esimerkiksi tavallinen MLM, jossa verkko valitsee todennäköisimmän seuraavan sanan perustuen sen opetusdatassa olevien sanaparien frekvensseihin. Esimerkiksi jos opetusdatassa esiintyvät sanat ”koira” ja ”haukkuu” usein yhdessä, malli ehdottaa ”koira”-sanaa seuraavaksi sanaksi todennäköisesti sanaa ”haukkuu”.

Tekstin generoinnin viimeisintä teknologiaa edustaa OpenAI-yrityksen GPT-3-malli, joka osaa tekstin lisäksi kirjoittaa myös esimerkiksi ohjelmakoodia. Mallille voidaan antaa selkokieliset ohjeet, joiden pohjalta se voi kirjoittaa esimerkiksi koodin neuroverkkomallin koulutukseen. Se voi generoida myös valeuutisia, joita ihmiset eivät erota aidoista. Heidän testissään noin 200 sanan pituisista mallilla generoiduista artikkeleista 52 % prosenttia tunnistettiin keinotekoisiksi, eli lähes puolta mallilla luotuja uutisia ei ihminen erottanut aidoista uutisista. (OpenAI 2020, 25-26.) Heidän tutkimuksessaan kyky luoda tekstiä luokiteltiin kuitenkin vielä yhdeksi puutteeksi, joka on tulevaisuudessa jatkokehityksen alla (OpenAI 2020, 33).

2.2.3 Luokittelu

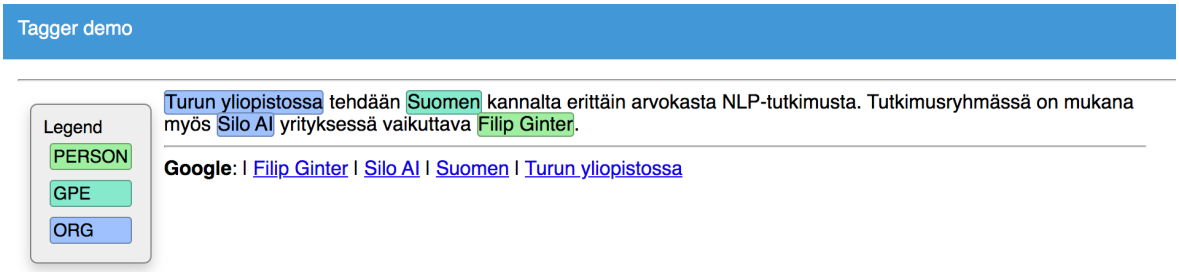
Tekstin luokittelu eri luokkiin dokumenttien sisällön perusteella on yksi useimmin käytetyistä NLP-tehtävistä. Luokittelua voidaan tehdä ohjatuilla tai ohjaamattomilla menetelmillä. Ohjatussa menetelmässä mallin koulutukseen käytettävä tietoaineisto on luokiteltu, eli koulutukseen käytettävien dokumenttien luokat ovat tiedossa. Esimerkiksi uutisdokumenttien luokittelussa tällaisia luokkia voisivat olla urheilu ja politiikka. Tämän luokittelun on yleensä tehnyt ihminen. Ohjatulla menetelmällä kouluttamalla saadaan luotua malli, joka osaa luokitella dokumentteja niihin luokkiin, joita koulutusvaiheessa on ollut tiedossa. Kun mallille annetaan syötteenä luokiteltavaksi uusi dokumentti, malli tulostaa todennäköisyydet kuulumisesta sen ymmärtämiin luokkiin. Esimerkiksi dokumentti, joka sisältää paljon urheiluun liittyviä sanoja, kuuluu suurella todennäköisyydellä urheiluun.

Kun tekstiaineistoa luokitellaan ohjaamattomasti, ei ennalta voida tietää miten malli dokumentteja luokittelee. Mallia koulutettaessa mallille annetaan parametrina haluttujen luokkien lukumäärä ja malli pyrkii koulutusvaiheessa erottelemaan datan eri ryhmiin. Tällaista

luokittelua kutsutaan klusteroinniksi. Ohjaamattomalla menetelmällä luokittelu ei ole välttämättä ihmisen näkökulmasta selkeä, sillä tietokoneen on mahdollista nähdä datassa useampia ulottuvuuksia kuin ihmisen.

2.2.4 NER (Named-entity recognition)

NER:in avulla voidaan tunnistaa tekstistä nimettyjä kokonaisuuksia, kuten esimerkiksi organisaatioita tai luonnollisia henkilöitä. Suomen kielellä NER:stä on viime aikoina saatu hyviä tuloksia uudella FinBERT-mallilla, kun sen hienosäätöön käytettiin UD Finnish-TDT-puupankkia (TurkuNLP 2020). NER-mallin demo löytyy osoitteesta http://86.50.253.19:8001/tagdemo/. Kuvassa 1 on esimerkki, kun demolla ajettiin lauseet ”*Turun yliopistossa tehdään Suomen kannalta erittäin arvokasta NLP-tutkimusta. Tutkimusryhmässä on mukana myös Silo AI yrityksessä vaikuttava Filip Ginter.”* Tulosteessa voidaan todeta mallin tunnistaneen nimetyt kokonaisuudet oikein.

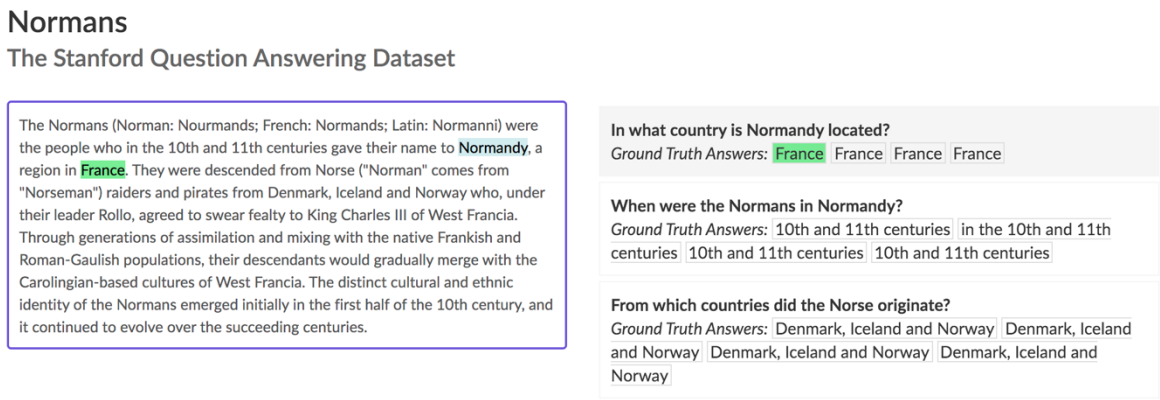


Kuva 1. NER-demon tuloste

2.2.5 QnA-kysymyksiin vastaaminen

Kysymyksiin vastaaminen tarkoittaa sitä, että malli osaa löytää vastauksen kysymykseen sille annetusta tekstiaineistosta. Malli tarvitsee siis aina sisältöä, josta etsiä vastauksia. Sellaisten vastuksien antaminen, jotka voisivat olla ihmisten antamia, on ollut monilla tavoitteena, mutta perinteisillä tekniikoilla ihmiskielen simuloiminen on osoittautunut haastavaksi. Perinteiset mallit ovat perustuneet lingvistisiin teorioihin ja kieliopin rakenteiden tunnistamiseen. Viimeisen kahden vuoden aikana tähän on kuitenkin tullut muutos, sillä uudet BERT-mallit, joiden taustalla on neuroverkko, ovat toimineet erittäin hyvin myös tässä tehtävässä. Nämä mallit mahdollistavat muun muassa älykkään kyselyn intrasta ja ”viisaampien” Chat-bottien luomisen. (Wongviboonsin 2020.)

Mallien hyödyntäminen on myös helpottunut huomattavasti. Alla on esimerkki Hugginfacen QnA-ratkaisusta heidän omalla API:llaan. Taustalla kyseisessä pipelinessa on BERT-malli, joka on hienosäädetty SQuAD (Stanford Question Answering Dataset)-tietoaineistolla. SQuAD on kielen ymmärtämisen tietoaineisto, jossa Wikipedia aineistoa on käsitelty niin, että luonnollisella kielellä esitettyihin kysymyksiin on osoitettu artikkeleista oikean vastauksen kohta. Kuvassa 2 on visualisointi SQuAD-tietoaineistosta.



Kuva 2. SQuAD visualisointi (rajpurkar.github.io 2020)

Malli on siis opetettu Wikipedian aineistolla, mutta silti se kykenee käsittelemään myös sellaista aineistoa, jota se ei ole ennen nähnyt. Se ymmärtää myös kielen rakennetta ja sanojen välisiä yhteyksiä. Malli on nimeltään ’bert-large-uncased-whole-word-masking-finetuned-squad’.

Alla oleva esimerkki on ajettu Chris McCormickin luoman Google Colab-Notebookin muokatussa versiossa. Notebook löytyy Githubista blob/master/Question_Answering_with_a_Fine_Tuned_BERT.ipynb

Mallille on annettu kuvan 3 mukainen tekstikappale erään yrityksen kotisivulta.

```
People tend to take the presence of light for granted. We take it seriously. By
creating the perfect solutions for our customers' needs, we offer solutions that
increase the comfort, functionality and safety of their applications. Our
passion is to design and manufacture lighting and interior systems for public
transportation vehicles and emergency lighting systems for buildings and cruise
ships.
```

Kuva 3. Syöte mallille

Sitten on esitetty kysymys ”*What are people thinking?*” ja mallin antama vastaus näkyy kuvassa 4.

```
Answer: "people tend to take the presence of light for granted"
```

Kuva 4. Mallin tuloste

Malli löytää hyvin kyseisessä tapauksessa oikean vastauksen, vaikka tekstissä ei esiinnykään sana ”*thinking*”, eli malli ymmärtää ”*tend to take”*-osan merkitsevän lähes samaa kuin ”*thinking”.* Kyseinen malli toimii vain englannin kielellä, sillä opetusaineisto on englanninkielistä. Samanlainen opetus on mahdollista myös esimerkiksi suomen kielellä opetetulle FinBERT-mallille. Suomen kielelle saman tyyppisen ominaisuuden voi tehdä myös kääntämällä aineistoin ja kysymyksen ensin englanniksi, ja ajaa vasta sen jälkeen englanninkielisen BERT-mallin läpi.

2.2.6 POS (Part of Speech tagging)

Part of speech tagging (POS) tarkoittaa lauseen sanaluokkien tunnistamista, eli tunnistetaan lauseesta esimerkiksi verbit, nominit ja konjunktiot. POS:lla on tärkeä rooli muiden tehtävien yhteydessä, sillä lauseen merkitys voi olla aivan toinen, kun sanat ovat eri järjestyksessä. Esimerkiksi lauseissa ”*I left the room*” ja *”Left of the room”* left-sana on ensimmäisessä lauseessa verbi ja toisessa nomini. Kuvassa 5 on esimerkki Pythonin nltk-kirjaston POS-operaatiosta. (Arumugam & Shanmugamani 2018, 38-39.)

```
>>> import nltk
>>> text1 = nltk.word_tokenize("I left the room")
>>> text2 = nltk.word_tokenize("Left of the room")
>>> nltk.pos_tag(text1,tagset='universal')
[('I', 'PRON'), ('left', 'VERB'), ('the', 'DET'), ('room', 'NOUN')]
>>> nltk.pos_tag(text2, tagset='universal')
[('Left', 'NOUN'), ('of', 'ADP'), ('the', 'DET'), ('room', 'NOUN')]
```

Kuva 5. POS-operaatio nltk-kirjastoa käyttäen

POS vaikuttaa merkittävästi muiden NLP-tehtävien tarkkuuteen, sillä esimerkiksi sentimenttianalyysissä sanajärjestys voi muuttaa merkityksen päinvastaiseksi.

2.3 Arkkitehtuurit

2.3.1 Algoritmit

Neuroverkot eivät pysty käsittelemään tekstiä tekstinä, vaan teksti pitää muuttaa ennen mallin luomista numeeriseen muotoon. Sanoja esitetään vektoreina, joiden ominaisuuksien mukaan voidaan arvioida esimerkiksi sanan kuulumista johonkin tiettyyn luokkaan, kuten karhu-sanan liittymisen eläimiin. Sanoja voi muuttaa vektoreiksi kolmella eri tavalla:

1. One hot encoding
   Tällä menetelmällä vektoreista tulee todella isoja, sillä 99,9 % vektorista on nollia.
2. Joka sanalla oma indeksi
   Tässä haittana on se, ettei lähekkäisten sanojen suhdetta voi arvioida.
3. Word Embeddings
   Tämä menetelmä on paras, sillä lähekkäisten sanojen vektorit ovat lähellä toisiaan. (Goodfellow ym. 2016)

Word Embeddings on tällä hetkellä käytetyin algoritmi sanojen vektoroimiseen. Vektorin ulottuvuus, jolla sanoja ilmaistaan, riippuu tietoaineiston koosta. Pienellä tietoaineistolla

ulottuvuus voi olla neljä, kun taas isolla se voi olla 1024 (TensorFlow.org 2020). Alla olevassa taulukossa 2 on esimerkki neliulotteisesta Word Embedding-matriisista. Taulukossa ruokaan liittyvien sanojen arvot ovat lähellä toisiaan. Word Embedding-matriisin voi luoda esimerkiksi Word2Vec-neuroverkolla, jota käsitellään seuraavaksi.

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

Taulukko 2. Esimerkki neliulotteisesta Word Embedding-taulukosta (Ng 2020)

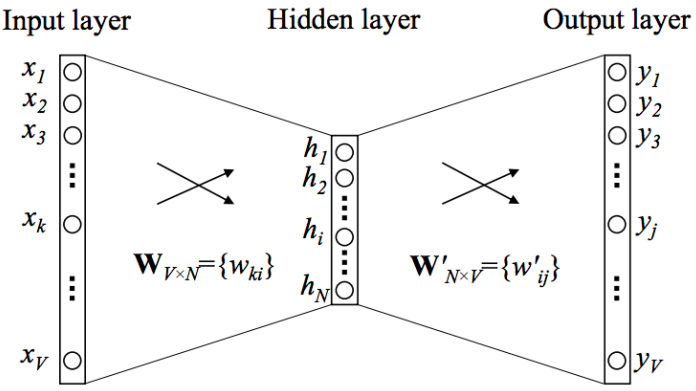
2.3.2 Word2Vec

Word2Vec on Tomas Mikolovin vuonna 2013 kehittämä menetelmä muuttaa sanat merkityksellisiksi vektoreiksi. Tällä on ollut suuri vaikutus NLP-kehitykseen. Sen perusajatus on luoda jokaisesta sanasta vektori, joka pitää sisällään tietoa sanan ominaisuuksista. Ominaisuuksiltaan saman tyyppisten sanojen vektorien kosinisamankaltaisuudet (engl. *cosine similarity*) ovat lähellä toisiaan (Mikolov ym. 2013). Esimerkiksi kissa ja koira ovat semanttisesti lähellä toisiaan, sillä molemmat ovat pieniä nisäkkäitä sekä ihmiset pitävät niitä lemmikkeinä. Kissan ja koiran samankaltaisuus voi olla esimerkiksi 0,8, kun vastaavasti koiran ja suden kosinisamankaltaisuus 0,6, vaikka koira biologisesti onkin lähempänä sutta.

Vektoreiden luominen lähtee siitä, että on olemassa korpus, josta halutaan muodostaa sanojen väliset riippuvuudet, eli luoda sanoille niiden ominaisuuksia kuvaavat vektorit. Korpuksen tulee olla riittävän iso, jotta jokainen sana esiintyy aineistossa riittävän monta kertaa. Esimerkiksi Mikolov käytti korpuksena Googlen uutisdataa, jossa sanoja oli yhteensä 6 miljardia. Seuraava vaihe on valita sanaston koko, eli kuinka monta sanaa mallin tulee ymmärtää. Voidaan valita esimerkiksi 10000 sanaa.

Seuraavaksi luodaan eteenpäin kytketty neuroverkko, jossa on yksi piilokerros (Kuva 6). Tulokerroksena verkolle tuodaan one-hot-encoding muotoon muutettu sanasto, eli esimerkiksi 10000 sanan sanastolle tulokerroksen koko olisi 10000 neuronia. Jokaista sanaa vastaa yksi tulokerroksen neuroni ja neuronin arvo voi olla 0 tai 1. Ulostulokerroksen pituus on myös saman pituinen kuin sanasto, eli 10000 neuronia. Piilokerroksen koko valitaan sen mukaan, kuinka montaa ominaisuutta sanoista halutaan tarkkailla. Mikolovin tutkimuksessa piilokerroksen pituus oli maksimissaan 600 neuronia.

Seuraavaksi verkko koulutetaan joko CBOW- tai Skip-gram–menetelmällä, joista Skip-gram antaa yleisesti ottaen hieman parempia tuloksia. Seuraavassa kuvassa on esimerkki verkon rakenteesta CBOW-menetelmällä. CBOW-menetelmässä periaate on pyrkiä päättelemään ympäröivien sanojen perusteella keskellä oleva sana. Esimerkiksi lause voisi olla ”Nainen ajoi autoa”, jolloin sanaa ”ajoi” yritettäisiin päätellä sanojen ”nainen” ja ”autoa” perusteella. Verkon tulokerrokselle syötetään ympäröivät sanat one-hot-encoding-muodossa, jossa sanojen ”nainen” ja ”autoa” neuronit ovat ykkösiä ja kaikki loput 9998 nollia. Ulostulokerroksella sanan ”ajoi” neuroni on yksi ja muut 9999 nollia. Kun verkkoa koulutetaan kyseisen muotoisilla lauseilla, verkko oppii päättelemään puuttuvan sanan.



Kuva 6. CBOW-verkon rakenne yhdellä kohdeluokalla (Xin 2016, 2)

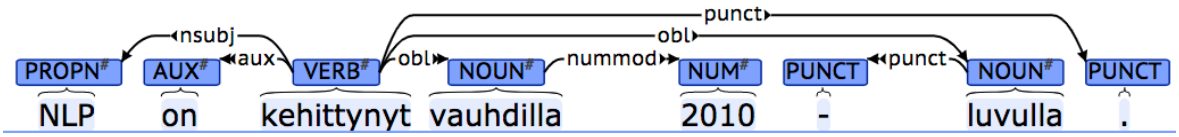
Word2Vec-mallissa puuttuvien sanojen päättely ei ole kuitenkaan lopullinen tavoite, vaan hienous piilee mallin koulutuksessa syntyneissä piilokerroksin painokertoimissa. Piilokerroksen painokertoimista saadaan koulutuksen jälkeen jokaiselle sanalle vektorit, joihin on tallennettu sanan ominaisuudet. Painokertoimista muodostuvaa matriisia kutsutaan termillä word embeddings, joka käytännössä tarkoittaa sanojen ominaisuuksien numeerista esitysmuotoa.

Turun NLP-tutkimusryhmän sivuilta löytyy demo, jossa word2Vecin toimintaa voi testata käytännössä. Demossa voi valita, millä datalla koulutettua mallia käytetään, jolloin vektorietäisyydet voi nähdä käytännössä sanaparien samankaltaisuuden arvosta. Esimerkiksi sanojan kissa ja koira etäisyys on eri malleilla erilainen. Demo löytyy osoitteesta http://bionlp-www.utu.fi/wv_demo/.

2.3.3 Esikäsittely

Dependency parsing

Dependency parsing, eli kielen rakenteen parsiminen on yksi olennainen vaihe tekstinkäsittelyssä. Koska lauseilla voi olla monia eri merkityksiä, tulee mallille syötettävä data olla sellaisessa muodossa, jossa kielen rakenne on selvillä. Alla on esimerkki Turun NLP-tutkimusryhmän tekemän Finnish-Neural – parserin tulosteesta. Parseri on opetettu suomalaisten puupankkien (TreeBank) aineistolla, joissa lauseiden jäsenet ja riippuvuudet ovat määritelty ihmisten toimesta. Kuten tulosteista selviää, parseri osaa tunnistaa lauseenjäsenet ja niiden väliset riippuvuudet. Riippuvuuksien lisäksi parseri myös lemmaa sanat, eli perusmuotoistaa ne. Parserin lemmausta kutsutaan sanakirjalemmaukseksi, sillä perusmuotoiset sanat ovat suomen kielen sanakirjan mukaisia. Toinen tapa perusmuotoistaa sanoja on stemmaus, jossa sanoja typistetään tiettyjen sääntöjen mukaisesti, kuten poistamalla suomen kielen päätteitä. Stemmaus on operaationa nopeampi, mutta ei niin tarkka kuin lemmaus. Esimerkkitulosteet on esitetty kuvissa 7 ja 8.



Kuva 7. Finnish-Neural-parserin tuloste. Demo sivulla http://bionlp-www.utu.fi/parser_demo/

```
# text = NLP on kehittynyt vauhdilla 2010 luvulla.
1   NLP          NLP        PROPN   N   Case=Nom|Number=Sing                                                    3   nsubj   _   _
2   on           olla       AUX     V   Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act        3   aux     _   _
3   kehittynyt   kehittyä   VERB    V   Case=Nom|Degree=Pos|Number=Sing|PartForm=Past|VerbForm=Part|Voice=Act  0   root    _   _
4   vauhdilla    vauhti     NOUN    N   Case=Ade|Number=Sing                                                   3   obl     _   _
5   2010-luvulla 2010#luku  NOUN    N   Case=Ade|Number=Sing                                                   3   obl     _   SpaceAfter=No
6   .   .        PUNCT      Punct   _                                                                          3   punct   _   SpacesAfter=\n
```

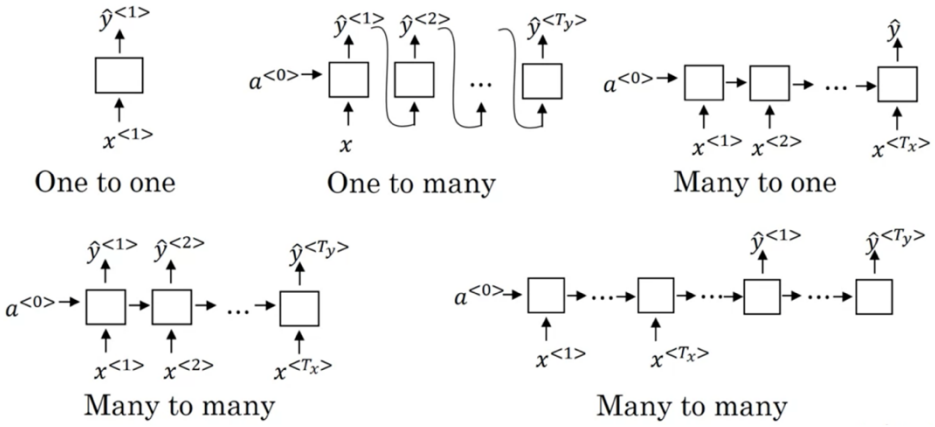
Kuva 8. Finnish-Neural-parserin tuloste CoNLL-U-muodossa.

Finnish-Neural-parseri perustuu nimensä mukaisesti myös neuroverkkoon. NLP-kentässä neuroverkot ovat tuoneet lisää suorituskykyä myös parsimiseen, joka auttaa tuomaan nopeutta myös muiden NLP-tehtävien suorittamiseen, sillä parsiminen on usein yksi vaihe NLP-tehtävän suorituksessa. Finnish-Neural - parserin kotisivu löytyy osoitteesta https://turkunlp.org/Turku-neural-parser-pipeline/. Suomen kielellä parhaita tuloksia viime aikoina on saatu FinBERT-malilla, joka kykenee tekemään myös muita tehtäviä kuin pelkkää parsimista (Virtanen ym. 2019, 11).

2.3.4 Neuroverkkoarkkitehtuurit

NLP:ssä tällä hetkellä parhaita tuloksia on saatu neuroverkkoihin perustuvien mallien avulla. Monet isot yritykset, kuten Google, kehittävät malleja jatkuvasti opettamalla niitä uudella datalla ja lisäksi kehittävät niihin liittyviä algoritmeja paremmiksi. Tällä hetkellä varsinkin suomen kieltä tunnistavat parhaiten Googlen kehittämään BERT(Bidirectional Encoder Representations from Transformers)-verkkoon pohjautuva FinBERT. Melkein yhtä hyvin suomea tunnistaa myös Facebookin kehittämä XLM-R. Näiden mallien periaatetta käsitellään tarkemmin seuraavissa kappaleissa. Google on kehittänyt BERT-mallistaan myös monikielisen M-BERT-mallin, joka ei kuitenkaan suorituskyvyssä yllä FinBERT- tai XLM-R-mallien tasolle.

NLP:ssä käytettävät neuroverkot ovat olleet viime aikoina usein rakenteeltaan RNN-tyyppisiä (Recurring Neural Network). Esimerkiksi puhetta tai tekstiä analysoitaessa data syötetään verkolle sekvenssinä, jossa sekvenssin edellinen tai myöhempi data vaikuttaa siihen, mikä on yksittäisen datapisteen merkitys. Esimerkiksi yksittäisen sanan merkitys lauseen keskellä riippuu sitä ympäröivistä sanoista. Kuvassa 9 on esitetty erityyppisiä NLP-tehtävissä käytettäviä RNN-verkkoja. Esimerkiksi lauseen sentimenttianalyysissä käytetään Many-to-one -verkkoa, jossa lauseen kaikki sanat muodostavat verkon sisääntulon ja ulostulo on arvio sentimentistä. Many-To-Many-rakennetta käytetään esimerkiksi kielen kääntämiseen, sillä sanojen määrä käännettävässä lauseessa on useimmiten eri lähdekielessä kuin kohdekielessä. (Ng 2020.)



Kuva 9. RNN-neuroverkkotyypit (Ng 2020)

BRNN

NLP-tehtävissä, joissa analysoidaan tekstimuotoista aineistoa, käytetään usein BRNN(Bi-directional Recurring Neural Network)-verkkoa. Verkko on nimensä mukaisesti kaksisuuntainen, jolloin tekstiä analysoitaessa analysoidaan lauseet myös takaperin. Esimerkiksi NER-tehtävässä esiintymän tunnistamiseen voi vaikuttaa oleellisesti se, mikä sana seuraa analysoitavaa sanaa. Alla olevissa lauseissa on mahdotonta sanoa, tarkoittaako Teddy ihmisen nimeä, jos lause ajetaan vain vasemmalta oikealle. Ajettaessa lause myös oikealta vasemmalle, voidaan päätellä Teddy-sanan olevan NER-esiintymä. (Ng 2020.)

"He said, Teddy Roosevelt..."

"He said, Teddy Bears..."

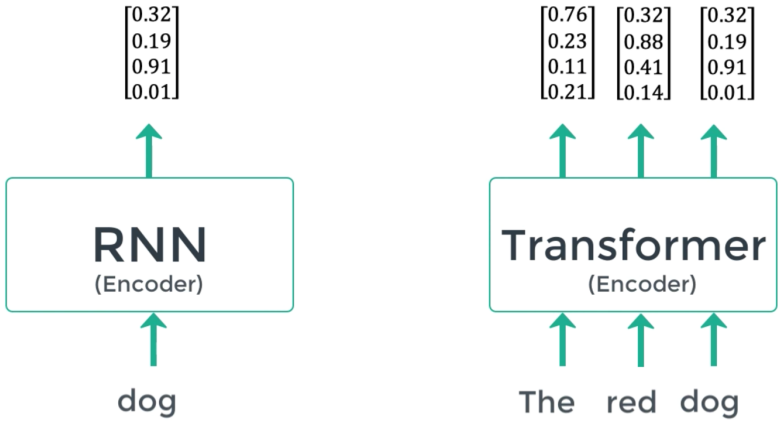
LSTM

NLP-tehtävissä yksi keskeinen menetelmä on LSTM (Long Short Term Memory). Sen avulla verkko voi ”muistaa”, mitä sanoja esimerkiksi yksittäisen lauseen käsittelyssä on aiemmin käsitelty. Esimerkiksi, jos luodaan mallilla tekstiä, voi verkko muistaa sen, onko lauseessa kyse yksiköstä vai monikosta ja sen perusteella valita oikean muodon. LSTM toi myös ratkaisun yksinkertaisissa RNN-verkoissa ilmenevään ongelmaan, eli häviäviin tai räjähtäviin gradientteihin. Kun yksinkertaisella verkolla käsitellään pitkiä tekstejä, gradientit voivat kasvaa tai pienentyä hallitsemattomasti, kun verkkoa ajetaan useita kertoja. Gradientti häviää pieneksi, jos luku on alle yhden, ja sitä kerrotaan useita kertoja itsellään. Käytännössä tämä tarkoittaa sitä, että varsinkin verkon ensimmäiset kerrokset oppivat huonosti (Olah 2020). LSTM:n toimintaperiaatteen ymmärtää usein parhaiten animaatioiden avulla. Animaatiot osoitteessa https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 avaavat toiminnan hyvin. Myös Olahin lähteenä käytetty blog-postaus on NLP-yhteisössä usein viitattu lähde.

LSTM on ollut hyvin suosittu vielä viime vuosina, mutta sen suosio on alkanut laskemaan transformer-arkkitehtuurin myötä, sillä LSTM ei toimi hyvin transfer learnin kanssa eikä myöskään kokonaan ratkaise gradienttiongelmaa (Dirac 2019). LSTM tekee myös verkosta raskaamman verrattuna yksinkertaiseen aktivointifunktioon, joka vaatii enemmän laskentatehoa. LSTM käsittelee sanoja rekursiivisesti, joten päättelyn nopeus riippuu käsiteltävän tekstin pituudesta, eli päättelyä ei voida suorittaa tehokkaasti rinnakkaisajona.

Transformer

Transformer on viimeisin suuri läpimurto NLP-kentässä. Se ratkaisee ongelmia, joita esimerkiksi LSTM-arkkitehtuurissa ilmeni. Suurin ero verrattuna tavalliseen RNN-verkkoon on mahdollisuus ajaa päättelyä rinnakkaisilla operaatioilla. Tavallisella verkolla sanoja käsitellään yksi kerrallaan, jolloin lauseen pituuden kasvaessa suoritusaika kasvaa ja gradientit joko häviävät tai räjähtävät. Transformer perustuu sen sijaan attention-malliin, jolloin laskentaa voidaan suorittaa rinnakkain. Transformer-malli onkin mullistanut NLP-kenttää sen jälkeen, kun Googlen tutkijat julkaisivat paperin ”Attention is all you need” vuonna 2017 (Vaswani ym. 2017). Kuvassa 10 on esimerkki tavallisesta RNN-verkosta verrattuna Transformer-verkkoon. Verkon suorittamat laskutoimitukset ovat käytännössä matriisilaskuja, joita moniytimisellä GPU:lla voidaan ajaa rinnakkain, joka nopeuttaa suoritusaikaa huomattavasti.

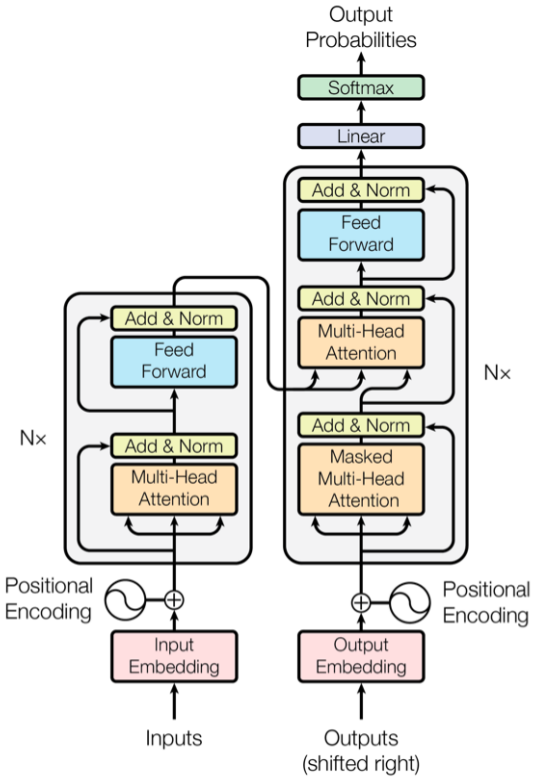


Kuva 10. RNN vs Transformer (Youtube 2020)

Transformer-malli on nopeuden lisäksi parempi myös siinä, että lauseiden pituus ei vaikuta niin paljon mallin tarkkuuteen. Alkuperäisessä tutkimuksessa mallia tutkittiin konekäännöstehtävissä, jossa käännettiin tekstiä englannista saksaan ja ranskaan. Konekäännöksen tarkkuuteen arviointiin käytettiin BLEU-indeksiä, joka kertoo muun muassa miten lausepituus vaikuttaa käännöksen tarkkuuteen. Transformer-malli päihitti vanhat mallit selvästi BLEU-pisteissä, eikä koulutukseenkaan kulunut niin paljon aikaa kuin vanhemmilla malleilla. (Vaswani ym. 2017.)

Ajatus, joka attention-ajatuksessa on mullistavaa, on käsitellä tekstiä samoin kuin ihminen. Esimerkiksi kirjaa käännettäessä, käännöstä tekevä ihminen ei tee niin, että lukisi ensin koko kirjan, sitten sulkisi sen ja aloittaisi kirjoittaa käännöstä. Ihminen tekee niin, että kiinnittää huomion (engl. *attention*) johonkin kohtaan, kääntää sen, ja sitten vasta kääntää seuraavan kohdan. Transformer-mallin attention toimii vähän samanlaisella periaatteella. Mallia opetettaessa malli oppii, kuinka paljon huomiota pitää kiinnittää mihinkin sanoihin. Perinteisessä RNN-mallissa esimerkiksi painoarvo riippuu enemmän siitä, mikä etäisyys sanoilla on lauseen sisällä, joten lauseen viimeisen ja ensimmäisen sanan riippuvuus toisiinsa on pieni. Transformer-mallissa käytetty attention-mekanismi riippuu vastaavasti siitä, mikä tarkasteltavien sanojen kosinietäisyys on.

Alkuperäisessä tutkimuksessa mallia käytettiin konekääntämiseen, eli tehtävä oli niin sanottu sekvenssistä sekvenssiin tehtävä. Lähdekielen lause on sekvenssi ja käännetty teksti kohdekielelle on sekvenssi. Tulosekvenssin muuttamiseen kohdesekvenssiksi mallissa käytetään enkooderi-dekooderi-arkkitehtuuria (engl. *encoder decoder*). Ensimmäisessä vaiheessa enkooderi muuttaa lähdekielen lauseen embedding-matriisiksi ja toisessa vaiheessa dekooderi muuttaa embedding-matriisin kohdekielen lauseeksi. Kuvassa 11 enkooderi on vasemmalla puolella ja dekooderi oikealla.



Kuva 11. Transformer-mallin arkkitehtuuri (Vaswani ym. 2017, 3)

Transformer-arkkitehtuuri on melko monimutkainen ja menetelmän syvällinen ymmärtäminen vaatii alkuperäisen artikkelin lukemisen lisäksi perehtymistä lineaarialgebraan sekä luultavasti muihin aiheeseen liittyvään materiaaliin esimerkiksi Youtubessa. Tähän työhön ymmärrystä saatiin Chric McCormickin Youtube-luennoista, joissa myös hän itse yrittää ymmärtää, miten arkkitehtuuri toimii. Ymmärtäminen helpottuu huomattavasti, mikäli yliopistotason lineaarialgebra on hyvin hallussa.

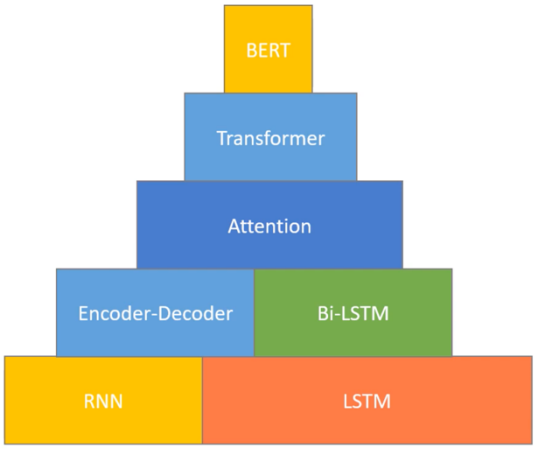
Alkuperäisen tutkimuksen jälkeen Transformer-mallilla on saatu hyviä tuloksia myös muissa NLP-tehtävissä. Muissa tehtävissä mallista käytetään ainoastaan enkooderia, sillä malli multihead attention ymmärtää erittäin hyvin tekstin kontekstin verrattuna edellisen sukupolven malleihin. Transformer-mallit eivät kuitenkaan sovi ihan joka sovellukseen, sillä niiden koko kasvaa nopeasti todella suureksi. Pienetkin mallit voivat sisältää kymmeniä miljoonia parametreja. Tästä johtuen ne vaativat paljon muistia, joten esimerkiksi Serverless-toteutus AWS:n lambda funktiolla ei onnistu.

2.4 NLP-malleja

Erilaisia malleja NLP-tehtävien suorittamiseen on olemassa valtavasti. Tutkimusryhmät ja AI-kehitykseen erikoistuneet yritykset hienosäätävät neuroverkkojen rakenteita sopimaan paremmin erilaisiin tehtäviin, jolloin erilaisia malleja syntyy paljon. Uusimpien mallien arkkitehtuuri on usein Transformer-pohjainen, mutta myös perinteisemmillä arkkitehtuureilla on vielä käyttökohteita, mikäli esimerkiksi laiteresurssit rajoittavat hienostuneempien mallien käytön. Tässä työssä käydään läpi muutama oleellisin malli, jotka ovat vuonna 2021 kehityksen kärjessä varsinkin suomen kielen näkökulmasta.

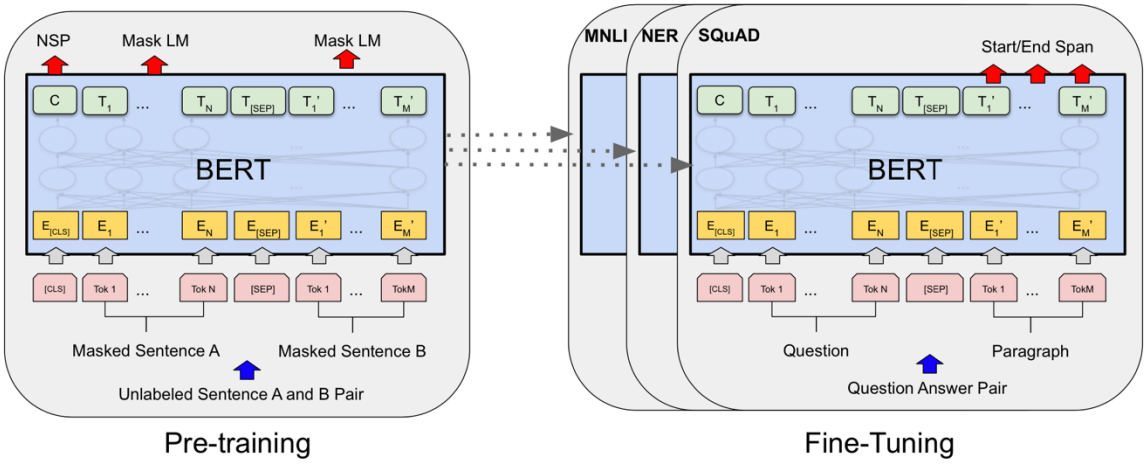
2.4.1 BERT

BERT on Googlen tutkimusryhmän vuonna 2018 julkaisema malli, joka vei NLP-kehitystä huimasti eteenpäin. BERT on koulutettu ohjaamattomalla menetelmällä ja opetukseen on käytetty englanninkielistä Wikipediaa (2500 miljoonaa sanaa) ja BookCorpusta (800 miljoonaa sanaa) (Devlin ym. 2019, 5). BERT perustuu transformer-arkkitehtuuriin, eli mallissa on taustalla artikkelissa *Attention is all you need* esiteltyjä transformer-komponentteja. Tutkijat käyttivät attention-mekanismia ja loivat uuden mallin perustuen transformer-arkkitehtuuriin. Malli käyttää kaksisuuntaisia enkoodereita, eli sanojen riippuvuuksia lasketaan edeltäviin ja seuraaviin sanoihin. BERT-mallissa transformerista otettiin käyttöön vain enkooderi ja malliin lisättiin useampi head. Alkuperäisessä mallissa oli 6 headia, mutta BERT-mallissa (base) niitä on 12. Useampi head mahdollistaa huomion kiinnittämisen useampiin asioihin. (Devlin ym. 2019) Kuvasta 12 nähdään BERT-mallin taustalla oleva arkkitehtuuri.



Kuva 12. BERT-mallin rakenne (McCormick 2020)

BERT-malli on esikoulutettu (engl. *Pre-trained*) MLM-tyyppiseksi malliksi (engl. *Masked Language Model*), eli korpusta on koulutettu niin, että malli pystyy päättelemään tekstistä puuttuvia sanoja ja seuraavia lauseita (engl. *Next Sentence Prediction*). Esikoulutettua mallia on kuitenkin mahdollista hienosäätää (engl. *fine-tuning*) muihin NLP-tehtäviin, kuten tekstin luokitteluun. Hienosäädössä malliin lisätään ulostulokerros, jonka rakenne on kyseiseen tehtävään sopiva ja koulutetaan sovelluskohtaisella datalla. Kuvassa 13 näkyy esimerkki hienosäätötehtävistä. (Devlin ym. 2019, 3.)

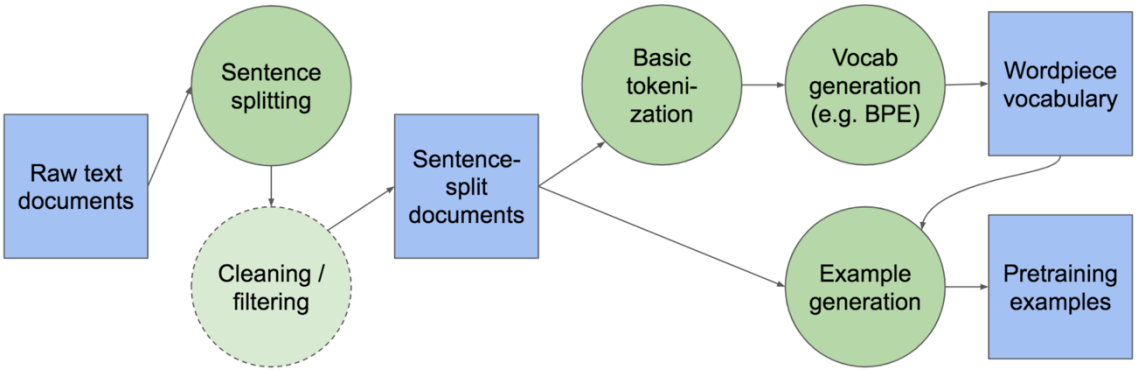


Kuva 13. BERT-mallin hienosäätö (Devlin ym. 2019, 3)

BERT-malli päihitti ilmestyessään sen hetkiset parhaat mallit selvällä marginaalilla. Tutkimuksessa myös selvisi, että mallin esikouluttaminen massiivisella korpuksella parantaa mallin pohjalta johonkin muuhun NLP-tehtävään hienosäädettyjen mallien tarkkuutta, vaikka hienosäätöön käytetty tietoaineisto olisi pieni (Devlin ym. 2019, 9). BERT muutti NLP-kenttää radikaalisti ja malliin pohjalta onkin kehitetty monia uusia malleja, joiden arkkitehtuurit on optimoitu eri NLP-tehtävien tarpeet huomioiden. Mallin arkkitehtuuria on myös käytetty uusien yksikielisten mallien koulutukseen. Yksi tällainen on seuraavassa luvussa esiteltävä suomen kielelle optimoitu FinBERT.

2.4.2 FinBERT

FinBERT on yksikielinen neuroverkkomalli, joka on koulutettu BERT-verkon pohjalta suomen kielelle. Malli on opetettu lähtemällä nollasta, eli opetukseen ei ole käytetty Transfer Learning-menetelmää. Mallin opetukseen käytettiin dataa suomen kielisistä uutisista (YLE ja STT), suomi24 keskustelusta ja lisäksi internetin suomen kielistä crawling-dataa. Data esikäsiteltiin tokenisoimalla ja ajamalla se Turku Neural Parser Pipelinen läpi. Dataa suodatettiin vielä tämän jälkeenkin viidessä eri vaiheessa, muun muassa poistamalla lauseita, jotka olivat liian lyhyitä, sisälsivät liikaa numeroita tai olivat konekäännettyjä. Tämä malli on huomattavasti tarkempi kuin mBERT-malli suomen kielellä (Virtanen ym. 2019). Kuvassa 14 on esitetty prosessi, jota FinBERT-mallin koulutuksessa on käytetty.

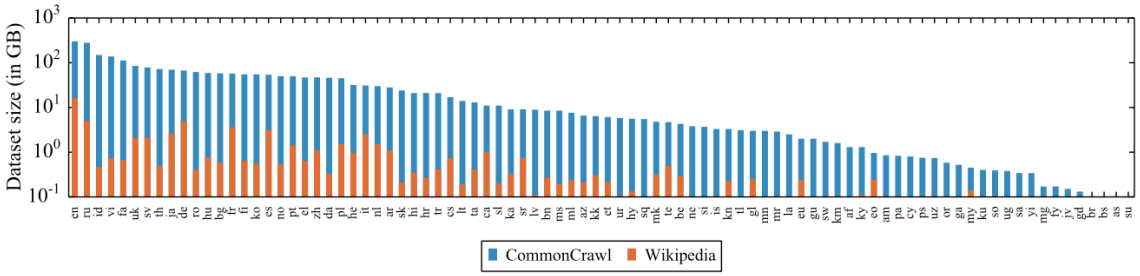


Kuva 14. BERT-koulutuksen prosessi (Virtanen ym. 2020, 17)

FinBERT:in koulutukseen käytettiin Kajaanissa sijaitsevaa Puhti supertietokonetta. Mallin koulutuksessa käytettiin kahdeksaa kappaletta Nvidia V100 GPU-prosessoreita. Koulutus kesti noin 12 päivää. FinBERT malli on julkaistu avoimen lähdekoodin lisenssillä ja tietovarasto (engl. *repository*) löytyy githubista osoitteesta https://github.com/TurkuNLP/FinBERT.

2.4.3 XLM-R

XLM-R on monikielinen malli, jolla voi luokitella monikielistä tekstiaineistoa, tehdä sekvenssi merkkaamista (engl. *labeling*) ja kysymys-vastaus sovelluksia. XLM-kirjainlyhenne tulee sanoista Cross Language Model ja jälkiliite 'R' tulee sanasta RoBERTa. Malli pystyy käsittelemään 100 erikielistä tekstiä, joita sille on käytetty opetusaineistona. Malli on opetettu ohjaamattomalla menetelmällä. Tutkimusraportin mukaan sen yleinen tarkkuus on 14,6 % tarkempi verrattuna monikieliseen mBERT-verkkoon ja marginaalikielillä sen tarkkuus on jopa 23 % mBERT:tiä parempi. Opetusaineistona mallille on käytetty puhdistettua CommonCrawls-dataa, joka tekee siitä tarkemman kuin muut mallit, joiden opetukseen on käytetty pääasiassa Wikipedian aineistoa. Wikipediassa marginaalikielien prosentuaalinen osuus aiheuttaa epätarkkuutta marginaalikielisen tekstin tunnistuksessa. Kuvassa 15 on esitetty tietoaineistojen koot eri kielille. Tutkimuksessa todettiin, että tarvittavan tekstiaineiston määrä on vähintään muutaman sata megatavua jokaista kieltä kohden, kun käytetään ohjaamatonta menetelmää. (Conneau & Khandelwal 2020, 1-3.)



Kuva 15. CommonCrawl ja Wikipedian datamäärät (Conneau ja Khandelwal 2020, 3)

XLM:ssä käytettiin sanojen tokenisointiin Sentence Piece Model (SPM)-menetelmää kaikille kielille, sillä tutkimuksessa ei havaittu eroa tarkkuudessa, kun verrattiin tuloksia kielikohtaisella tokenisoijalla tehtyihin tuloksiin. Tämä yksinkertaisti operaatiota huomattavasti. (Conneau & Khandelwal 2020, 6.)

2.4.4 FastText

FastText on Facebookin AI-tutkimusryhmän luoma NLP-kirjasto, joka on tehty toimimaan heikkotehoisilla laitteilla, kuten esimerkiksi tavallisella kotitietokoneen CPU:lla tai mobiililaitteella. Itseasiassa FastText ei toimi tällä hetkellä lainkaan GPU:lla. Kirjastolla voi tehdä muun muassa tekstin luokittelua, luoda embedding-vektoreita ja tunnistaa tekstin kielen. Kirjastoa voi ajaa joko Pythonilla tai ajaa omana ohjelmanaan. Siitä on tehty myös WebAssembly-moduuli, jolloin sitä voi ajaa myös selaimessa Javascriptillä. (Facebook Inc 2020.)

FastText-mallin keveys tekee siitä varteenotettavan vaihtoehdon myös pilviympäristössä ajettavaksi. Sen avulla on muun muassa mahdollista toteuttaa NLP-sovellus Serverless-arkkitehtuurilla, jolloin pilvessä ei tarvitse olla jatkuvasti päällä olevaa instanssia. Tästä voi koitua huomattavia kustannussäästöjä, sillä varsinkin tekoälymallien ajamiseen erityisesti tarkoitetut virtuaalikoneet ovat minuuttihinnaltaan melko kalliita. Serverless-ratkaisussa kustannukset muodostuvat todellisen käytön mukaan, joten varsinkin harvakseltaan kutsuttava endpoint, joka kuitenkin pitää olla aina saavutettavissa, voi olla järkevää toteuttaa Serverless-tyyppisenä.

2.4.5 GPT-3

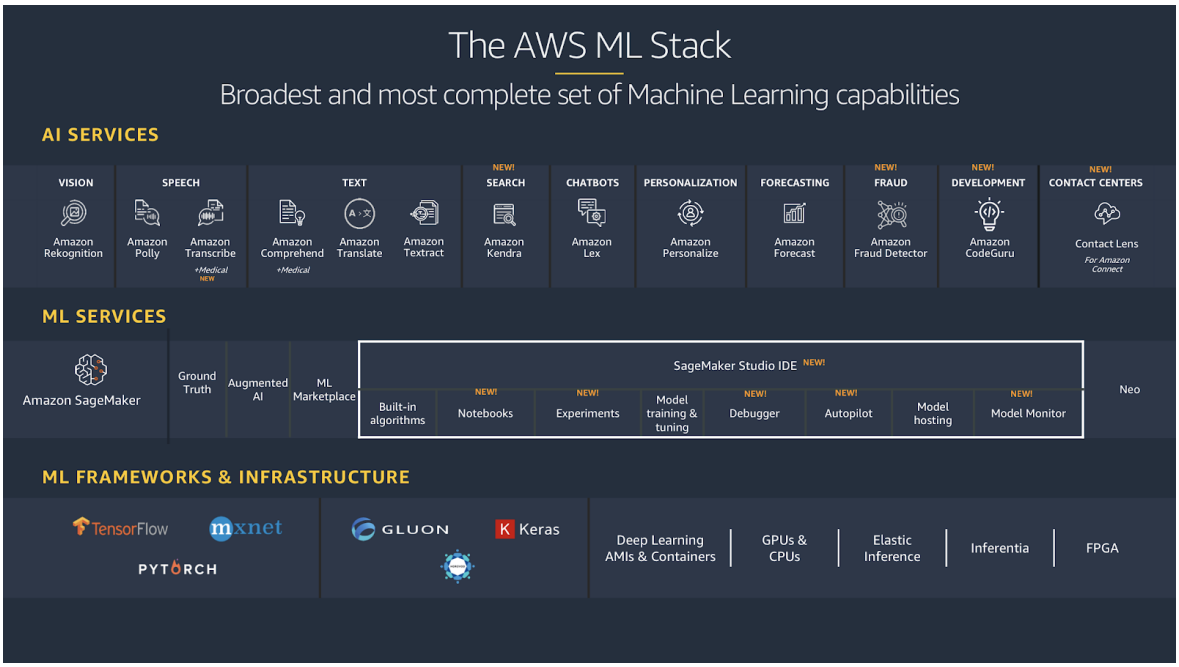
GPT-3 on OpenAI-yrityksen kehittämä massiivinen kielimalli, jonka ympärillä on viime aikoina ollut paljon hypeä. Malli on valtava verrattuna esimerkiksi BERT-malleihin ja isoin versio sisältääkin 175 miljardia parametria (OpenAI 2020). Verkon arkkitehtuuri perustuu Transformer-malliin, kuten BERT-malleissakin. Malli eroaa tavallisista kielimalleista siinä, että se pystyy myös generoimaan muun muassa koodia. Malli on herättänyt ihmisissä jopa pelkoa, sillä se pystyy tuottamaan jo niin hyvää tekstiä, että sitä voidaan käyttää helposti myös laittomiin tarkoituksiin. Laiton tarkoitus voisi olla esimerkiksi väärennettyjen tuotearvostelujen luominen verkkokauppoihin. Tunnettu kongitiotieteilijä David Chalmers on sanonut GPT-3-mallin olevan yksi mielenkiintoisimmista ja tärkeimmistä ikinä luoduista AI-sovelluksista (Chalmers 2020).

Yrityksen taustalla toimii myös Elon Musk, joka oli yksi perustajista, mutta tällä hetkellä enää rahoittajan roolissa. Yritys aloitti voittoa tavoittelemattomana yrityksenä, mutta nyt se on muuttunut voittoa tavoittelevaksi. Tässä työssä mallia ei ollut mahdollista testata, sillä GPT-3:n rajapinta ei ole vielä julkisessa käytössä. Malli on vaiheessa julkinen beta, mutta käyttäjämäärä on hyvin rajattu. Malli on tosiaan niin valtavan kokoinen, että sitä on luultavasti pakko käyttää API-rajapinnan kautta, sillä sen vaatimat resurssit ovat niin suuret. Monella ei olisi mahdollisuutta hankkia mallin ajoon vaadittua laitteistoa, vaikka

lähdekoodi olisi avoin. OpenAI perustelee suljetun koodin tarpeellisuutta myös sillä, että mallia ei ole niin helppo käyttää laittomiin tarkoituksiin, kun sitä käytetään API:n kautta. He voivat toimia moderaattoreina, mikäli mallilla aletaan luomaan esimerkiksi valeuutisia. Edellisen sukupolven mallia, eli GPT-2-mallia käytettiin juuri tuolla tavalla hyväksi. (OpenAI 2020.)

3 AWS:N TEKOÄLYPAKETTI

AWS on tällä hetkellä kolmen kärjessä AI-kehityksessä ja he kehittävät aiheeseen liittyvää tekoälypakettiaan (AI/ML-stack) jatkuvasti. Kuvasta 16 nähdään tämänhetkinen tarjonta, mutta kuva on todennäköisesti vanhentunut jo puolen vuoden päästä tämän opinnäytetyön kirjoitushetkestä. Perusjaottelu tarjoamassa pysyy kuitenkin varmasti samanlaisena, eli jako AI-palveluihin, koneoppimispalveluihin ja tuettuihin kehitysympäristöihin.



Kuva 16. AWS:n tekoälypaketti (AWS 2020, 2)

3.1 AI-Palvelut (AI Services)

AWS:n AI-palvelut ovat palveluita, joita on mahdollista hyödyntää ilman syvempää ymmärrystä tekoälystä. Palvelut tarjoavat rajapinnan, jonka kautta voi tehdä esimerkiksi objektien tunnistusta kuvista tai tekstin tunnistusta pdf-dokumenteista. Osassa palveluista on myös käyttöliittymä, jolloin palvelu voidaan luokitella verkkosovellukseksi.

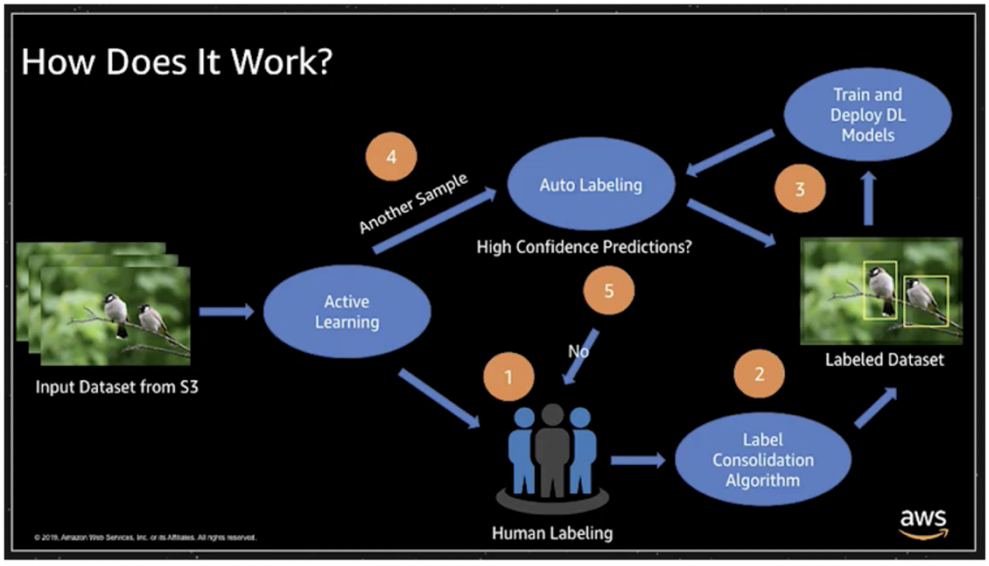
Rajapintasovellus on esimerkiksi tekstintunnistuspalvelu Amazon Textract, jossa lähetetään dokumentti rajapintaan ja rajapinta antaa vastauksena JSON-muotoisen vastauksen, jota voidaan sitten edelleen hyödyntää omiin tarpeisiinsa. Käyttöliittymän tarjoava palvelu on vastaavasti esimerkiksi Contact Lens, jota voidaan hyödyntää esimerkiksi asiakaspalvelun laadun arviointiin. Palvelusta saadaan helposti näyttäviä graafisia esityksiä asiakaspalvelun tapahtumista.

Laskutustapa palveluissa vaihtelee. On transaktio- ja tuntiperusteisia, sekä näiden yhdistelmiä. Tässä luvussa on käyty tarkemmin läpi NLP-tehtävistä erityisesti tekstin käsittelyyn liittyviä palveluita. Palveluilla on mahdollista toteuttaa hyvin monenlaisia NLP-tehtäviä.

3.1.1 Ground Truth – datan luokittelu

AWS:llä on palvelu Ground Truth, jonka avulla voidaan luokitella dataa automaattisesti. Ajatus palvelussa on se, että automatiikka luokittelee osan datasta, mutta se osa minkä tarkkuus ei ole tarpeeksi hyvä, menee luokiteltavaksi ihmisille. Kuviossa 1 on esitetty prosessi, jolla luokittelu toimii.

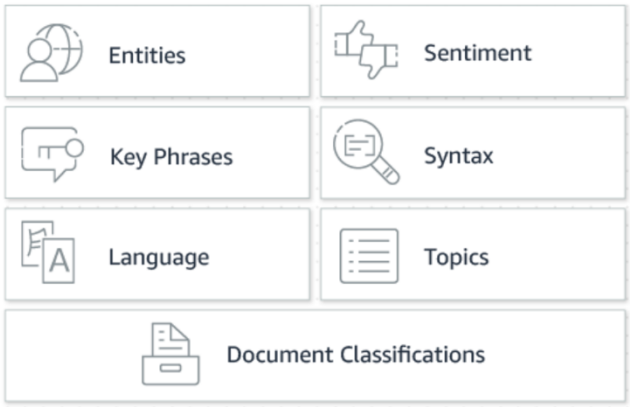
1. Ensimmäisessä vaiheessa palvelu lähettää satunnaisen määrän dataa luokiteltavaksi ihmisille.
2. Algoritmi yhdistää eri ihmisten tekemät luokittelut lopulta yhdeksi luokaksi. (Consolidated Annotation)
3. Tästä syntyy oma pieni tietoaineisto, jota käytetään yhdessä Sagemakerin tietoaineiston kanssa koneoppimismallin luomiseen.
4. Vaiheessa 4 palvelu tekee luokittelun lopulle raakadatalle. Mikäli tarkkuus on yli kynnysarvon, luokitellaan data automaattisesti.
5. Mikäli tarkkuus on alle kynnysarvon, lähetetään data ihmisille luokiteltavaksi.
6. Ihmisten ja palvelun luokittelemaa dataa käytetään edelleen mallin uudelleenopettamiseen, jolloin malli paranee sitä mukaa, mitä enemmän dataa kertyy.



Kuvio 1. Ground Truth-prosessi (Badr 2019)

3.1.2 Comprehend

Comprehend on hyvin monipuolinen NLP-palvelu, jolla on mahdollista suorittaa monia tekstiaineistoin käsittelyyn liittyviä tehtäviä. Kuvasta 17 näkyy AWS:n jaottelu eri tehtävistä. Tekstianalyysin tuettuja kieliä ovat tällä hetkellä englanti, ranska, saksa, italia, portugali, espanja, japani, korea, hindi, arabia ja kiina. Tuettujen kielien määrä elää jatkuvasti, eikä AWS:n oma dokumentaatiokaan välttämättä ole aina ajan tasalla. Kirjoitushetkellä AWS:n nettisivulla kielien määrä oli pienempi kuin virallisessa dokumentaatiossa. (AWS 2020) Palvelulla on mahdollista myös kääntää tekstiä kielestä toiseen, joten analyysia voidaan pienen välivaiheen avulla tehdä useammille kielille.



Kuva 17. Comprehend tehtävät (AWS 2020)

Comprehend toimii kahdella tavalla, joko reaaliaikaisena API-rajapintana esimerkiksi sentimentin tunnistuksessa tai massa-ajona dokumenttien luokittelussa.

Tehtävä	Tuetut kielet
Kielen tunnistaminen	Useita kieliä (myös suomi)
NER	Tuetut kielet (tekstianalyysi)
Avainfraasien tunnistus	Tuetut kielet (tekstianalyysi)
Henkilön personoivan tiedon tunnistus	Englanti
Sentimentin tunnistus	Tuetut kielet (tekstianalyysi)
Syntaksin analysointi	englanti, ranska, saksa, italia, portugali ja espanja
Luokkien tunnistus (Topic Modeling)	Kieliriippumaton
Kustomoitu luokittelu (oman luokittelun pohjalta)	englanti, ranska, saksa, italia, portugali ja espanja
Kustomoitu NER	englanti, ranska, saksa, italia, portugali ja espanja

Kuva 18. Tehtävät eri kielillä (AWS 2020, 4-5)

Suomen kielen näkökulmasta palveluista mielenkiintoinen on aiheen tunnistus, jonka luvataan toimivan millä kielellä tahansa. Aiheen tunnistus perustuu LDA:han ja mahdollisten aiheiden määrä on maksimissaan 100. Dokumentteja, joissa jokaisessa on vähintään kolme lausetta, tulisi olla minimissään 1000 kappaletta.

Luokittelun testaukseen käytettiin YLE:n suomen kielistä tietoaineistoa, jossa jokainen uutinen on yhdellä rivillä tekstitiedostossa. Rivejä tiedostossa oli yhteensä 1000. Alkuperäisessä tietoaineistossa jokainen uutinen oli luokiteltu johonkin luokkaan, mutta Comprehendin Topic Modeling-testiä varten luokkien nimet poistettiin. Luokittelutyö kesti 27 minuuttia ja tulokset olivat hieman hämmentäviä. Comprehend löysi luokkia, mutta mallin luomien luokkien mahdollisia otsikoita oli vaikea päätellä. Merkittävimpien sanojen painokertoimet olivat hyvin pieniä, joten malli ei ollut kovin vakuuttava suomen kielellä. Seuraavassa taulukossa on esimerkkinä mallin kaksi ensimmäistä luokkaa. Sitä seuraavassa taulukossa on esimerkkinä dokumenttien kuulumisista ei luokkiin. Malli löysi luokan todella harvalle dokumentille. Taulukosta näkyy, että ensimmäinen dokumentti, jolle luokka määritettiin, oli rivillä viisi. Seuraava dokumentti on rivillä 40.

Yhteenvetona Comprehend-palvelusta voi sanoa, että suomen kielellä luokittelu ei näytä toimivalta. Palvelu on kuitenkin todella helppo käyttää, joten englanninkielisissä dokumenteissa luokittelu on varmasti todella hyvä valinta Serverless-tyyppiseen arkkitehtuuriin. Luokittelun hinnoittelu perustuu luokiteltavan aineiston tiedostokokoon. Ensimmäinen 100MB maksaa noin yhden euron ja sen ylimenevät megatavut noin 0,004€/MB.

topic	term	weight
0	suomen	0.025020357
0	suomi	0.0056426083
0	suomessa	0.0056326604
0	mm	0.0046291454
0	itä	0.0042491932
0	keski	0.0037217445
0	sanoo	0.0031074882
0	varsinais	0.0024670418
0	euroopan	0.0025693288
0	yliopiston	0.0023639966
1	poliisi	0.029746681
1	poliisin	0.013487081
1	mies	0.017474864
1	kiinni	0.0069286497
1	tutkii	0.006308585
1	miestä	0.005734961
1	epäillään	0.0054833204
1	miehen	0.007805202
1	auton	0.0044155144
1	auto	0.004321679

Kuva 19. Comprehend Topic Modeling, kaksi ensimmäistä luokkaa

docname	topic	proportion
yle-train-1000-nolabel.txt:5	0	0.791717
yle-train-1000-nolabel.txt:5	3	0.147581
yle-train-1000-nolabel.txt:5	7	0.060702
yle-train-1000-nolabel.txt:40	2	1.0
yle-train-1000-nolabel.txt:75	4	0.444904
yle-train-1000-nolabel.txt:75	0	0.284772
yle-train-1000-nolabel.txt:75	7	0.270324
yle-train-1000-nolabel.txt:110	7	1.0
yle-train-1000-nolabel.txt:145	7	1.0
yle-train-1000-nolabel.txt:180	1	0.71157
yle-train-1000-nolabel.txt:180	4	0.099506
yle-train-1000-nolabel.txt:180	0	0.096409
yle-train-1000-nolabel.txt:180	7	0.092515
yle-train-1000-nolabel.txt:215	1	1.0

Kuva 20. Comprehend Topic Modeling, dokumenttien luokkia

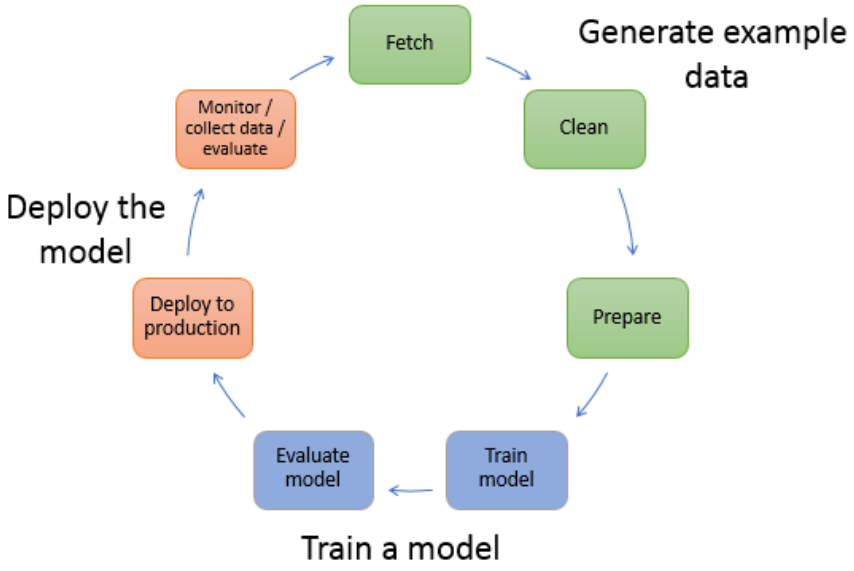
3.1.3 Translate

Translate on palveluna erittäin helposti ymmärrettävä, eli nimensä mukaisesti se tekee konekäännöksen sille annetusta tekstistä. Tällä hetkellä Translate tukee 55 eri kieltä, mukaan lukien suomen kieli. Kääntäminen voidaan tehdä kahdella tavalla, joko reaaliaikaisena tai massa-ajona. Reaaliaikaisessa käännöksessä Translate-palveluun tehdään JSON-muotoinen pyyntö. Palvelun vastauksessa on käännetty teksti. Massa-ajossa käännettävät dokumentit viedään ensin S3:ssa olevaan kansioon ja palvelulle kerrotaan näiden dokumenttien sijainti. Sen jälkeen palvelu suorittaa dokumenttien kääntämisen ja tallentaa käännetyt versiot toiseen S3-kansioon. Translate-palvelussa voidaan määrittää myös automaattinen lähdekielen tunnistus. Taustalla palvelu käyttää Comprehend-palvelua tunnistukseen, josta peritään sen hinnoittelun mukainen maksu. Tämän tyyppinen palveluiden yhdistely on hyvin tyypillistä pilven eri palveluissa, sillä ”konepellin alla” AWS yhdistelee usein eri palveluita ja muotoilee niistä asiakkaille uusia palveluita eri sovellusten tarpeiden mukaan.

Verkkosivuillaan AWS kertoo Translaten perustuvan syviin neuroverkkoihin, eli se ei käytä aiemmin suosittuja sääntöpohjaisia käännösmalleja. Translate osaa ottaa käännöksessä huomioon aiemmin käännettyjen lauseiden kontekstin, joten taustalla on todennäköisesti transformers-arkkitehtuuria ja attention-mekanismia käyttävä malli. AWS ei kuitenkaan paljasta tarkkaan, mitä mallia palvelu tällä hetkellä käyttää. Hinnoittelu Translate-palvelussa menee käännettyjen merkkien perusteella. Tällä hetkellä hinta on noin 15€/(miljoona merkkiä), eli esimerkiksi tyypillisen uutisartikkelin kääntämisen hinnaksi tulisi noin 0,1€.

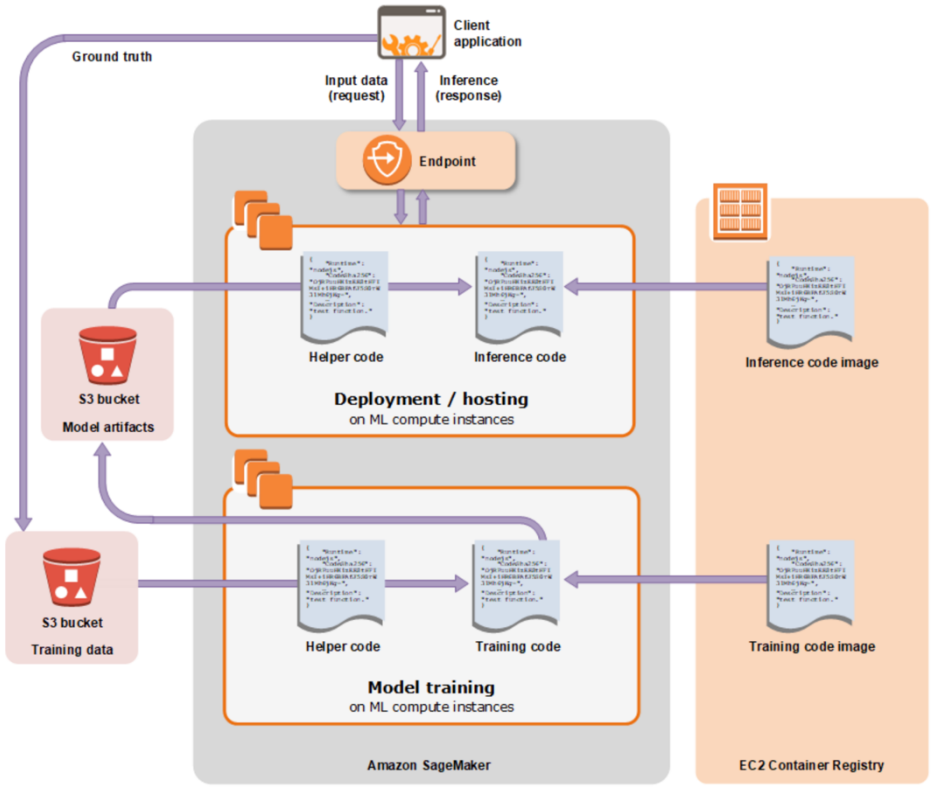
3.2 Koneoppimispalvelut (ML Services)

ML Services on palvelu, joka on keskittynyt pitkälti Amazon Sagemakerin ympärille. Sagemaker on selainpohjainen IDE koneoppimisprojektien infrastruktuurin ja koodin hallintaan. Sillä voidaan hallita koko koneoppimisprojektiin liittyvä prosessi, eli datan esikäsittely sekä mallin hallintaan liittyvät tehtävät. Sillä voidaan myös kouluttaa, julkaista ja monitoroida mallin toimintaa. Kuviossa 2 on esitetty ML-projektin tyypilliset vaiheet.



Kuvio 2. Koneoppimisen prosessisykli (AWS 2020)

Kuviossa 3 näkyy tarkemmin, mitä muita AWS-palveluita ML-projektissa käytetään. Tietokoneet, joissa malleja kehitetään, koulutetaan ja ajetaan, ovat virtuaalisia koneita heidän omassa pilvessään. Artefaktit, eli data jota prosessiin liittyy, tallennetaan S3-tietovarastoon. S3 on tallennuspaikka staattisille tiedostoille, kuten esimerkiksi videoille, tekstitiedostoille ja kuville. Myös monet muut AWS:n palvelut käyttävät S3:a tiedostojen tallennukseen, kuten esimerkiksi ympäristöt mobiiliapplikaatioiden kehitykseen. Kuvassa näkyy myös EC2, eli virtuaalisten koneiden varasto. Varastossa on koneita eri kehitysympäristöjä varten, joissa on esiasennettuna tarvittavia ohjelmistoja sekä valmiita algoritmeja. Koneita valittaessa määritellään myös koneen suorituskyky. AWS:ssä on erittäin paljon erityyppisiä koneita, joista valita. Koneen minuuttihinta yleensä suoraan verrannollinen sen suorituskykyyn ja tyyppiin. Merkittävimmät tekijät, varsinkin koneoppimiseen liittyen, ovat muistin määrä ja näytönohjaimen suorituskyky. Varsinkin neuroverkoissa hyvällä näytönohjaimella varustettu kone on paras valinta.



Kuvio 3. Koneoppimisen datavirta (AWS 2020)

3.2.1 Hinnoittelu

Sagemaker Studion hinnoittelu, jonka ympärille ML-palvelut rakentuvat, rakentuu monesta palasesta ja kustannukset syntyvät pääasiassa käytön mukaan. Sagemaker käyttää taustalla AWS:n perinteisiä resursseja, kuten EFS (*Elastic File Storage*)- ja S3-tiedostojärjestelmiä ja erityyppisiä virtuaalikoneita. Kun tilille ensimmäisen kerran luodaan Sagemaker Studio-ympäristö, luodaan taustalle myös tallennuspaikka ML-projektien tiedostoille. Tallennuspaikka on EFS, josta AWS laskuttaa käytön mukaan.

3.2.2 Datan esikäsittely ja CI/CD

Joulukuussa 2020 AWS julkaisi uusia hyödyllisiä palveluita, joilla datan esikäsittely muuttui helpommaksi. Datan esikäsittely on yleensä ML-projektissa työläin osuus, joka voi vielä 80 prosenttia koko projektin ajasta. Uudet palvelut ovat Data Wrangler, Feature Store ja Sagemaker Pipelines.

Data Wrangler

Data Wranglerilla voidaan hallita Sagemakerin käyttöliittymässä koko datan esikäsittely. Sillä voidaan tuoda dataa AWS:n S3:sta, Athenasta tai Redshiftista. Tuotu data voidaan edelleen yhdistää eri lähteistä haettuun dataan. Esimerkiksi säädata voidaan yhdistää aikasarjadataan, kun tarvitaan tietoaineisto säästä riippuvaisen ennustusmallin koulutusta varten. Dataa voidaan myös puhdistaa tai muuttaa, esimerkiksi poistamalla epäsopivia datapisteitä tai muuttamalla tekstiä kokonaisluvuiksi. NLP-tehtäviin liittyvä tekstin tokenisointi on myös mahdollista. Lopulta käsitelty tietoaineisto voidaan viedä edelleen muihin palveluihin esimerkiksi mallin koulutusta varten. Data Wranglerin luoman Python-koodin voi myös ladata halutessaan, jolloin työnkulku on mahdollista suorittaa myös jossain muussa ympäristössä.

Data Wrangler on käytännössä AWS:n kehittämä framework, joka helpottaa yleisesti käytettyjen esikäsittelykirjastojen käyttöä. Taustalla se käyttääkin luultavasti pandas-kirjastoa, sillä koodia voi kirjoittaa myös halutessaan pandas-syntaksilla. Taustalla Data Wrangler käyttää erittäin tehokasta virtuaalikonetta, jonka tuntihinta on Irlannin alueella noin 1,07 €. Koneen tyyppi on ml.m5.4xlarge, eli siinä on 16 vCPU:ta ja 64 Gt ram-muistia. Palvelua käytettäessä tuleekin olla huolellinen, että sulkee palvelun oikeaoppisesti, jottei virtuaalikone jää tahattomasti käyntiin taustalle, sillä konetta ei tuhota automaattisesti. Päälle unohtunut Data Wrangler voikin luoda nopeasti suuren laskun.

Feature Store

Feature Store on uusi palvelu, joka helpottaa ML-projektin piirteiden suunnittelua (engl. *feature engineering*). Siinä pääkäsite on Feature Group, joka kuvastaa jonkin tietoaineiston piirteitä. Feature Group on kyseisen tietoaineiston skeema. Tietoaineisto voi olla esimerkiksi csv-tiedosto, jossa on talojen hinnat sekä niihin liittyvät piirteet. Toinen käsite Feature Storeen liittyen on Feature Record, joka tarkoittaa prosessoitua yhden datapisteen sisältämää dataa. Data, jonka pohjalta tietue (engl. *record*) on luotu, voi olla esimerkiksi yksi rivi csv-tiedostossa.

Perinteisesti dataa on käsitelty niin, että tämän tyylinen tietoaineisto ladataan Notebookiin yksilöllisesti ja mahdollinen piirteiden suunnittelu tehdään samassa Notebookissa. Sama tietoaineisto piirteineen saattaa olla kuitenkin käytössä monessa mallissa, kuten esimerkiksi talojen hinnat, mutta suunniteltujen piirteiden hyödyntäminen uusien mallien kehittämisessä voi olla hankalaa. Feature Store tuo helpotusta tähän, sillä sen avulla on helppo jakaa jo luotuja tietoaineistoja tiimin kesken. Keskitetty hallinta parantaa myös laatua, sillä kaikkien käyttäessä Feature Storessa olevaa dataa, voidaan olla varmoja datan rakenteesta. (AWS, https://www.youtube.com 2020)

Sagemaker Pipelines

Sagemaker Pipelines on kehitetty tuomaan CI/CD-ominaisuuksia myös ML-kehitykseen. Palvelu toimii suoraan Sagemaker studiossa, jossa on helppo luoda valmista pohjaa käyttäen koko ML-sovelluksen pipeline. Palvelu luo muun muassa git-tietovarastot automaattisesti, jolloin ML-projektin lähdekoodin hallinta ja koodin struktuuri on kerralla kunnossa. Myös sovelluksen eri vaiheet (engl*. stage*), kuten CI/CD-viitekehykseen kuuluu, pystyy tekemään Pipelinen avulla. CI/CD-pipelineen voi luoda ehdollisia ja manuaalisia portteja. Esimerkiksi koulutetun mallin tarkkuudesta voidaan luoda portti, joka estää prosessin etenemisen, mikäli tarkkuus on määritellyn rajan alapuolella. Lopullinen hyväksyntä tuotantoon tehdään AWS:n CodePipeline-palvelussa manuaalisesti valtuutetun henkilön toimesta.

Pipeline toimii samalla periaatteella kuin monet muutkin AWS:n tarjoamat palvelut, eli se hyödyntää AWS:n jo olemassa olevia palveluita. Se toimii yksinkertaistavana rajapintana palveluiden ja käyttäjän välissä, kuten esimerkiksi edellä mainittu CodePipeline-palvelun integrointi osaksi Sagemaker Pipelinea. NLP-tehtävien näkökulmasta Sagemaker Pipeline helpottaa huomattavasti koko ML-sovelluksen hallintaan, jolloin aikaa jää enemmän itse mallien virittämiseen.

3.2.3 Mallin koulutus ja validointi

Mallin voi kehittää ja kouluttaa Sagemakerissa käyttämällä perinteistä Jupyter-Notebookia. Itse kehitystyössä käytetään Sagemakerin IDE:tä, jossa Notebook käyttää taustalla kehittämiseen käynnistettyä Jupyter-palvelua. Jupyter-palvelua ajetaan AWS:n virtuaalikoneessa, jossa on esiasennettuna Sagemakerin SDK, jolla voidaan hallita ML-kehityksessä tarvittavia muita resursseja. Esimerkiksi koulutukseen käytettävän resurssin tyyppi määritellään koodissa muutamalla rivillä. Kuvassa 21 on esimerkki koulutuskoneen määrittelystä, jossa koulutukseen valitaan ml.p3.2xlarge-tyyppinen instanssi.

```
from sagemaker.tensorflow import TensorFlow


mnist_estimator = TensorFlow(entry_point='mnist.py',
                             role=role,
                             instance_count=2,
                             instance_type='ml.p3.2xlarge',
                             framework_version='1.15.2',
                             py_version='py3',
                             distribution={'parameter_server': {'enabled': True}})
```

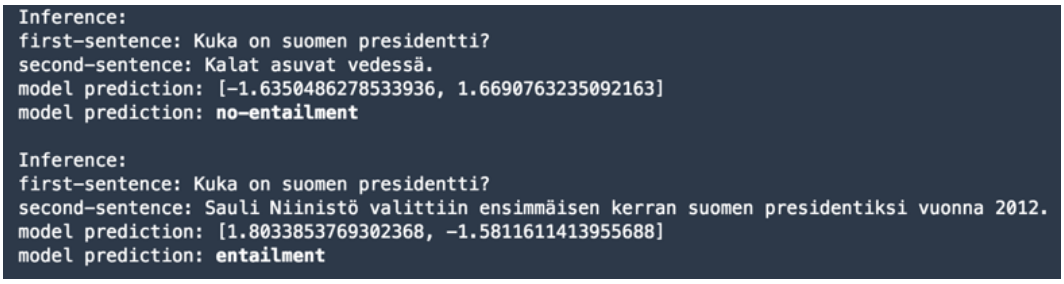
Kuva 21. Resurssin määritys Jupyter-Notebookilla (AWS, https://github.com/aws 2021)

Edellisen esimerkin mukaisesti Notebook-komennolla voidaan luoda virtuaalikoneita myös mallin validointiin ja hostaamiseen. API on selkeä käyttää, mutta vaatii Python-osaamista sekä kyseisen ML-frameworkin hallintaa. Edellisessä esimerkissä käytössä oli TensorFlow. Oppimiskäyrää madaltaakseen AWS on kehittänyt palveluita, joilla liikkeelle pääseminen on tehty helpommaksi. Tällainen on esimerkiksi Jumpstart, josta seuraavassa kappaleessa enemmän.

Sagemaker JumpStart

Vuoden 2020 re:Invent toi NLP-mallien koulutukseen uusia palveluita ja yksi niistä on Sagemaker JumpStart. JumpStart on palvelu, jonka avulla voi luoda koko ML-sovelluksen suoraan Sagemaker Studion web-konsolista. Konsolista voi valita minkä mallin pohjalta sovellus luodaan, jonka jälkeen JumpStart luo kaikki sovellukseen liittyvät palvelut yhdellä napin painalluksella. Taustalla AWS luo Cloudformation-infrastruktuurin, joka pitää sisällään sovelluksen tarvitsemat palvelut. Palvelu luo myös automaattisesti JupyterLab-Notebookin, jolla luodun ML-rajapinnan (engl. *endpoint*) toimintaa voi testata suoraan uudessa instanssissa.

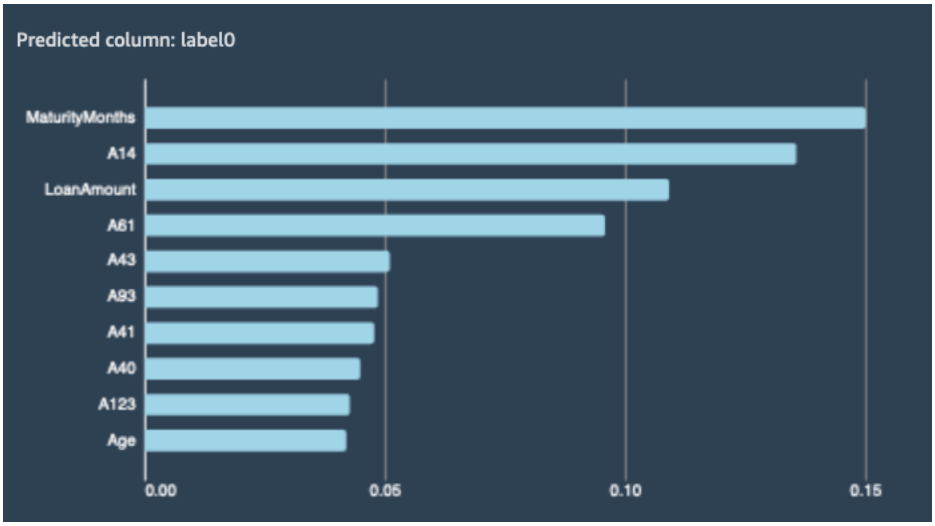
NLP-malleja JumpStart sisältää tällä hetkellä 39 kappaletta. Tarjolla on muun muassa useita BERT-malleja, mutta suomen kielen näkökulmasta valmiit mallit ovat toistaiseksi vielä rajoittuneita. Suurin osa malleista on yksikielisiä malleja, jotka on koulutettu englannin kielellä. Esimerkiksi FinBERT- ja XLM-R-malleja ei JumpStartista löydy. Mallit on myös esikoulutettu lauseparien vastaavuustehtävillä, joten myös tehtävien määrä on toistaiseksi rajoittunut. Käyttöliittymän yleisnäkymässä tarjotaan myös hienosäätömahdollisuutta, mutta hienosäätö ei vielä tällä hetkellä ole käytössä. Testaushetkellä palvelu oli vasta kuukauden ikäinen, joten nämä puutteet tullaan korjaamaan varmasti nopealla aikataululla. Kuvassa 22 näkyy ennustuksen tulos JumpStart-palvelulla luodusta distilbert-base-multilingual-cased-mallista. Kuva on kaapattu JumpStart-palvelun luomasta Notebookista.



Kuva 22. JumpStart-mallin ennustus distilbert-base-multilingual-cased-mallilla

Sagemaker Clarify

Sagemaker Clarify on Sagemaker studioon sisältyvä uusi palvelu, joka nimensä mukaisesti auttaa ymmärtämään koulutetun mallin ominaisuuksia, ennustukseen eniten vaikuttavia piirteitä ja mahdollisia vinoutumia (engl. *bias*). Clarifyn avulla on helppo luoda raportti, josta näkee mitkä piirteet mallille syötettävässä datassa vaikuttavat eniten ennustukseen. Esimerkiksi luotonantopäätöstä tekevässä mallissa suurin vaikuttava tekijä voi olla haetun luoton pituus ja pienin hakijan ikä, kuten kuviossa 4 on esitetty.



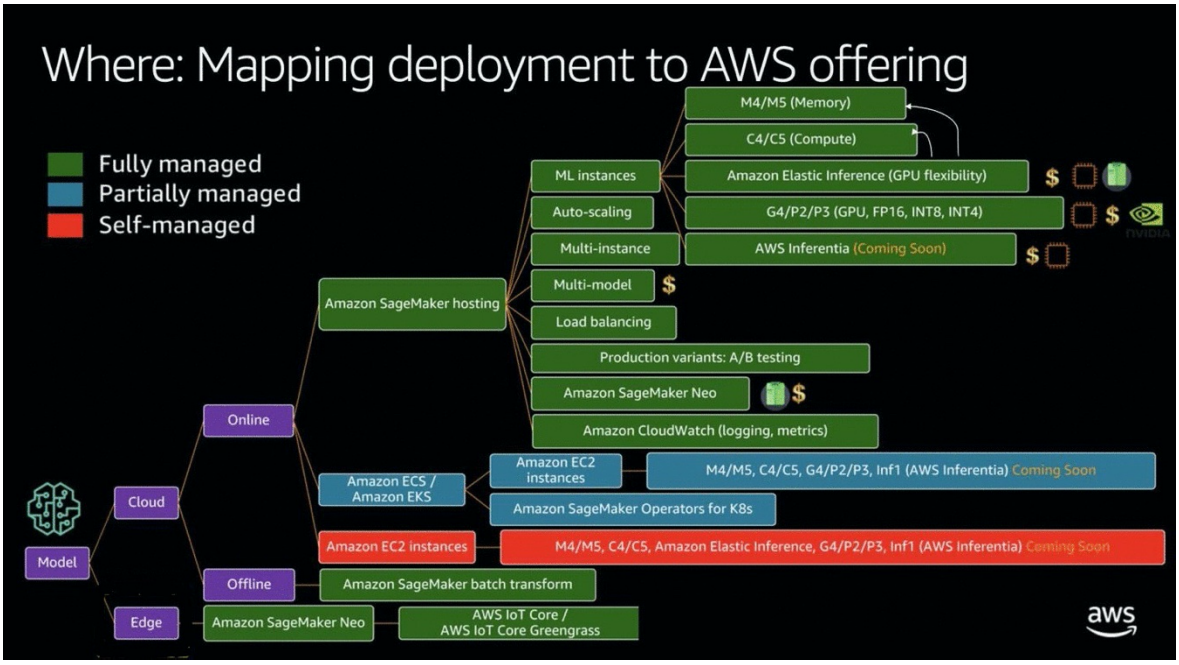
Kuvio 4. Sagemaker Clarify-palvelun esimerkkiraportti (Simon 2020)

Luotonantopäätöksen antava malli voi myös vinoutua, mikäli piirteiden esiintymien määrä tietoaineistossa poikkeaa merkittävästi. Mikäli kyseisen mallin koulutusdatassa on jokin etninen ryhmä merkittävästi aliedustettuna, mallin antamat ennustukset voivat olla vinoutuneita. Clarifyn avulla voidaankin tutkia, suosiiko luotonantopäätöstä antava malli jotain tiettyä etnistä ryhmää. Clarify ei kuitenkaan kerro syytä, mistä mahdollinen suosiminen johtuu, vaan ihmisen on tutkittava mahdollinen syy manuaalisesti.

3.2.4 Julkaisuvaihtoehdot

Kaikki tekoälymallit ovat käytännössä tietokoneohjelmia, joten niitä voidaan ajaa monessa erityyppisessä ympäristössä. Mallien koosta, arkkitehtuurista ja vaaditusta suorituskyvystä riippuu, missä ympäristössä mallia voi ajaa. Kuviossa 5 esitetään tämän hetken vaihtoehdot AWS-ympäristössä. Kuvion esittämien vaihtoehtojen lisäksi on mahdollista ajaa mallia myös Serverless-arkkitehtuurilla, kuten FastText-mallia käsittelevässä luvussa mainittiin.

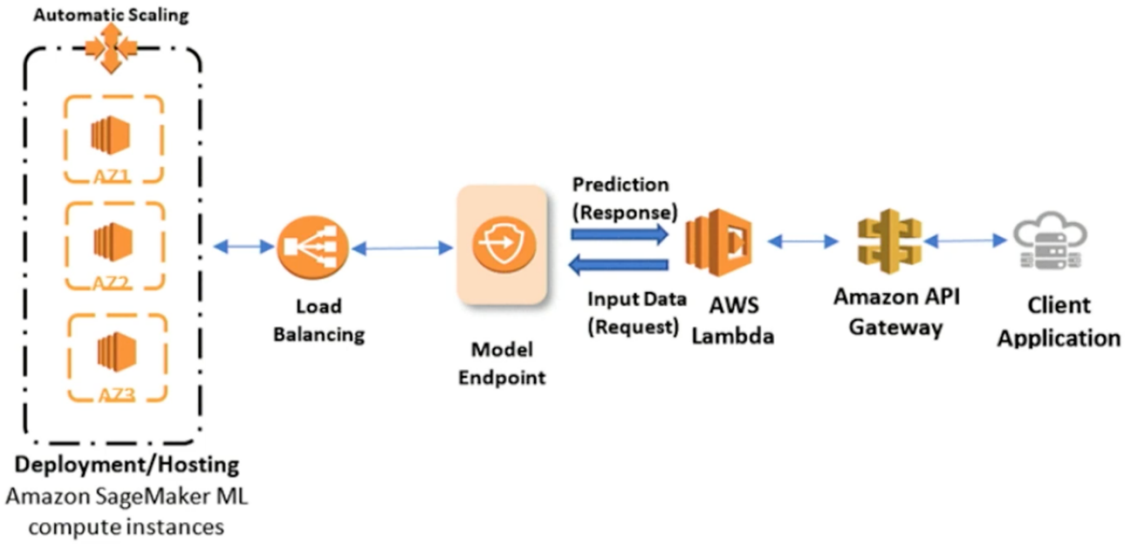
Kuvion violetit laatikot esittävät hyvin perusjaottelun, eli ajetaanko mallia lokaalisti lähellä kohdetta, eli edgessä vai ajetaanko sitä pilvessä. Valintaan vaikuttavat sovelluksen vaatimukset, kuten esimerkiksi lokaali mallin ajaminen voi olla ainoa vaihtoehto päättelyn vaatiman nopeuden tai tietoturvavaatimusten vuoksi. Esimerkiksi itseohjautuvan auton mallien ajamiseen edge on käytännössä ainoa vaihtoehto, sillä pilvessä ajettaessa viiveet kasvaisivat niin suuriksi, että autolla pitäisi ajaa todella hiljaa, jotta se olisi turvallista.



Kuvio 5. Mallien julkaisuvaihtoehdot AWS-ympäristössä (AWS 2020)

Pilvessä mallia voidaan ajaa käytännössä kolmessa eri ympäristössä, eli Sagemakerissa, kontissa (ECS/EKS) tai itse hallitussa EC2-virtuaalikoneessa. Vaikka vaihtoehtojen määrä näyttää suurelta, loppujen lopuksi taustalla on kuitenkin aina Linux-käyttöjärjestelmä ja sovelluksen vaatimuksen mukainen laitteisto. Hostaus Sagemakerissa kuitenkin helpottaa hallintaa monella tavalla, kuten esimerkiksi uusien mallien julkaisua tai kuormantasausta. Kuviossa 6 on esimerkki hostauksesta Sagemakerilla. Esimerkin arkkitehtuuri noudattaa

tämän hetken parhaita käytäntöjä, sillä komponentit ovat Serverless-tyyppisiä ja skaalautuminen vaihtelevaan kuormitukseen on automatisoitu.



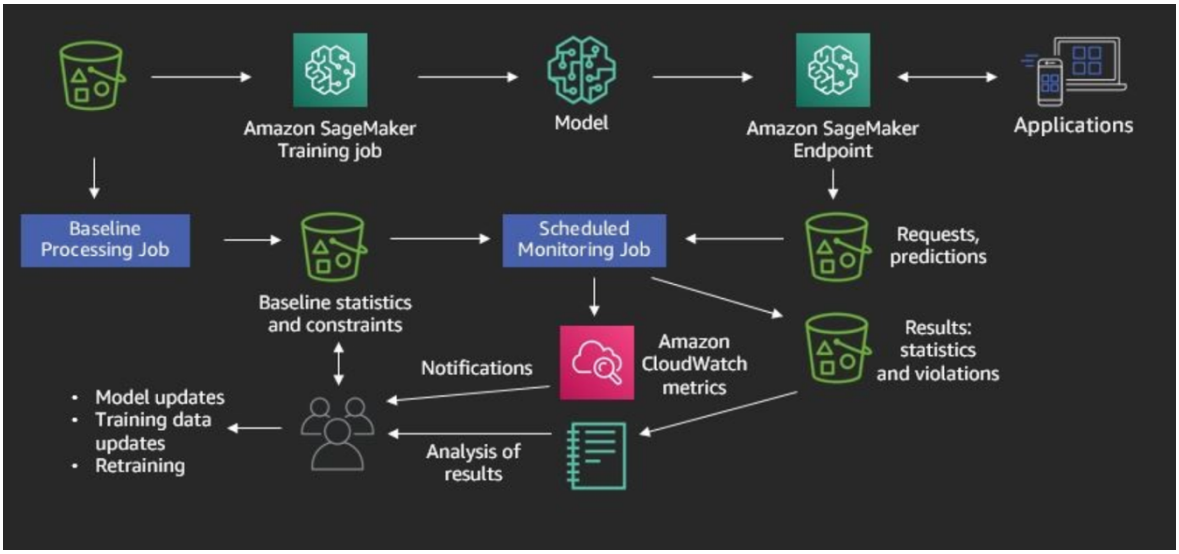
Kuvio 6. Mallin hostaus AWS:n pilviympäristössä (AWS 2019)

Kuviossa 5 edge-julkaisuvaihtoehtoja on vain yksi, mutta edge pitää sisällään monipuolisen ja mullistavan frameworkin. AWS Iot Core Greengrass mahdollistaa paikallisesti ajettavien mallien keskitetyn hallinnan pilvessä. Malleja voi ajaa Windows- ja Linux-käyttöjärjestelmillä varustetuissa koneissa, jollaisia ovat esimerkiksi Raspberry Pi-minitietokoneet. Mallit voidaan kouluttaa vaikka omalla koneella tai pilvessä ja siirtää hallitusti paikallisille koneille. Koneet ovat käytännössä IoT-laitteita, joissa Greengrass toimii rajapintana paikallisen tietokoneen käyttöjärjestelmän ja pilven välillä. Greengrass takaa tietoturvallisen tiedonsiirron, johon kuuluu muun muassa sertifikaatit itse laitteissa ja pilvessä.

NLP-sovellusten näkökulmasta Greengrass mahdollistaa paikallisesti ajettavien mallien helpomman hallinnan. Esimerkiksi sairaalaympäristössä olevan puhetta ymmärtävän sovelluksen mallia voidaan Greengrassin avulla kouluttaa helposti uudelleen itse laitteessa ja säilyttää silti mallien hallinta keskitetysti pilvessä. Eli tällä saavutetaan sovellusten joustava hallinta tinkimättä korkeista tietoturva- tai yksityisyydensuojavaatimuksista.

3.2.5 Mallin monitorointi

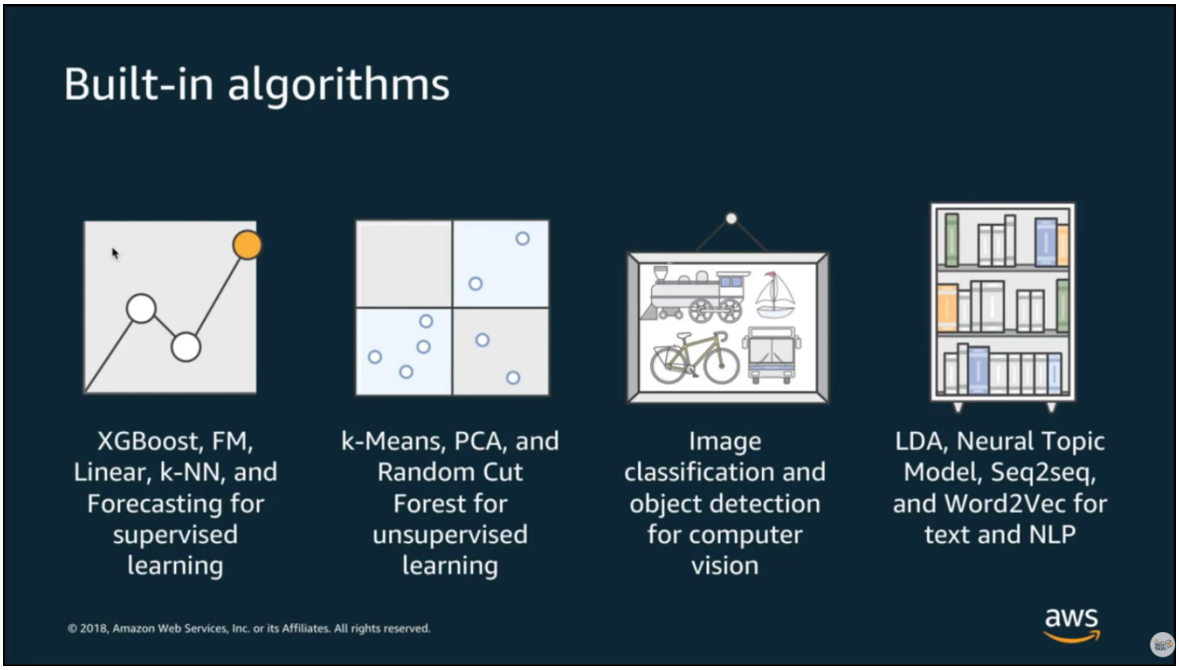
Ajan saatossa koulutettu malli alkaa pikkuhiljaa huonontumaan, sillä maailma mallin ympärillä muuttuu. Aiemmin mallin tarkkuuden arviointi on ollut manuaalista työtä. Malli on pitänyt uudelleenkouluttaa uudella datalla tasaisin aikavälein. AWS tarjoaa tähän myös automaattisen ratkaisun. Automaattisella mallin monitorointitehtävällä Sagemaker käyttää tilastollista analyysia ja vertaa mallille syötettyä dataa lähtötilanteeseen. Monitorointia varten lähtötilanne pitää olla siis tallessa. AWS suosittelee, että lähtötilanteen luomiseen käytetään koulutukseen käytettyä tietoaineistoa. Mikäli data alkaa vaeltamaan lähtötilanteesta, voidaan tästä luoda hälytys mallista ja ryhtyä tarvittaviin toimenpiteisiin. Kuviossa 7 esitetään mallin monitoroinnin prosessi.



Kuvio 7. Mallin monitorointi (AWS 2020)

3.3 Valmiit algoritmit

AWS tarjoaa myös valmiita algoritmeja, joita voi käyttää silloin, kun käyttää koulutukseen tai hostaukseen heidän ympäristöään. Valmista algoritmia voi käyttää esimerkiksi Jupyter-Notebookissa importoimalla algoritmin sisältävä moduuli Notebook-ympäristöön. Kuvassa 23 on esitetty AWS:n algoritmitarjonta tällä hetkellä. NLP-tehtävistä varsinkin luokitteluun löytyy useita vaihtoehtoja, joiden taustalla olevat arkkitehtuurit poikkeavat hieman toisistaan. Tässä työssä käydään tarkemmin läpi BlazingText, LDA ja NTM.



Kuva 23. AWS:n valmiita algoritmeja https://youtu.be/R0vC31OXt-g?t=491

3.3.1 BlazingText

BlazingText on AWS:n versio FastText-mallista, jolla mallia voidaan ajaa myös moniytimisellä CPU:lla sekä GPU:lla hyödyntäen CUDA-ytimiä. Alkuperäinen FastText toimii vain yksiytimisellä CPU:lla. BlazingText on tästä johtuen huomattavasti suorituskykyisempi kuin FastText. (Amazon Web Services 2020, 656-660)

Mallilla voidaan suorittaa kaksi tehtävää:

1. Word2Vec (luodaan word embedding-matriisi korpuksesta)
2. Tekstin luokittelu

Rajoite alkuperäiseen FastText-malliin on se, ettei BlazingText tue dokumentaation mukaan kielen tunnistusta. FastText pystyy tunnistamaan 170 eri kieltä pieneen kokoon pakatulla mallilla. (Facebook Inc 2017).

3.3.2 Latent Dirichlet Allocation (LDA)

LDA on ohjaamaton algoritmi, jolla voidaan luokitella tekstimuotoista dataa sisällön perusteella. Etukäteen ei voida tietää, millaisia luokkia algoritmi dokumenteista luo. Jokainen dokumentti saa jonkin todennäköisyyden kuulua algoritmin luomiin luokkiin dokumentin sisältämien sanojen perusteella. Taulukossa 3 nähdään esimerkkiluokittelu, jossa algoritmi on luonut luokat Topic 1 ja Topic 2. Kissoja käsittelevät dokumentit, joissa esiintyy sana meow, kuuluvat todennäköisesti luokkaan Topic 1 ja koiria käsittelevät dokumentit luokkaan Topic 2. (AWS 2020, 333). Taulukossa 4 on listattu LDA:n hyperparametrit. LDA:ssa luokkien määrä on maksimissaan 150, joka voi olla laajemmissa aineistoissa rajoittava tekijä.

Topic	*eat*	*sleep*	*play*	*meow*	*bark*
Topic 1	0.1	0.3	0.2	0.4	0.0
Topic 2	0.2	0.1	0.4	0.0	0.3

Taulukko 3. LDA esimerkkiluokittelu (AWS 2020, 333)

Parametrin nimi	Kuvaus
num_topics	Kuinka monta erilaista luokkaa algoritmi luo aineistosta. Pakollinen parametri. Arvo 1 - 150
feature_dim	Ominaisuuksien määrä, jonka algoritmi aineistosta luo. Jokainen yksittäinen sana on ominaisuus, joten tämä tarkoittaa käytännössä sanastoa.
mini_batch_size	Satsin koko, eli opetukseen yhdellä ajolla käytettävien dokumenttien määrä.
alpha0	*Concentration parameter* arvon arvaus. Tämä parametri kuvaa sitä, miten aineisto on jakautunut todennäköisyysjakaumalla. Pieni luku tarkoittaa hajallaan olevaa aineistoa ja yli yhden arvot tiivistä. Oletus: 0,1 Arvo 0,1 - 10
max_restarts	Algoritmin *Alternating Least Squares (ALS)* – vaiheen parametri, jonka avulla voidaan löytää funktion paikallinen minimi paremmin. Oletus: 10
max_iterations	Algoritmin ALS – vaiheen parametri. Kuinka monta iteraatiota ajetaan. Oletus: 1000
tol	Algoritmin ALS – vaiheen parametri. Funktion virhetoleranssin arvo. Oletus: 1e-8

Taulukko 4. LDA-hyperparametrit (AWS 2020, 336)

3.3.3 Neural Topic Model (NTM)

NTM on ohjaamaton algoritmi tekstiaineiston luokitteluun, toisin sanoen sillä on mahdollista tehdä klusterianalyysia. NTM on saman tapainen kuin LDA, mutta antaa kuitenkin eri tuloksia. NTM:ää voi sanoa kehittyneemmäksi ratkaisuksi, sillä se käyttää nimensä mukaisesti luokitteluun neuroverkkomallia. Myös hyperparametrit ovat tyypillisiä neuroverkkomallin liittyviä. Tällaisia ovat muun muassa epochien määrä ja optimointiin käytettävä algoritmi. LDA:han verrattuna NTM on huomattavasti parempi, mikäli lajiteltavia luokkia on yli 150, sillä NTM:n maksimäärä on 1000. LDA:ssa maksimi on 150.

Sagemaker ympäristössä on valmiita Notebook-malleja, joilla voi luoda luokittelutyön. Tässä työssä ei testattu NTM:n toimintaa, mutta dokumentaation perusteella mallin voi päätellä olevan modernilla arkkitehtuurilla toteutettu neuroverkko. NTM-mallin ajaminen Sagemaker Notebookissa on helppoa, eli muutamalla koodirivillä saa käynnistettyä opetuksen, jolloin AWS luo virtuaalisen ajoympäristön mallin koulutukseen. Parametreilla saa määritettyä, minkä tyyppisellä laitteistolla koulutus tehdään. NTM vaatii kuitenkin datan esikäsittelyä, joten aivan automaattinen toiminta ei ole. Data tulee muun muassa stemmata tai lemmata ja tehdä myös muut perinteiset esikäsittelyt. AWS:n videoblogissa on hyvä esimerkki NTM:stä käytännössä. Video löytyy osoitteesta https://youtu.be/aFE_tejgegU?t=813. Myös AWS:n verkkosivun blogista avautuu paremmin, mitä esitöitä mallin käyttäminen vaatii. Blogi NTM:stä löytyy linkistä https://aws.amazon.com/blogs/machine-learning/introduction-to-the-amazon-sagemaker-neural-topic-model.

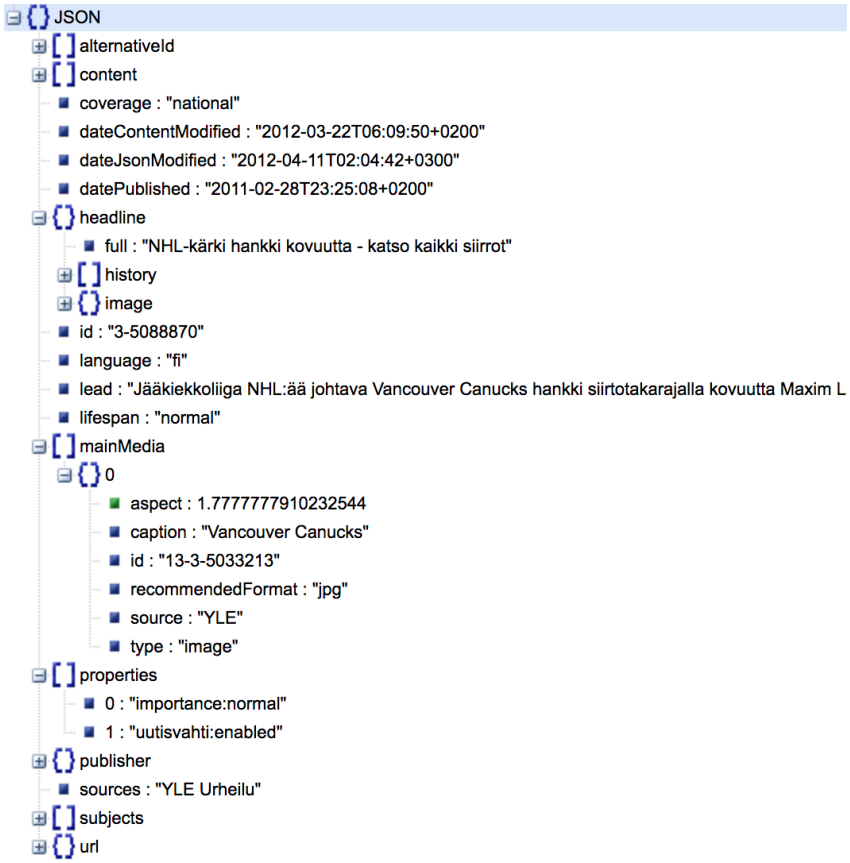
4 EMPIIRINEN OSUUS - LUOKITTELUMALLIEN VERTAILUA

Tämän opinnäytetyön käytännön osuudessa eri malleja testataan samalla tietoaineistolla ja arvioidaan niiden tarkkuutta sekä muita ominaisuuksia, myös mallin implementoinnin ja kustannustehokkuuden näkökulmasta.

4.1 Tietoaineisto - YLE:n uutisdata

Tässä työssä käytetään Ylen suomalaisia uutisia aikaväliltä 2011–2018. Raakadata on JSON-muodossa, jota ei sellaisenaan voi käyttää neuroverkkomallien koulutukseen. Data pitää suodattaa ja sen muotoa pitää muuttaa, jotta sitä on mahdollista käsitellä tokenisointityökaluilla.

Raakamuodossa YLE:n uudisdata on JSON-muotoista ja sisältää paljon metadataa. Yksi lyhyehkön urheilu-uutisen JSON-tiedostossa on yhteensä melkein 700 riviä pitkä, jossa jokainen tekstikappale varaa yhden rivin. Alla esimerkki yhdestä uutisesta, jossa itse sisältö on content – ominaisuuden alla. JSON-data visualisointiin käytettiin http://json-viewer.stack.hu sivuston parseria.



Kuva 24. YLE-uutisten datamalli

4.2 Datan esikäsittely

Korpuksen luomiseen käytetään Turun NLP-ryhmän luomaa Python skriptiä. YLE:n lisenssi kieltää datan uudelleenjakelun, joten raakadata on ladattava itse YLE:n tarjoamasta lähteestä ja ajettava skripti, joka luo kymmenen luokan tasapainotetut tietoaineiston. Koodit luomiseen löytyvät Sampo Pyysalon GitHub tietovarastosta https://github.com/spyysalo/yle-corpus. Skripti luo tietoaineistot koulutukseen, validointiin koulutuksen aikana ja testaukseen. Skriptin luomat luokat on esitetty seuraavassa taulukossa. Koulutusdataa on yhteensä 99975 riviä, validointidataa 9999 riviä ja testausdataa 9999 riviä. Koulutusdata on tasapainotettu niin, että jokaisen luokan uutisia on noin 10000 kappaletta. Taulukossa 5 on esitetty luokat ja niiden osuus tietoaineistosta.

Luokka	% - osuus	Määrä
kulttuuri	~10	9999
liikenne ja kuljetus	~10	9995
luonto	~10	10000
politiikka	~10	9998
urheilu	~10	9997
talous	~10	9998
onnettomuudet	~10	9998
rikokset	~10	9999
terveys	~10	9994
koulutus ja kasvatus	~10	9997
	Yhteensä	99975

Taulukko 5. Koulutukseen käytetty tietoaineisto

Pyysalon parserilla JSON-muotoinen data parsitaan tekstitiedostoksi, jossa jokainen uutinen on omalla rivillään. Uutisen luokka on rivin ensimmäinen sana ja erotettu uutisen ensimmäisestä sanasta välilyönnillä. Formaatti on muodossa, joka voidaan syöttää suoraan fastText-mallille. FinBERT ja XLM-R eivät tosin hyväksy kyseistä formaattia, vaan dataa pitää käsitellä hieman. Yleinen formaatti, jossa dataa käsitellään, on csv- tai tsv-muoto. Tekstitiedoston muuttamiseen tsv-muotoon käytettiin sed-ohjelmaa. Jokaisen rivin ensimmäinen välilyönti muutettiin sarkainerottimeksi ja tiedostopäätteeksi vaihdettiin tsv. Sed-komento kyseisen operaation suorittamiseksi on esitetty kuvassa 25.

```
sed 's/ /'$'\t''/' alkupe.txt > muokattu.tsv
```

Kuva 25. Aineiston esikäsittely sed-komennolla

4.3 Ajoympäristöt ja kirjastot

Kuten tässä työssä on aiemmin mainittu, malleja voi kehittää, kouluttaa ja ajaa monessa eri ympäristössä. AWS:n palveluista löytyy kaikki mahdolliset ajoympäristöt, mutta tässä työssä ympäristöinä päädyttiin käyttämään omalle koneelle asennettavaa Anacondaa sekä Googlen tarjoamaa Colab-ympäristöä. Näihin valintoihin vaikutti paljon kustannusnäkökulma, sillä Colab on tällä hetkellä ilmainen.

Ilmaisuudesta huolimatta, Colab tarjoaa erittäin tehokkaat resurssit, kuten esimerkiksi parhaassa tapauksessa Nvidian Tesla V100 GPU:n. Kuluttajalle tuo GPU maksaisi tällä hetkellä kaupasta ostettuna 6200 €, joten Colabia käyttämällä voi saavuttaa merkittäviä kustannussäästöjä. Colab ei kuitenkaan aina anna tehokkaimpia resursseja, vaan jakaa niitä oman logiikkansa perusteella. Ajoajoilla on myös rajoituksia, kuten 12 tunnin maksimiaika, joten pidempien koulutusten tekeminen ei onnistu kerta-ajolla. Tähänkin on kyllä olemassa ratkaisu, sillä Wandb-palvelulla neuroverkon opetetut parametrit on mahdollista tallentaa jokaisen iteraation jälkeen ja jatkaa koulutusta siitä mihin koulutus jäi Colabin katkaistessa yhteyden. Tässä työssä ei tosin kouluteta mallia lähtemällä nollasta, vaan tehdään hienosäätöä mallien päälle, jolloin ajoaikarajoitukset eivät aiheuta ongelmia.

Fasttext-mallia koulutettiin omalla Macbook Pro kannettavalla Anaconda-ympäristössä. Fasttext toimii vain CPU:lla, eikä tällöin Colabin tehokkaasta GPU:sta olisi ollut mitään hyötyä.

4.3.1 Huggingface - Transformers

Huggingfacen Transformers on suosittu NLP-framework, jolla voi suorittaa NLP-tehtäviä jopa muutamilla koodiriveillä. Siitä löytyy esikoulutettuja malleja, joita on helppo käyttää sen oman pipeline-API:n kautta. Myös mallien hienosäätö oman datan avulla on mahdollista ja Transformers tukee tällä hetkellä kahta suosituinta ML-frameworkia, eli TensorFlow:ta ja PyTorch:ia. Huggingfacen verkko-osoite on https://huggingface.co, josta löytyy linkit Transformersiin.

Myös tässä työssä käytettävät BERT-mallit löytyvät esikoulutettuina Huggingfacen kirjastosta.

4.3.2 SimpleTransformers

Huggingfacen Transformers-NLP-framework voi olla aloittelijalle vaikea ymmärtää, vaikka se onkin yksinkertaistanut NLP:n käyttöä huomattavasti. Tähän ongelmaan on luotu uusi framework, joka automatisoi ja piilottaa käyttäjältä mahdollisesti vaikeaselkoisia parametreja. Framework käyttää taustalla Huggingfacen Transformers-frameworkia, joten sillä voi käyttää kaikkia samoja malleja kuin alkuperäiselläkin frameworkilla. SimpleTransformers tarjoaa myös muita hyviä lisäominaisuuksia, kuten yksinkertainen integrointi Wandb-palvelun kanssa. Muutamalla koodirivillä saadaan luotua yhteys Wandb-rajapintaan, jolloin koulutusvaiheessa saadaan tallennettua koulutukseen liittyvää statistiikkaa. Wandb:llä voidaan tallentaa myös koulutuksen sen hetkiset parametrit, jolloin voidaan tarvittaessa palata johonkin koulutuksen pisteeseen ja jatkaa koulutusta.

4.4 Luokittelijan tarkkuuden arviointi

Mallien suorituskyvyn arviointiin käytettiin symmetristä tietoaineistoa, jossa oli 1000 datapistettä jokaisesta luokasta. Mallin yksittäisen datapisteen ennuste on lista painoarvoja pisteen kuulumisesta kuhunkin luokkaan. Listamuotoisesta ennusteesta ei ole kuitenkaan mahdollista luoda sekaannusmatriisia (engl*. confusion matrix*), joten ennuste pitää muuttaa skalaariksi, jonka arvo kertoo todennäköisimmän luokan. Muunnokseen käytettiin NumPy:n argmax - algoritmia. Mallin arviointiin käytettiin Scikit learn - kirjaston (sklearn) metriikoita. Taulukosta 6 nähdään esimerkkinä FinBERT-mallin yhteenveto mallin tarkkuudesta sekä luokkakohtaiset tunnusluvut.

	precision	recall	f1-score	support
kulttuuri	0.94	0.91	0.93	1000
liikenne_ja_kuljetus	0.85	0.95	0.90	1000
luonto	0.91	0.93	0.92	999
politiikka	0.89	0.89	0.89	1000
urheilu	0.96	0.99	0.98	1000
talous	0.92	0.78	0.85	1000
onnettomuudet	0.94	0.90	0.92	1000
rikokset	0.93	0.95	0.94	1000
terveys	0.92	0.92	0.92	1000
koulutus_ja_kasvatus	0.94	0.95	0.95	1000
accuracy			0.92	9999
macro avg	0.92	0.92	0.92	9999
weighted avg	0.92	0.92	0.92	9999

Taulukko 6. FinBERT-luokitteluraportti

Tunnusluvut voidaan määritellä taulukon 7 mukaisesti. Kaavoissa luokittelutulosten lyhenteet ovat seuraavat:

TP(True Positive) = Oikein positiiviseksi luokiteltu positiivinen datapiste

TN(True Negative) = Oikein negatiiviseksi luokiteltu negatiivinen datapiste

FP(False Positive) = Väärin positiiviseksi luokiteltu negatiivinen datapiste

FN(False Negative) = Väärin negatiiviseksi luokiteltu positiivinen datapiste

Tunnus-luku	Kaava	Määritelmä	Vastaa kysymykseen
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Luokittelijan kokonaissuoritus-kyky	Kuinka monta datapistettä luokiteltiin oikein?
Precision	$\frac{TP}{TP+FP}$	Oikein positiivisiksi luokiteltu-jen datapisteiden lukumäärä jaettuna kaikkien positiivisiksi luokiteltujen datapisteiden lu-kumäärällä	Kuinka moni positiiviseksi luokiteltu on todella positiivi-nen?
Recall	$\frac{TP}{TP+FN}$	Oikein positiivisiksi luokiteltu-jen datapisteiden lukumäärä jaettuna tietojoukon kaikkien positiivisien datapisteiden luku-määrällä	Kuinka monta tietojoukossa olevaa oikeasti positiivista luokiteltiin positiiviseksi?
F1-score	$\frac{2 * Precision * Recall}{Precision + Recall}$	Precision- ja recall-arvojen yh-distelmä (harmoninen kes-kiarvo)	

Taulukko 7. Luokittelijan tunnuslukuja (Sokolova ja Lapalme 2009, 429-430)

Moniluokkaluokittelijan tunnusluvuista tämän tyyppisessä tekstin luokittelussa kuvaavin on yleensä F1-score, mikäli tietoaineisto on epäsymmetrinen. F1-score on precision- ja re-call-arvojen harmoninen keskiarvo, joten se antaa paremman kuvan mallin suoritusky-vystä. Tässä tapauksessa kuitenkin, kun tietoaineisto on symmetrinen, saa accuracy sa-man arvon kuin F1-score. Yksittäisen luokan luokittelun suorituskykyä F1-score kuvaa tässä tapauksessa kuitenkin parhaiten.

Recall (herkkyys) kertoo suhteen oikein menneiden ennusteiden ja todellisten oikeiden vä-lillä. Esimerkiksi FinBERT-luokitteli liikenneuutisista oikein 950 kappaletta, jolloin suhde-luku on 950/1000=0,95. Precision vastaavasti kertoo sen, kuinka moni muuhun luokkaan kuuluva uutinen luokiteltiin virheellisesti kyseiseen luokkaan. Esimerkiksi liikenneuutisiin luokiteltiin yhteensä 1121 uutista, joten väärin positiivisesti luokiteltuja oli yhteensä 171 kappaletta.

Esimerkki. Liikenne ja kuljetus:

$Precision_{liikenne} = \frac{950}{950+171} = 0{,}85$ $\quad$ $Recall_{liikenne} = \frac{950}{950+50} = 0{,}95$

$F1_{score_{liikenne}} = \frac{2*0{,}95*0{,}85}{0{,}95+0{,}85} = 0{,}90$

4.5 FinBERT-luokittelu

FinBERT-luokitteluun käytettiin Turun NLP-tutkimusryhmän esikoulutettua mallia Huggingfacen Transformers-kirjastoa hyödyntäen. Malli löytyy Huggingfacen hallitsemasta kirjastosta, josta se oli helppo ladata Notebookiin. FinBERT on esikoulutettu MLM-malli, jonka voi edelleen kouluttaa johonkin erityisempään tehtävään. Tässä tapauksessa se koulutettiin YLE:n uutisdatalla luokittelemaan uutisia. Koska malli on melko iso, on koulutukseen järkevää käyttää GPU:ta. Koulutusta kannettavalla tietokoneella ei edes harkittu. Tässä työssä koulutukseen käytettiin Googlen Colab-ympäristöä, jossa koulutuksen sai tehtyä ilmaiseksi. Koulutettujen mallien varastopaikkana oli Google Drive, jolloin mallien käsittely Colab-ympäristössä oli nopeaa. Koulutukseen, validointiin ja testaukseen käytettiin samaa tietoaineistoa kuin FastText-koulutuksessakin. Huggingfacea käytettäessä framework-vaihtoehdot ovat TensorFlow ja PyTorch. FinBERT-luokitteluun valittiin PyTorch, jotta koodin kanssa tulisi vähemmän yllätyksiä. TensorFlow-tuki on Huggingfacessa melko tuore, joten ohjelmistovirheitä voi löytyä vielä jonkin verran.

Colab-Notebookissa esikoulutetun BERT-mallin hienosäätö luokittelutehtävään etenee seuraavasti:

1. Asennetaan Huggingfacen Transformers Colab - ympäristöön
2. Importoidaan tarvittavat kirjastot (pandas, NumPy, TensorFlow, PyTorch, transformers)
3. Liitetään Google Drive virtuaaliseksi levyasemaksi (engl. *mount*)
4. Ladataan koulutus- ja testidata ympäristöön
5. Ladataan esikoulutettu FinBERT-malli Huggingfacen palvelimelta (TurkuNLP/bert-base-finnish-cased-v1)
6. Alustetaan tokenisoija esikoulutetulla mallilla
7. Tokenisoidaan data. Yksittäisen datapisteen maksimipituus BERT-mallissa on 512 tokenia, joten samalla ylipitkät lauseet typistetään (engl. *truncate*) ja alle 512 tokenia pitkien tensorien loppuosa täytetään nollilla (engl. *padding*), jotta kaikki datapisteet ovat täsmällään 512 tokenia pitkiä. Tokenisoinnissa luodaan myös attention-maski ja label-vektori.
8. Yhdistetään tokenisoitu data, attention-maski ja label-vektori yhdeksi isoksi TensorDataset-objektiksi.

9. Importoidaan PyTorch-kirjastosta tarvittavat moduulit datan lataamiseen GPU:lle (DataLoader, RandomSampler, SequentialSampler). Alustetaan satsin koko (engl. *batch*).
10. Luodaan malli importoimalla Transformers-kirjastosta luokitteluun sopiva moduuli (BertForSequenceClassification) ja alustetaan se esikoulutetulla FinBERT-mallilla.
11. Alustetaan optimointialgoritmi. Käytetään AdamW-optimoijaa Transformers-kirjastosta. Se on paranneltu versio Adam-optimointialgoritmista.
12. Määritellään epochien määrä, eli montako kertaa dataa käytetään koulukseen. Valittiin arvo 2.
13. Koulutetaan malli. Mallin koulutus kestää noin 1,5 tuntia/epoch, eli yhteensä 3 tuntia.
14. Arvioidaan koulutuksen onnistumista kuvion 8 perusteella.
15. Tallennetaan koulutettu malli Google Drive-asemalle. Koulutettu malli tulee tallentaa jonnekin Colab-ympäristön ulkopuolelle, sillä malli tuhotaan, mikäli Google sammuttaa kyseisen Colab-ympäristön. Maksimi ajoaika on noin 12 tuntia.

Kuvio 8. FinBERT-koulutus

Kun malli on koulutettu, sitä voidaan käyttää luokitteluun uudella tekstiaineistolla. Mallille syötettävä teksti pitää esikäsitellä samoin kuin koulutusvaiheessakin, eli tokenisointiprosessi on identtinen. Kuvassa 26 on esimerkki luokittelukoodista, joka tulostaa painoarvon eri luokkiin kuulumisesta. Koodin tuloste on esitetty kuvassa 27.

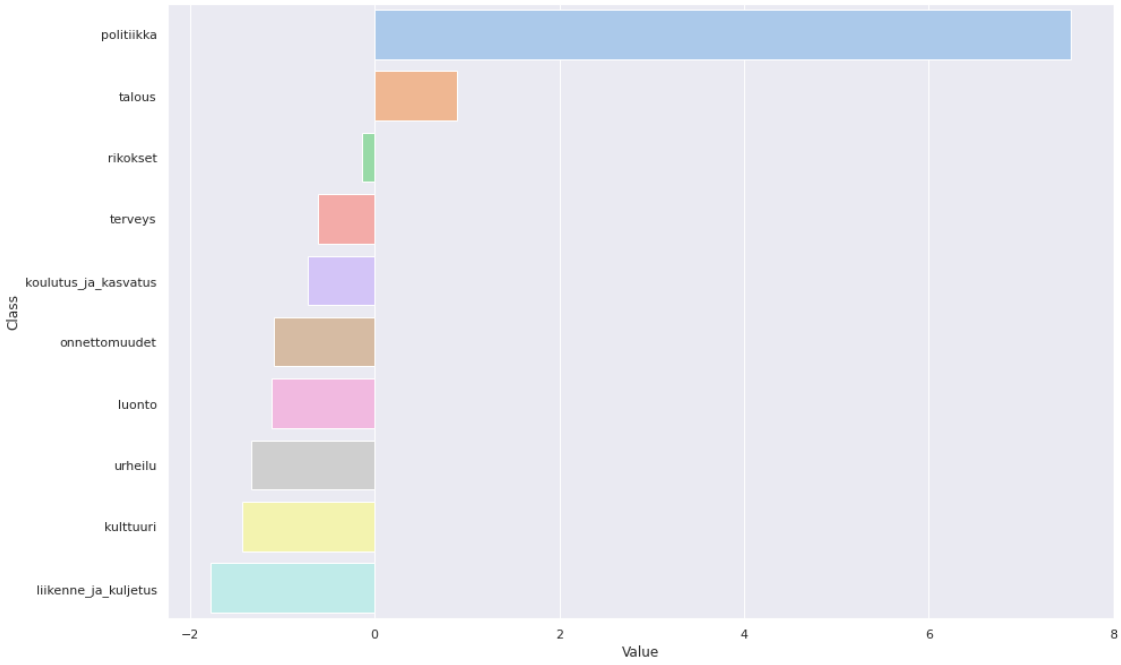
```
# Tokenisoidaan teksti
teksti = '''Biden uskoo voittavansa vaalit. Trump on väittänyt voittaneensa useissa vaa'an-
kieliosavaltioissa'''
teksti_token = tokenizer(teksti, return_tensors="pt")

# Tehdään ennuste (inference). Palauttaa tuplen
outputs = model(teksti_token['input_ids'], token_type_ids=teksti_token['token_type_ids'],
attention_mask=teksti_token['attention_mask'])

logits = outputs[0]

classValues = []
i = 0
for luokka in logits[0]:
  classValue = luokka.detach().numpy().flatten() # Tensor to numpy
  print(labelTexts[i], classValue)
  classValues.append(classValue[0])
  i = i + 1
```

Kuva 26. Luokittelija koodi

```
kulttuuri [-1.4357278]
liikenne_ja_kuljetus [-1.7759652]
luonto [-1.1124291]
politiikka [7.539601]
urheilu [-1.3352009]
talous [0.893491]
onnettomuudet [-1.093991]
rikokset [-0.13131955]
terveys [-0.61456215]
koulutus_ja_kasvatus [-0.72506076]
```

Kuva 27. Luokittelijan tuloste

Visualisointiin käytettiin Seaborn - kirjastoa, josta tulokset voidaan lukea helposti. Kuviosta 9 nähdään, että malli luokittelee lauseen ”*Biden uskoo voittavansa vaalit. Trump on väittänyt voittaneensa useissa vaa'ankieliosa-valtioissa*”, todennäköisimmin politiikkaan kuuluvaksi. Myös talouden painoarvo on positiivinen. Lauseesta voi myös ihminen nähdä selkeästi, mitkä sanat vaikuttavat luokitteluun eniten. Tässä kyseisessä lauseessa on selkeitä sanoja, kuten ”vaalit”, ”Biden” ja ”Trump”, jotka määrittävät luokaksi politiikan.



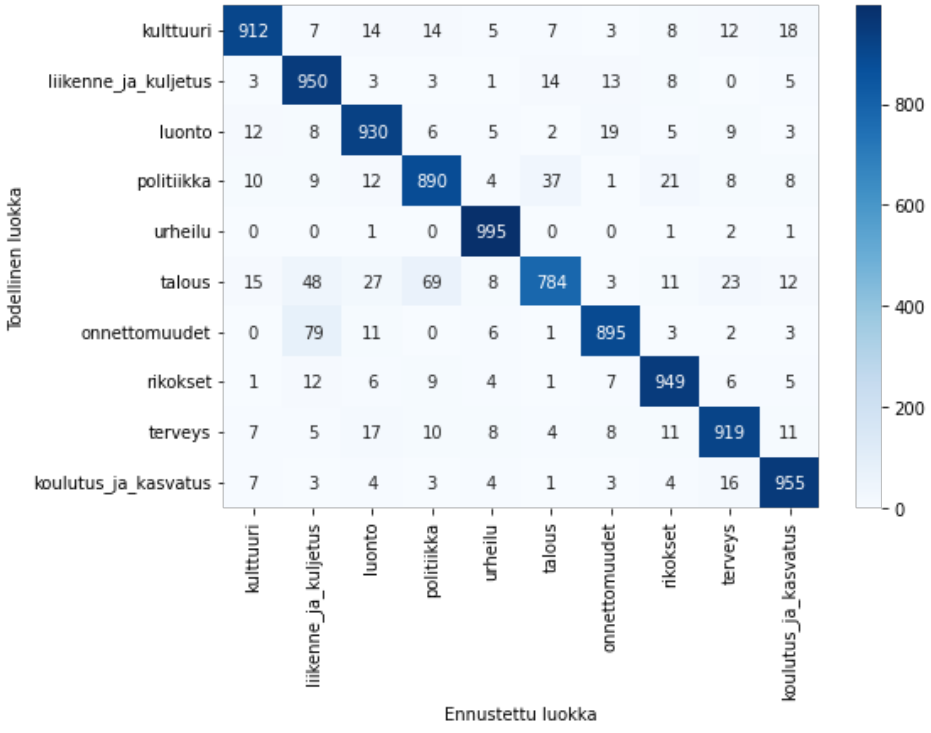
Kuvio 9. FinBERT-luokittelu

4.5.1 Mallin suorituskyvyn arviointi

Taulukosta 8 nähdään yhteenveto mallin tarkkuudesta sekä luokkakohtaiset tunnusluvut.

	precision	recall	f1-score	support
kulttuuri	0.94	0.91	0.93	1000
liikenne_ja_kuljetus	0.85	0.95	0.90	1000
luonto	0.91	0.93	0.92	999
politiikka	0.89	0.89	0.89	1000
urheilu	0.96	0.99	0.98	1000
talous	0.92	0.78	0.85	1000
onnettomuudet	0.94	0.90	0.92	1000
rikokset	0.93	0.95	0.94	1000
terveys	0.92	0.92	0.92	1000
koulutus_ja_kasvatus	0.94	0.95	0.95	1000
accuracy			0.92	9999
macro avg	0.92	0.92	0.92	9999
weighted avg	0.92	0.92	0.92	9999

Taulukko 8. FinBERT – luokitteluraportti



Kuvio 10. FinBERT-sekaannusmatriisi

Sekaannusmatriisista (Kuvio 10) voidaan helposti havaita mallin suorituskyky luokkakohtaisesti. Voidaan todeta, että urheilu on luokkana helpoin tunnistaa, joka johtuu luultavasti

luokalle tyypillisistä sanoista ja henkilöiden nimistä. Tietyt sanat esiintyvät vain ja ainoastaan urheiluluokassa. Voidaan havaita myös, että talous ja politiikka menevät usein sekaisin, sillä aiheet ovat myös reaalimaailmassa limittäin. Poliitikot antavat talouteen liittyviä lausuntoja, joten näiden luokkien henkilöihin liittyvät sanat ovat usein samoja. Toinen sekaannusta aiheuttava pari on onnettomuudet ja liikenne_ja_kuljetus. Onnettomuuksiin liittyvissä uutisissa on mainittu usein kulkuneuvoja, joten malli valitsee välillä väärän luokan todennäköisimmäksi vaihtoehdoksi. Näissä väärin luokitelluissa ennusteissa myös oikea luokka on saanut todennäköisesti myös positiivisen painokertoimen, mutta NumPy:n argmax jättää sen huomioimatta, sillä se valitsee vain suurimman todennäköisyyden saaneen luokan.

4.6 FastText-luokittelu

FastText mallin koulutukseen käytettiin Anacondassa virtuaalista ympäristöä. FastTextiä käytettiin Python-modulin kautta, mutta sitä on mahdollista ajaa myös sovelluksena, jolloin suorituskyky on entistä parempi.

Mallin koulutettiin alla olevalla koodilla, eli epochien määrä oli 25.

```
model = fasttext.train_supervised(input=f'{dataRoot}/yle-train.txt', epoch=25,
minn=3, maxn=5)
```

Kuva 28. Mallin koulutuskomento 25 epochilla

Myös ngramien pituutta rajoitettiin asettamalla minimiarvoksi 3 ja maksimiksi 5.

Malli käyttää ulostulokerroksessa Softmax-funktiota, joka laskee jokaiselle luokalle todennäköisyyden ja summaa todennäköisyydet niin, että niiden yhteenlaskettu summa on 1.

Mallin tarkkuutta testattiin testisetillä, joka antoi tarkkuudeksi 90,25 %, jota voidaan pitää hyvänä tuloksena. Malli kasvaa kuitenkin helposti suurikokoiseksi, kuten tässä tapauksessakin, sillä mallin lopullinen koko oli 1,53Gt. Tämän kokoisen mallin ajaminen vaatii laitteistoltakin jo paljon.

Tässä työssä yksi kiinnostuksen kohteita oli luokittelun ajaminen hyödyntäen AWS:n Lambda-funktioita, eli Serverless-funktioita, joissa massiivisten mallien käyttäminen ei ole mahdollista. FastText tarjoaakin juuri tähän ongelmaan ratkaisun, sillä mallin kokoa on helppo pienentää pakkaamalla, sillä FastText pitää huolen pakkaamiseen liittyen hyperparametrien säädöstä. Mallin tarkkuus toki laskee, kun mallin kokoa pienennetään radikaalisti, mutta sovelluksesta riippuen mallin laskenut tarkkuus ei ole välttämättä ongelma. Kokeilu, jossa mallin maksimikooksi asetettiin 2Mt ja epochien määräksi yksi, ajettiin kuvassa 29 esitetyllä komennolla.

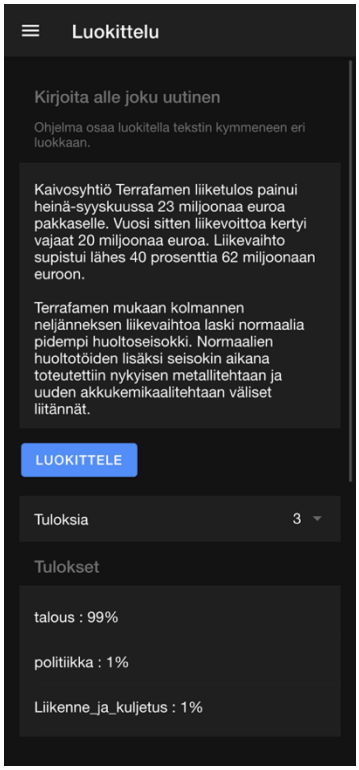
```
model = fasttext.train_supervised(input=f'{dataRoot}/yle-train.txt', epoch=1,
minn=3, maxn=5, autotuneValidationFile=f'{dataRoot}/yle-dev.txt', autotuneModel-
Size="2M")
```

Kuva 29. Mallin koulutuskomento yhdellä epochilla

Pakatun mallin tarkkuudeksi tuli 85,41 %, eli tulos oli erittäin hyvä, ottaen huomioon mallin radikaalisti pienentyneen koon. Pakkaaminen ei siis hukannut informaatiota ainakaan tämän tietoaineiston yhteydessä kovinkaan paljon.

4.6.1 Web-sovellus tekstin luokitteluun

Mallista tehtiin vielä kolmas versio, jossa mallin kooksi valittiin 10Mt, joka on sopivan kokoinen ajettavaksi AWS:n Lambda-funktiolla. Mallin tarkkuudeksi tällä mallilla tuli 89,7 %, jota voidaan pitää hyvänä tarkkuutena. Testausta varten kehitettiin myös koko web-sovellus Serverless-arkkitehtuurilla. Frontendinä käytettiin Googlen Firebasea ja framework frontissa oli Ionic Reactilla koodattuna. Backend oli vastaavasti AWS:n puolella, jossa tekoälylambda laitettiin API-Gatewayn taakse. AWS:n backendin kehitykseen käytettiin Serverless Frameworkia, jolla esimerkiksi fastText-kirjaston lisääminen Lambda-pakettiin oli melko vaivatonta. Myös koulutetun mallin lisääminen pakettiin onnistui Serverless Frameworkilla. Lambda-paketin lopullinen koko oli 75,6Mt, joten sitä ei voinut ladata suoraan lambda ”varastoon”, vaan sitä pitää käsitellä S3:n kautta, sillä suorien Lambda-pakettien kokorajoitus on 50Mt. Serverless Framework osaa tehdä automaattisesti oikean tyyppisen julkaisupaketin sen koon perusteella. S3:a hyödyntäen paketin maksimikoko on tällä hetkellä 250Mt. API:n authentikointiin käytettiin API-avainta. Rajapintaan laitettiin myös rajoitus kutsuille yhden päivän aikana, jotteivat esimerkiksi hyökkäykset API-rajapintaa kohtaan aiheuttaisi yllättäviä kustannuksia. Maksimi kutsujen määrä päivässä on 100 kappaletta. Nettisivu testausta varten löytyy osoitteesta https://serverlessai.web.app/page/Luokittelu. Alla kuvankaappaus esimerkkiluokittelusta.



Kuva 30. FastText-luokittelun tulos

Malli toimi todella hyvin, ja kuten Serverless-arkkitehtuuriin kuuluu, kustannukset syntyvät käytön mukaan. Eli jos Lambdaa ei kutsuta, ei synny myöskään kustannuksia, sillä palvelimia ei tarvitse pitää jatkuvasti päällä. Serverless tuo mukanaan kuitenkin myös ominaisuuksia, jotka vaikeuttavat käyttöä joissakin sovelluksissa. Kylmäkäynnistysaika Lambdalla vaihteli 5–6 sekunnin välillä, joten käyttäjät joutuvat ajoittain sietämään suhteellisen pitkää odottelua. Lambda toimii niin, että sitä kutsuttaessa AWS luo virtuaalisen ajoympäristön Lambdalle, joka on kuin pieni virtuaalinen tietokone. Ensimmäisellä kutsukerralla tuon ympäristön luomiseen kuluu hetki aikaa, joka esimerkiksi tässä kyseisessä tapauksessa on noin viisi sekuntia. AWS ei kuitenkaan tuhoa ympäristöä välittömästi kutsun jälkeen, vaan jättää sen pyörimään, mikäli Lambdaa kutsutaan hetken kuluttua uudelleen. Seuraavat kutsut ovatkin huomattavasti nopeampia, sillä itse Lambdan suoritusaika oli alle kolme millisekuntia. Kuvassa 31 on esimerkkiloki Lambdan ajosta niin, että mukana on myös kylmäkäynnistys.

```
START RequestId: 1782bd89-3ced-4e21-a35a-a476e8632650 Version: $LATEST
END RequestId: 1782bd89-3ced-4e21-a35a-a476e8632650
REPORT RequestId: 1782bd89-3ced-4e21-a35a-a476e8632650
Duration: 1.71 ms         Billed Duration: 100 ms
Memory Size: 256 MB       Max Memory Used: 255 MB
Init Duration: 3355.24 ms
```

Kuva 31. Lambda-funktion loki

4.6.2 Mallin suorituskyvyn arviointi

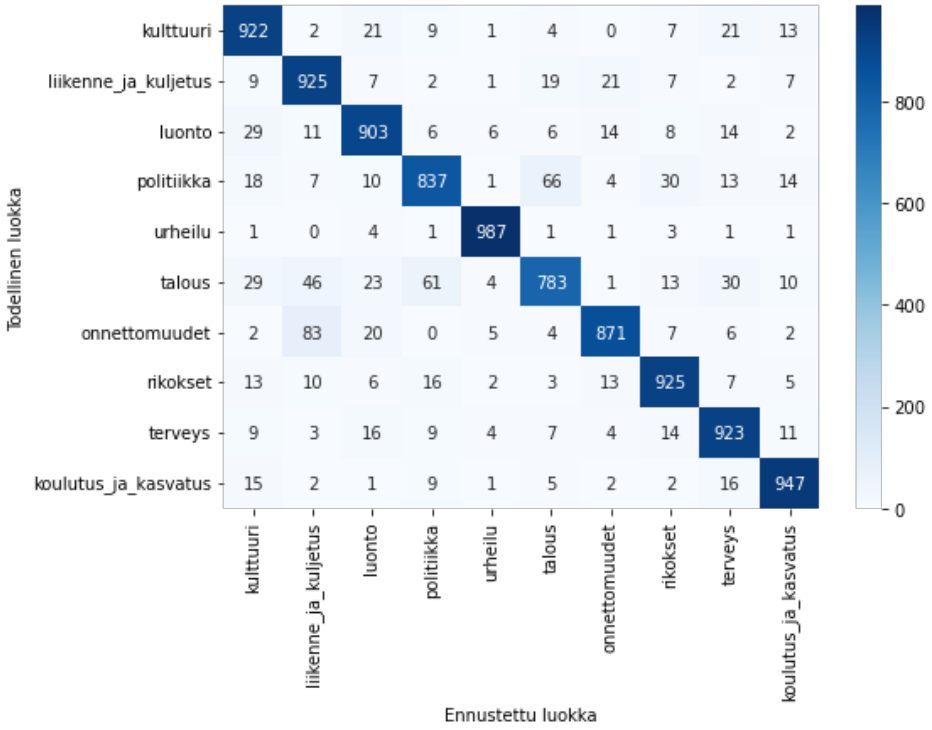
FastText-malli suoriutui luokittelusta erittäin hyvin, eikä mallin radikaali pakkaaminen heikentänyt tarkkuutta merkittävästi. 10Mt kokoinen (Taulukko 10) malli suoriutui luokittelusta lähes yhtä hyvin kuin pakkaamaton malli (Taulukko 9) ja se oli hieman tarkempi kuin monikielinen XLM-R-malli. Myös kuvioissa 11 ja 12 esitetyt sekaannusmatriisin tulokset olivat suorituskyvyltään hyvin lähellä toisiaan.

	precision	recall	f1-score	support
kulttuuri	0.88	0.92	0.90	1000
liikenne_ja_kuljetus	0.85	0.93	0.89	1000
luonto	0.89	0.90	0.90	999
politiikka	0.88	0.84	0.86	1000
urheilu	0.98	0.99	0.98	1000
talous	0.87	0.78	0.83	1000
onnettomuudet	0.94	0.87	0.90	1000
rikokset	0.91	0.93	0.92	1000
terveys	0.89	0.92	0.91	1000
koulutus_ja_kasvatus	0.94	0.95	0.94	1000
accuracy			0.90	9999
macro avg	0.90	0.90	0.90	9999
weighted avg	0.90	0.90	0.90	9999

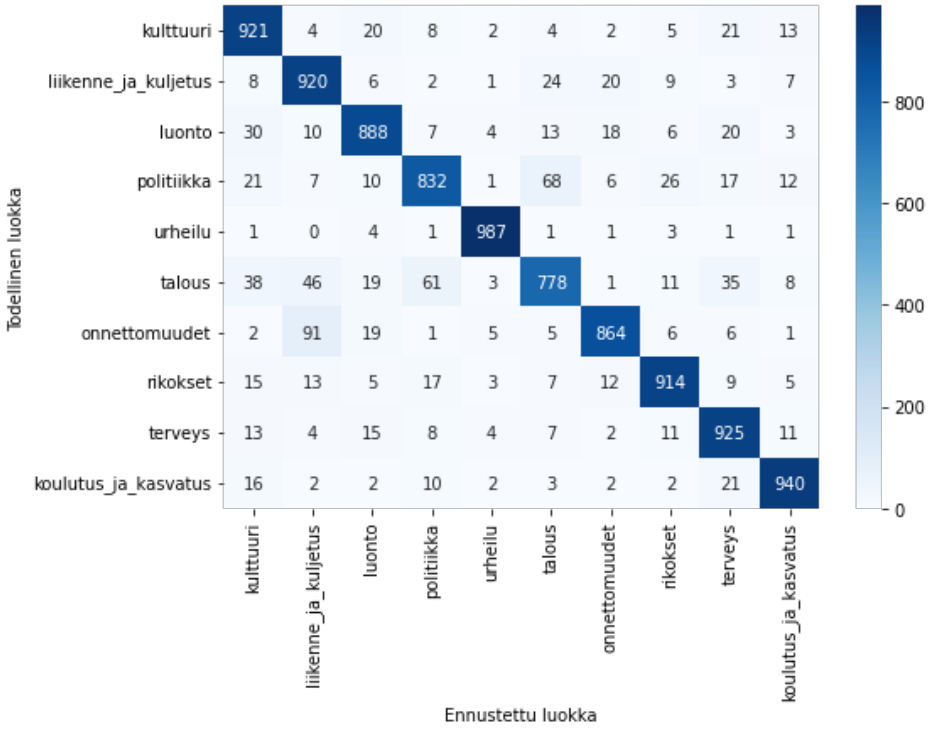
Taulukko 9. FastText-luokitteluraportti (1,53Gt malli)

	precision	recall	f1-score	support
kulttuuri	0.86	0.92	0.89	1000
liikenne_ja_kuljetus	0.84	0.92	0.88	1000
luonto	0.90	0.89	0.89	999
politiikka	0.88	0.83	0.85	1000
urheilu	0.98	0.99	0.98	1000
talous	0.85	0.78	0.81	1000
onnettomuudet	0.93	0.86	0.90	1000
rikokset	0.92	0.91	0.92	1000
terveys	0.87	0.93	0.90	1000
koulutus_ja_kasvatus	0.94	0.94	0.94	1000
accuracy			0.90	9999
macro avg	0.90	0.90	0.90	9999
weighted avg	0.90	0.90	0.90	9999

Taulukko 10. FastText-luokitteluraportti (10Mt malli)



Kuvio 11. FastText-sekaannusmatriisi (1,53Gt malli)



Kuvio 12. FastText-sekaannusmatriisi (10Mt malli)

4.7 XLM-R-luokittelu

XLM-R-mallin testaukseen käytettiin Colab-ympäristöä, kuten FinBERT-malliinkin, mutta Transformers-kirjastoa käytettiin SimpleTransformers-frameworkin kautta. Se yksinkertaisti koodia, mutta aiheutti myös uusia ongelmia kehityksessä, sillä framework on hyvin tuore. Se on myös erittäin tarkka riippuvien moduulien versioista, joten Colab-Notebookissa piti määritellä esimerkiksi NumPy:n versio eksplisiittisesti. Tarkasta määrittelystä huolimatta ilmeni kuitenkin satunnaisia varoituksia mahdollisista ristiriidoista pakettien välillä, mutta toimi kuitenkin ilmeisen virheettömästi.

Tietoaineisto, jolla malli koulutettiin, oli sama kuin edellisilläkin malleilla, eli 1000 datapistettä jokaisesta uutisluokasta. Colab-Notebookissa koulutus etenee seuraavasti:

1. Asennetaan yhteensopivat versiot NumPy:sta ja muista kirjastoista
2. Asennetaan SimpleTransformers ja wandb
3. Importoidaan tarvittavat kirjastot (pandas, NumPy, PyTorch)
4. Liitetään Google Drive virtuaaliseksi levyasemaksi (engl. *mount*)
5. Ladataan koulutus- ja validointidata ympäristöön
6. Esikäsitellään data pandas-dataframessa (target numeeriseksi, sarakkeet oikeaan järjestykseen)
7. Alustetaan malli esikoulutetulla XLM-R-mallilla. Kirjasto lataa oikean mallin automaattisesti Huggingfacen palvelimelta (xlm-roberta-base)
8. Koulutetaan malli. Kirjasto hoitaa automaattisesti kaikki datan esikäsittelyyn liittyvät tähtävät.
9. Arvioidaan koulutuksen onnistumista taulukon 11 perusteella.
10. Tallennetaan koulutettu malli Google Drive - asemalle. Mallin fyysinen koko oli noin 1Gt.

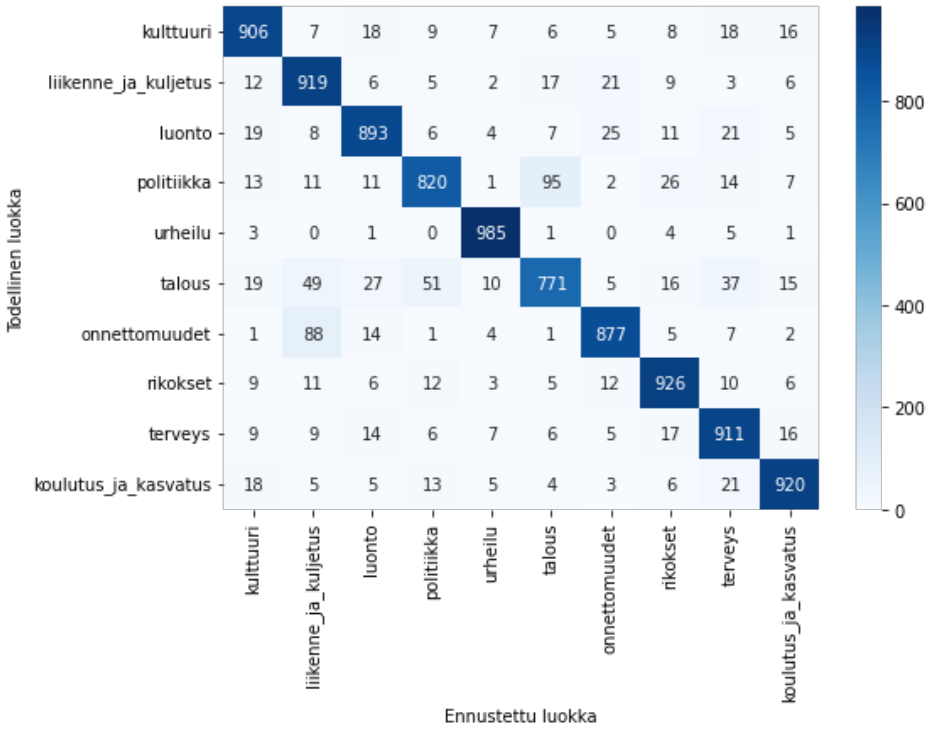
4.7.1 Mallin suorituskyvyn arviointi

XLM-R-malli suoriutui luokittelutehtävässä lähes yhtä hyvin kuin FinBERT, joten tulosta voi pitää erittäin hyvänä, sillä kyseessä on monikielinen malli. Tarkkuudessa malli jäi kokonaissuorituskyvyssä vain 3 prosenttiyksikköä FinBERT-mallia jälkeen. Taulukosta 11 nähdään mallin kokonaissuorituskyky ja luokkakohtaiset tulokset. Yksittäisistä luokista voidaan todeta mallin suoriutuvan huonommin erityisesti niissä luokissa, jotka tuottivat

ongelmia myös FinBERT-mallille. Eli talous, politiikka ja liikenne_ja_kuljetus olivat suhteellisesti huonommin tunnistettuja. Sekaannusmatriisista (Kuvio 13) voidaan havaita virheitä tapahtuneen enemmän juuri noissa luokissa.

	precision	recall	f1-score	support
kulttuuri	0.90	0.91	0.90	1000
liikenne_ja_kuljetus	0.83	0.92	0.87	1000
luonto	0.90	0.89	0.90	999
politiikka	0.89	0.82	0.85	1000
urheilu	0.96	0.98	0.97	1000
talous	0.84	0.77	0.81	1000
onnettomuudet	0.92	0.88	0.90	1000
rikokset	0.90	0.93	0.91	1000
terveys	0.87	0.91	0.89	1000
koulutus_ja_kasvatus	0.93	0.92	0.92	1000
accuracy			0.89	9999
macro avg	0.89	0.89	0.89	9999
weighted avg	0.89	0.89	0.89	9999

Taulukko 11. XLM-R - luokitteluraportti



Kuvio 13. XLM-R - sekaannusmatriisi

5 YHTEENVETO

NLP on tällä hetkellä erittäin paljon tutkittu tieteenala, jolloin uusia innovaatioita ja malleja syntyy jatkuvasti. Uudet mallit, jotka perustuvat neuroverkkoihin, ovat tarkempia ja tehokkaampia kuin aiemmat sääntöpohjaiset ratkaisut. Tällä hetkellä käydään myös kovaa kilpailua siitä, kuka pilviratkaisuja tarjoava yritys tarjoaa parhaat ratkaisut, joista on eniten hyötyä asiakkaiden liiketoiminnalle ja kuka pystyy tarjoamaan parhaan kehittäjäkokemuksen. Aktiivinen kehitys ja kova kilpailu tekevät NLP:stä mielenkiintoisen, mutta edelleen haastavan tieteenalan lähestyä. Kynnys NLP:n hyödyntämiseen ja edelleen kehittämiseen kuitenkin madaltuu jatkuvasti uusien teknologioiden myöstä. Esimerkiksi pilviratkaisuja tarjoavien yritysten uudet työkalut, kuten AWS:n Jumpstart, helpottavat yksinkertaisen sovelluksen luontia merkittävästi. Myös kolmansien osapuolien kehittämät kirjastot, kuten Huggingface, auttavat nykyaikaisten neuroverkkoarkkitehtuurien hyödyntämisessä.

Työssä tutkittiin suomen kielisen tekstin luokitteluun soveltuvia neuroverkkomalleja, joita olisi mahdollista ajaa niin sanotulla Serverless-arkkitehtuurilla, eli pilviympäristössä ei olisi jatkuvasti päällä olevaa palvelinta. Työn kolme vertailtavaa mallia olivat FinBERT, XLM-R ja FastText. Malleista luokittelussa tarkin oli FinBERT, toiseksi paras XLM-R ja epätarkin FastText. Malleja ei kuitenkaan voi suoraan vertailla, sillä mallien arkkitehtuurit ovat hyvin erityyppisiä. FinBERT on BERT-arkkitehtuuriin perustuva malli, joka on koulutettu suomen kielisellä aineistolla. XLM-R on vastaavasti monikielinen malli, joka on koulutettu sadalla eri kielellä. FastText on taas malli, joka on kehitetty erityisesti heikkotehoisissa laitteissa ajettavaksi. Näistä malleista vain FastText soveltuu ajettavaksi Serverless-tyyppisesti, sillä mallin koko saadaan puristettua niin pieneksi, että se on mahdollista ladata Serverless-ympäristössä ajettavaksi. Muita mallien ajaminen vaatii jatkuvasti päällä olevan virtuaalipalvelimen. Vaikka FastText oli epätarkin, ero muihin malleihin ei ollut suuri. 10 Mt FastText-mallin tarkkuus oli 0,90, kun taas vastaavasti tarkimman FinBERT-mallin tarkkuus oli 0,92. FinBERT-mallin fyysinen koko oli noin 500 Mt. Serverless-luokittelua testattiin 10 Mt FastText-mallilla ajamalla mallia Python ympäristössä fasttext-moduulilla Lambda-funktiossa. Kylmäkäynnistyksessä luokittelijan ajoaika oli noin 5 sekuntia, mutta seuraavat kutsut olivat lyhyille lauseilla noin 200 ms luokkaa.

Tässä työssä koulutukseen käytettiin Google Colab-ympäristöä, joka ei mahdollista suoraan ennustuspalvelimen luomista. Tuotantokäyttöön soveltuva ML-pipeline onkin syytä rakentaa sille suunnitellulle alustalle. Alustat tarjoavat kaiken koneoppimissovellukseen liittyvät komponentit, kuten datan hostaamisen ja esikäsittelyn, mallin koulutuksen ja hostaamisen sekä mallin tarkkuuden monitoroinnin. Tällaisia kokonaisvaltaisia ratkaisuja löytyy kaikilta suurilta pilvitoimittajilta, kuten Googlelta, Microsoftilta ja AWS:ltä. Tällä hetkellä

toimittajat kehittävät alustojaan erittäin aggressiivisesti ja uusia ominaisuuksia julkaistaan nopealla vauhdilla. Esimerkiksi AWS:n tekoälypalveluihin liittyvä keynote heidän vuosittaisessa Re:Invent-tapahtumassaan oli lähes yhtä pitkä kuin koko tapahtuman pää - keynote.

Tätä työtä voisi jatkokehittää tutkimalla nopeampia tapoja kouluttaa malli. Google Colab tarjoaa käyttöön myös TPU:n, jolla BERT - mallien koulutus nopeutuisi blogipostausten perusteella kymmeniä prosentteja. TPU:t tuovat myös lisätehoa ennusteiden tekemiseen, joten myös sillä puolella on mahdollista parantaa suorituskykyä. Tällä hetkellä tosin Serverless-funktiot eivät tarjoa ajoympäristönä TPU:n sisältämiä ympäristöjä, vaan prosessoreina on tarjolla vain CPU-ratkaisuja.

LÄHTEET

Arumugam, Rajesh, and Rajalingappaa Shanmugamani. 2018. Hands-On Natural Language Processing with Python. Packt Publishing.

AWS 2019. Deploy Your ML Models to Production at Scale with Amazon SageMaker. Youtube-video. Viitattu 16.11.2020. Saatavissa https://youtu.be/KFuc2KWrTHs?t=53

AWS 2020. AI & AWS DeepComposer. SlideShare-esitys. Viitattu 5.9.2020. Saatavissa https://www.slideshare.net/AmazonWebServices/ai-aws-deepcomposer/2

AWS 2020. Amazon SageMaker Developer Guide. Viitattu 15.9.2020. Saatavissa https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-dg.pdf

AWS 2020. Amazon Comprehend: Developer Guide (Versio: 28.9.2020). Viitattu 9.10.2020. Saatavissa https://docs.aws.amazon.com/comprehend/latest/dg/comprehend-dg.pdf

AWS 2020. Amazon SageMaker Model Monitor. Viitattu 10.9.2020. Saatavissa https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.htm

AWS 2020. AWS on Air 2020: AWS What's Next ft. Amazon SageMaker Feature Store. Youtube-video. Viitattu 12.1.2021. Saatavissa https://youtu.be/2egoqWYZ730?t=225

AWS 2020. Machine Learning with Amazon SageMaker. Viitattu 6.10.2020. Saatavissa https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-mlconcepts.html

AWS 2021. Amazon-sagemaker-examples. Viitattu 5.5.2021. Saatavissa https://github.com/aws/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_script_mode_training_and_serving/tensorflow_script_mode_training_and_serving.ipynb

Badr, Will. 2019. Accurately Automating Dataset Labelling Using Amazon SageMaker Ground-Truth. Youtube-video. Viitattu 8.9.2020. Saatavissa https://youtu.be/8J7y513oSsE?t=444

Chalmers, David. 2020. Wikipedia-artikkeli. Viitattu 16.10.2020. https://en.wikipedia.org/wiki/David_Chalmers

Code Emporium. 2020. Transformer Neural Networks - EXPLAINED! (Attention is all you need). Youtube-video. Viitattu 25.9.2020. Saatavissa. https://youtu.be/TQQlZhbC5ps?t=206

Conneau, Alexis, and Kartikay Khandelwal. 2020. Unsupervised Cross-lingual Representation Learning at Scale. Facebook AI.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Tutkimusartikkeli. Viitattu 20.10.2020. Saatavissa https://arxiv.org/abs/1810.04805

Dirac, Leo. 2019. LSTM is dead. Long Live Transformers!. Youtube-video. Viitattu 20.10.2020. Saatavissa https://youtu.be/S27pHKBEp30?t=391

Facebook Inc. 2020. Language identification. fastText-tuotesivu. Viitattu 24.9.2021. Saatavissa https://fasttext.cc

Facebook Inc. 2017. Language identification. fastText-tuotesivu. Viitattu 2.10.2020. Saatavissa https://fasttext.cc/blog/2017/10/02/blog-post.html

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning Book. MIT Press. Viitattu 10.9.2020. Saatavissa http://www.deeplearningbook.org

Google Research. 2020. BERT. Github-tietovarasto. Viitattu 14.9.2020. Saatavissa https://github.com/google-research/bert

Jackson, Peter, ja Isabelle Moulinier. 2002. Natural Language Processing for Online Applications Text Retrieval, Extraction and Categorization. Homson Legal & Regulatory.

Julien Simon. 2020. Amazon SageMaker Clarify Detects Bias and Increases the Transparency of Machine Learning Models. AWS News-blogi. Viitattu 12.1.2021. Saatavissa https://aws.amazon.com/blogs/aws/new-amazon-sagemaker-clarify-detects-bias-and-increases-the-transparency-of-machine-learning-models

Lukka, Kari. 2001. Konstruktiivinen tutkimusote. Metodiopas verkossa Viitattu 25.11.2020. Saatavissa https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote

McCormick, Chris. 2020. BERT Research - Ep. 1 - Key Concepts & Sources. Youtube-video. Viitattu 27.8.2020. Saatavissa https://www.youtube.com/watch?v=FKlPCK1uFrc&pbjreload=101

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. Viitattu 11.9.2020. Saatavissa https://arxiv.org/abs/1301.3781

Ng, Andrew. 2020. Sequence Models. Coursera-luento. Viitattu 15.9.2020. Saatavissa https://www.coursera.org/learn/nlp-sequence-models

Ng, Andrew. 2020. Sequence Models, Different types of RNNs. Coursera-luento. Viitattu 15.9.2020. Saatavissa https://www.coursera.org/learn/nlp-sequence-models/lecture/BO8PS/different-types-of-rnns

Olah, Christopher. 2020. Understanding LSTM Networks. Blogiteksti. Viitattu 20.9.2020. Saatavissa http://colah.github.io/posts/2015-08-Understanding-LSTMs

OpenAI. 2020. OpenAI API. Blogiteksti. Viitattu 1.11.2020. Saatavissa https://openai.com/blog/openai-api

OpenAI. 2020. Language Models are Few-Shot Learners. Tutkimusartikkeli. Viitattu 1.11.2020. Saatavissa https://arxiv.org/abs/2005.14165

Stanford NLP Group. 2020. SQuAD2.0 - The Stanford Question Answering Dataset. Tuotesivu. Viitattu 28.9.2020. Saatavissa https://rajpurkar.github.io/SQuAD-explorer/explore/v2.0/dev/Normans.html

Sokolova, Marina, and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. Information Processing and Management 8.5.2009, 427-437.

TensorFlow.org. 2020. Word embeddings. Tutoriaali. Viitattu 21.8.2020. Saatavissa https://www.tensorflow.org/tutorials/text/word_embeddings

TurkuNLP. 2020. Finnish NER. Verkkosivu. Viitattu 28.9.2020. Saatavissa https://turkunlp.org/finner.html

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. Tutkimusartikkeli. Viitattu 16.10.2020. Saatavissa https://arxiv.org/abs/1706.03762

Virtanen, Antti, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: BERT for Finnish. Turku NLP group, University of Turku.

Virtanen, Antti, Sampo Pyysalo, and Filip Ginter. 2020. Training BERT from scratch (a brief tutorial). Luentoesitys. Viitattu 5.11.2020. Saatavissa http://svn.emmtee.net/outreach/skeikampen/2020/finbert.pdf

Wongviboonsin, Paton. 2020. Question Answering with PyTorch Transformers: Part 1. Tutoriaali. Viitattu 28.9.2020. Saatavissa https://medium.com/@patonw/question-answering-with-pytorch-transformers-part-1-8736196bf20e

Xin, Rong. 2016. word2vec Parameter Learning Explained. Tutkimusartikkeli. Viitattu 20.9.2020. Saatavissa https://arxiv.org/abs/1411.2738