Johdatus älykkään data-analyysin prosessimalliin, luonnollisen kielen käsittelyyn, koneoppimiseen sekä niihin perustuvien käyttötapausten kehittämiseen Valtorin hallinnoiman tekoälyalustan päälle

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos

Pro gradu -tutkielma

Data-analytiikka

Lokakuu 2020

Marita Risku

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos
MARITA RISKU: Johdatus älykkään data-analyysin prosessimalliin, luonnollisen kielen käsittelyyn, koneoppimiseen sekä niihin perustuvien käyttötapausten kehittämiseen Valtorin hallinnoiman tekoälyalustan päälle

Pro gradu -tutkielma, 86 s., 47 liites.
Data-analytiikka
Lokakuu 2020

Valtion tieto- ja viestintätekniikkakeskus Valtorin ja Valtion talous- ja henkilöstöhallinnon palvelukeskus Palkeiden meneillään olevassa yhteishankkeessa on tavoitteena parantaa asiakaspalvelua tekoälyn avulla. Molempien organisaatioiden palveluprosesseista on tunnistettu käyttötapauksia, joita olisi mahdollista automatisoida ja tehostaa tekoälyä hyödyntäen. Käyttötapausten toteuttamiseksi on hankittu Valtorin hallinnoima tekoälyalusta, jota on tulevaisuudessa tarkoitus hyödyntää myös Valtorin asiakkaiden tekoälyyn pohjautuvien käyttötapausten alustana. Tekoälyalusta koostuu IT-infrastruktuurista ja sen päällä ajettavasta tekoälyratkaisusta. Tavoitteena on, että tulevien uusien käyttötapausten kehittäminen tekoälyalustan päälle olisi mahdollisimman yksinkertaista, nopeaa ja kustannustehokasta.

Tämän opinnäytetyön tavoitteena on toimia julkisena perehdytysmateriaalina uusien koneoppimiseen perustuvien käyttötapausten rakentamiseksi Valtorin hallinnoiman tekoälyalustan päälle. Keskityn kuvaamaan niitä koneoppimiseen liittyviä teorioita ja käytäntöjä, joiden katson olevan oleellisia menestyksellisten koneoppimista hyödyntävien IT-projektien läpiviennissä valtionhallinnossa Valtorin tekoälyalustaa hyödyntäen.

Opinnäytetyö sisältää johdatuksen älykkään data-analyysiin prosessimalliin, luonnollisen kielen käsittelyyn, koneoppimiseen ja neuroverkkoihin. Näiden teoriapainotteisten lukujen jälkeen esittelen konkreettisen esimerkin avulla tekstidokumenttien luokitteluun käytettävien ennustemallien kehittämistä tutkivana prosessina. Valtorin tekoälyalustasta kerron sillä tarkkuudella, mikä on julkisessa dokumentissa mahdollista. Esimerkkinä käyttötapauksen rakentamisesta tekoälyalustan päälle käytän Valtorin omaa ’Tikettien luokittelu ja ohjaaminen oikeaan työjonoon’ -käyttötapausta. Lopuksi pohdin tulevaisuuden kehittämistarpeita ja teen myös ehdotuksen Valtiovarainministeriön tekoälyä hyödyntäviä hankkeita koskevien erityisrahoitushakujen kehittämiseksi.

Asiasanat: data-analyysin prosessimalli, luonnollisen kielen käsittely, NLP, koneoppiminen, neuroverkot

Sisällys

Käytetyt lyhenteet

AINO	Valtorin tekoälyratkaisun ns. perinteisen tietojärjestelmän osuus, joka ei sisällä koneoppimista (AI no)
AION	Valtorin tekoälyratkaisussa koneoppimismallien kehittämiseen käytettävä tietojärjestelmä (AI on)
AutoML	Automatisoitu koneoppiminen (Automated Machine Learning)
BERT	Googlen tutkijoiden kehittämä kielimalli **(**Bidirectional Encoder Representations from Transformers)
CALL	Toiminnanohjausjärjestelmän tikettityyppi, kun ei vielä tiedetä, onko kyseessä palvelupyyntö vai häiriö
CNN	Konvoluutioneuroverkko (Convolutional Neural Network)
CRISP-DM	Tiedonlouhinnan prosessimalli (CRoss Industry Standard Process for Data Mining)
df	Dokumenttifrekvenssi
fastText	Facebookin tutkijoiden kehittämä ohjelmistokirjasto sanaupotusten muodostamiseksi
FinBERT	Turun yliopiston tutkijoiden kouluttama suomenkielinen BERT-kielimalli
GloVe	Stanfordin yliopiston tutkijoiden kehittämä algoritmi sanaupotusten muodostamiseksi (Global Vectors)
GPU	Grafiikkasuoritin (Graphical Processing Unit)
INC	Toiminnanohjausjärjestelmän tikettityyppi, joka tehdään häiriöstä (Incident)
Keras	TensorFlown päälle kehitetty käyttäjäystävällisempi ohjelmointirajapinta
Kubeflow	Valtorin tekoälyalustalla käytettävä koneoppimiseen liittyvien ajojen orkestraattori
M-BERT	Googlen esikouluttama monikielinen BERT-kielimalli (Multilingual BERT)
NER	Nimettyjen entiteettien tunnistaminen (Named Entity Recognition)
NLP	Luonnollisen kielen käsittely (Natural Language Processing)
Palkeet	Valtion talous- ja henkilöstöhallinnon palvelukeskus Palkeet

PCA	Pääkomponenttianalyysi (Principal Component Analysis)
ReLU	Suosittu neuroverkoissa käytetty aktivointifunktio (Rectified Linear Unit)
RITM	Toiminnanohjausjärjestelmän tikettityyppi, joka tehdään palvelupyynnöstä (Requested Item)
SOM	Ohjaamattoman koneoppimisen menetelmä klusterointiin ja datan visualisointiin (Self-Organizing Map)
TensorFlow	Googlen kehittämä ilmainen avoimen lähdekoodin koneoppimiskirjasto
TFX	TensorFlow Extended. TensorFlow:hun perustuva kokonaisvaltainen tuotantoputki koneoppimismallien kehittämiseksi, tuotantoon viemiseksi ja monitoroinniksi
TOP	Valtorin käyttämä Service Now -toiminnanohjausjärjestelmä
TPU	Googlen kehittämä tensorisuoritin koneoppimisen ja erityisesti neuroverkkojen vaatimaan laskentaan (Tensor Processing Unit)
tf	Termifrekvenssi
tf_idf	Termifrekvenssi jaettuna dokumenttifrekvenssillä
Valtori	Valtion tieto- ja viestintätekniikkakeskus Valtori
word2vec	Googlen tutkijoiden kehittämä ohjelmistokirjasto sanaupotusten muodostamiseksi

1. Johdanto

Valtion tieto- ja viestintätekniikkakeskus Valtori (jäljempänä ”Valtori”) tuottaa valtionhallinnon toimialariippumattomat ICT-palvelut sekä korkean varautumisen ja turvallisuuden vaatimukset täyttäviä tieto- ja viestintäteknisiä palveluja ja integraatiopalveluja. Valtorin asiakkaita ovat muun muassa kaikki valtionhallinnon virastot ja laitokset. [1]

Valtorin ja Valtion talous- ja henkilöstöhallinnon palvelukeskus Palkeiden (jäljempänä ”Palkeet”) meneillään olevassa yhteishankkeessa on tavoitteena parantaa asiakaspalvelua tekoälyn avulla. Molempien organisaatioiden palveluprosesseista on tunnistettu käyttötapauksia, joita olisi mahdollista automatisoida ja tehostaa tekoälyä hyödyntäen. Käyttötapausten toteuttamiseksi on hankittu Valtorin hallinnoima tekoälyalusta, jota on tulevaisuudessa tarkoitus hyödyntää myös Valtorin asiakkaiden tekoälyyn pohjautuvien käyttötapausten alustana. Tekoälyalusta koostuu IT-infrastruktuurista ja sen päällä ajettavasta tekoälyratkaisusta. Tekoälyalustan hankinnan tavoitteena on, että tulevien uusien käyttötapausten kehittäminen sen päälle olisi mahdollisimman yksinkertaista, nopeaa ja kustannustehokasta. Hankkeen yhtenä keskeisenä tavoitteena on myös tekoälyn hyödyntämiseen liittyvän osaamisen kehittäminen Valtorissa ja Palkeissa. [2]

Tekoälylle tyypillisiä ominaisuuksia ovat autonomisuus ja adaptiivisuus. Tekoälysovellukset pystyvät suorittamaan tehtäviä monimutkaisessakin ympäristössä autonomisesti, ilman jatkuvaa ohjausta. Ne pystyvät myös parantamaan suoritustaan oppimalla kokemuksesta. [3]

Tekoälysovellukset käyttävät oppimiseen koneoppimismalleja. Koneoppimisalgoritmit pystyvät oppimaan datasta funktioita, joita olisi liian vaikeaa tai kallista ohjelmoida käsin. Neuroverkot ovat yhdentyyppisiä koneoppimismalleja. Useita kerroksia sisältävien eli niin sanottujen syvien neuroverkkojen yhteydessä koneoppimista kutsutaan syväoppimiseksi.

Tämän opinnäytetyön tavoitteena on toimia perehdytysmateriaalina uusien koneoppimiseen perustuvien käyttötapausten rakentamiseksi Valtorin hallinnoiman tekoälyalustan päälle. Valtionhallinnossa ja muulla julkisella sektorilla työskentelee tuhansia IT-alan ammattilaisia, joille erilaiset IT-projektit ovat arkipäivää. Koneoppimista hyödyntävien sovellusten rakentamisprojektit ovat paljolti tavallisia IT-projekteja, mutta sen lisäksi koneoppimisen hyödyntäminen vaatii omaa erityisosaamistaan.

Keskityn tässä opinnäytetyössä kuvaamaan niitä koneoppimiseen liittyviä teorioita ja käytäntöjä, joiden katson olevan oleellisia menestyksellisten koneoppimista hyödyntävien IT-projektien läpiviennissä valtionhallinnossa Valtorin tekoälyalustaa hyödyntäen. Koneoppimisen perinpohjainen

ymmärtäminen vaatisi muun muassa lineaarialgebran, todennäköisyyslaskennan, informaatioteorian ja numeerisen laskennan osaamista. Yritän kuitenkin välttää liian matemaattista lähestymistä ja pysyä IT-ammattilaisten mukavuusalueella. Koska kyseessä on perehdytysmateriaali, käytän eksaktin akateemisen kirjoitustyylin rinnalla myös vapaammin kuvailevaa ilmaisutapaa. Jotta eri lukuja voisi opiskella myös itsenäisinä kokonaisuuksina, luvuissa on jonkin verran toistoa.

Projekteissa tarvitaan koneoppimisen perustietämyksen lisäksi myös siihen liittyvää syvää erityisosaamista ja usein tarvittava erityisosaaminen hankitaan kilpailuttamalla asiantuntijapalveluja. Tavoitteenani on, että tämän perehdytysmateriaalin avulla olisi mahdollista saavuttaa riittävä osaamisen taso ulkopuolisten asiantuntijoiden hankkimiseksi ja ohjaamiseksi. Lisäksi jokaisella projektiin osallistuvalla on hyvä olla perustason ymmärrys projektissa tarvittavista koneoppimisen osa-alueista.

Käsiteltävän aiheen laajuuden vuoksi olen joutunut priorisoimaan kattavuuden syvyyden edelle. Pidän kattavan suomenkielisen perehdytysmateriaalin kirjoittamista tärkeänä, koska olen itse löytänyt hyvin vähän suomenkielistä opetusaineistoa koneoppimisesta. Mikäli lukija haluaa syventää osaamistaan, se on mahdollista esimerkiksi lähdeluettelosta löytyvän englanninkielisen materiaalin avulla. Toivon, että opinnäytetyöni opiskelulla on mahdollisuus saavuttaa sellainen perustason ymmärrys, että aiheen jatko-opiskelu osaamisen syventämiseksi helpottuisi merkittävästi.

Koska suomenkielistä materiaalia on niin vähän saatavilla, olen joutunut suomentamaan termejä osin omavaltaisesti. Koska suomen kieltä käytetään koneoppimisen yhteydessä hyvin harvoin, olen pyrkinyt lisäämään suomenkielisten termien esittelyn yhteyteen aina myös niiden alkuperäiset englanninkieliset vastineet.

Aloitan kuvaamalla luvussa kaksi älykkään data-analyysin prosessimallin. Data-analyysiprosessin tärkeimpiä tavoitteita on kehittää haluttua soveltamiskohdetta mahdollisimman luotettavasti ennustava tai selittävä koneoppimismalli, sekä evaluoida kehitetyn mallin kyvykkyys suhteessa sille asetettuihin tavoitteisiin. Vasta tämän prosessin evaluointivaiheen tulosten perusteella tiedetään, onko tavoiteltavia liiketoimintahyötyjä mahdollista saavuttaa käyttäen hyväksi käytettävissä olevaa dataa ja koneoppimismenetelmiä. Lopullinen päätös koneoppimista hyödyntävien käyttötapausten rakentamisprojektin käynnistämisestä tulisikin tehdä vasta tämän analyysin jälkeen.

Koska suurin osa valtionhallinnossa käsiteltävästä tiedosta on tekstimuotoista, esittelen luvussa kolme luonnollisen kielen käsittelyn menetelmiä, joiden avulla tekstimuotoinen aineisto on muunnettavissa koneoppimismallien ymmärtämään numeeriseen muotoon. Vasta tämän jälkeen, luvussa neljä, perehdytään tarkemmin itse koneoppimiseen ja luvussa viisi neuroverkkoihin. Näiden

teoriapainotteisten lukujen jälkeen, luvussa kuusi, esittelen konkreettisen esimerkin avulla tekstidokumenttien luokitteluun käytettävien ennustemallien kehittämistä tutkivana prosessina.

Luvussa seitsemän kerron Valtorin tekoälyalustasta sillä tarkkuudella, mikä on julkisessa dokumentissa mahdollista. Luvussa kahdeksan käytän Valtorin omaa käyttötapausta ’Tikettien luokittelu ja ohjaaminen oikeaan työjonoon’ esimerkkinä koneoppimiseen perustuvan käyttötapauksen kehittämisestä Valtorin tekoälyalustan päälle. Viimeisessä kappaleessa pohdin Valtorin tekoälyalustaan ja toimintatapoihin liittyviä tulevaisuuden kehitystarpeita ja teen myös ehdotuksen Valtiovarainministeriön tekoälyä hyödyntäviä hankkeita koskevien erityisrahoitus-hakujen kehittämiseksi.

Valtorin tekoälyalustaan kuuluu erillisenä tuotteena myös chatbot-palveluiden kehittämisen ja tuottamisen ohjelmistoratkaisu. Kyseessä on kaupallinen Boost.ai:n tuote, jonka sisäinen toiminta on liiketoimintasalaisuus ja jonka kokonaispalveluun kuuluu kattava koulutus. Chatbotin kehittäminen on hyvin erilainen projekti kuin muiden koneoppimista hyödyntävien käyttötapausten rakentaminen. Tämän vuoksi chatbot-tuotteen päälle rakennettavat käyttötapaukset on rajattu tämän opinnäytetyön ulkopuolelle. Luonnollisen kielen käsittelyä kuvaava kappale antaa kuitenkin perusteoriatietoa myös chatbotin toiminnan ymmärtämiseksi.

2. Älykkään data-analyysin prosessimalli

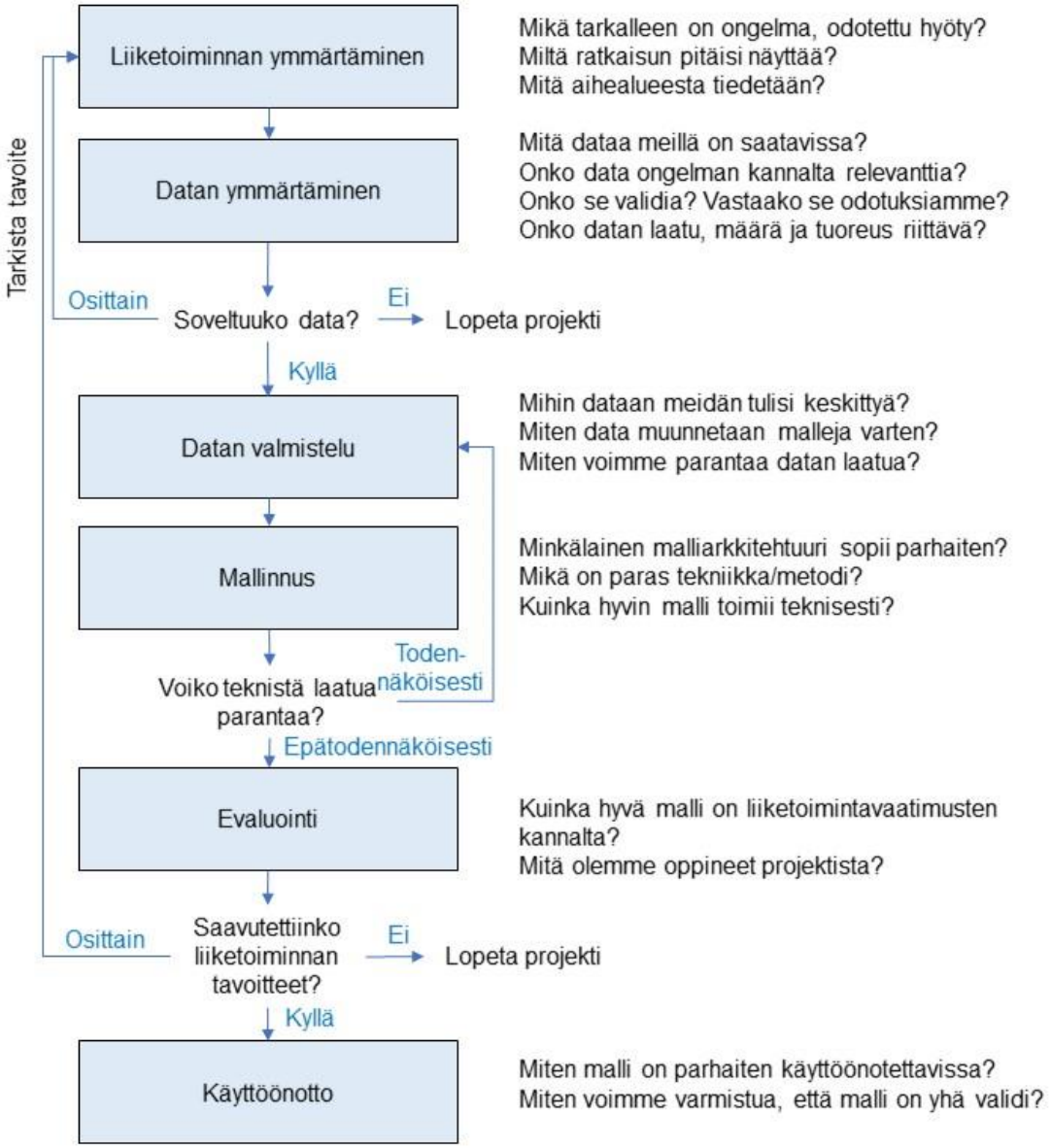
Älykäs data-analyysi (intelligent data analysis) on Berthold et al.:in kirjassa [4] käytetty termi, jolla haluan painottaa ihmisten asiantuntemuksen, älykkyyden ja intuition merkitystä tiedonlouhinnan (data mining) prosessissa. Jokainen projekti on soveltamiskohteeltaan, toimintaympäristöltään ja tavoitteiltaan erilainen. Siksi teknisten edellytysten sekä mahdollisesti hyvinkin automatisoitujen tiedonlouhintatyökalujen käytön lisäksi tarvitaan myös ihmisten älykkyyttä, jotta saatavissa olevasta datasta saataisiin puristettua paras mahdollinen hyöty.

Data-analyysiprosessin tärkeimpiä tavoitteita on kehittää mahdollisimman hyvin haluttua asiaa ennustava tai selittävä koneoppimismalli, sekä evaluoida kehitetyn mallin kyvykkyys suhteessa sille asetettuihin tavoitteisiin. Prosessi on luonteeltaan tutkivaa ja iteratiivista. Dataa tutkimalla ja mallintamalla on mahdollista saada dataan perustuvaa uutta tietoa käsiteltävästä aiheesta. Aiheeseen perehtynyt asiantuntija voi saada tästä uusia oivalluksia tai hän saattaa antaa uudelle tiedolle syvällisemmän tulkinnan. Uuden ymmärryksen valossa on mahdollista palata edellisiin vaiheisiin entistä viisaampana.

Vasta tämän prosessin evaluointivaiheen tulosten perusteella tiedetään, onko tavoiteltavia liiketoimintahyötyjä mahdollista saavuttaa käyttäen hyväksi käytettävissä olevaa dataa ja koneoppimismenetelmiä. Lopullinen päätös koneoppimista hyödyntävien käyttötapausten rakentamisprojektin käynnistämisestä kannattaisikin tehdä vasta tämän analyysin jälkeen.

Tiedon louhinnan prosessimalleja ovat mm. SEMMA (Sample, Explore, Modify, Model, Asses), KDD (Knowledge Discovery in Databases) ja CRISP-DM (CRoss Industry Standard Process for Data Mining). Näistä CRISP-DM näyttäisi olevan laajimmin käytetty. [4]

CRISP-DM on kuvattu seikkaperäisesti CRISP-DM konsortion englanninkielisessä julkaisussa [5], johon tämä luku 2 perustuu. Julkaisussa käytetyn termin ’tiedonlouhinta’ olen korvannut termillä ’älykäs data-analyysi’. Kuvassa 2.1 on esitetty yleiskatsaus CRISP-DM prosessimalliin ja sen eri vaiheissa esitettävät tyypilliset kysymykset [4]. CRISP-DM prosessimallissa ensimmäisen vaiheen nimi on liiketoiminnan ymmärtäminen (business understanding) [5]. Tätä vaihetta voidaan kutsua myös projektin ymmärtämiseksi (project understanding) [4].



Kuva 2.1: Yleiskatsaus CRISP-DM prosessimalliin ja sen eri vaiheissa esitettävät tyypilliset kysymykset. Kuva on suomennettu Berthold et al.:in kirjasta [4] sivulla 9 esitetystä kuvasta.

CRISP-DM prosessimallissa on kuusi eri vaihetta. Olen itse jakanut prosessimallin viimeisen eli käyttöönottovaiheen kahteen osaan erottamalla monitoroinnin ja ylläpidon omaksi erilliseksi vaiheekseen. Haluan näin painottaa niiden tärkeyttä. Liiketoimintahyödyt realisoituvat yleensä pitkän ajan kuluessa. Koska kuitenkin elämme jatkuvassa muutoksessa, myös koneoppimismalli on pidettävä niistä ajan tasalla. Muuten on riskinä, että mallin kyvykkyys heikkenee hyvinkin nopeasti. Mikäli näin pääsee tapahtumaan, jäävät liiketoimintahyödyt saavuttamatta ja pahimmassa tapauksessa vanhentuneen koneoppimismallin käytön jatkaminen voi tuottaa vakaviakin ongelmia.

Älykkään data-analyysin prosessimallin seitsemän eri vaihetta (CRISP-DM prosessimallin kuusi vaihetta, joista viimeinen on jaettu kahteen osaan):

1. Liiketoiminnan ymmärtäminen
2. Datan ymmärtäminen
3. Datan valmistelu
4. Mallinnus
5. Evaluointi
6. Käyttöönotto
7. Monitorointi ja ylläpito

Koska älykäs data-analytiikka on luonteeltaan tutkivaa, prosessimalliin kuuluu, että vaiheiden välillä siirrytään tarvittaessa edestakaisin sen mukaan, mitä tuloksia niissä saavutetaan. Projekti on myös mahdollista keskeyttää, mikäli käytettävissä oleva data ei sovellu kyseiseen tarkoitukseen tai mallin ennustekyvykkyys evaluoidaan tavoitteiden kannalta riittämättömäksi. Seuraavissa aliluvuissa vaiheet kuvataan tarkemmin. Kuvaukset ovat pääosin tiivistelmiä CRISP-DM konsortion englanninkielisestä julkaisusta [5].

2.1. Liiketoiminnan ymmärtäminen

Ihan ensimmäiseksi keskitytään projektin tavoitteiden ja vaatimusten ymmärtämiseen liiketoiminnan näkökulmasta. On tärkeää, että projektille määritellään onnistumisen kriteerit liiketoimintalähtöisesti. Jos jotkin onnistumisen kriteerit vaativat ihmisen tai ihmisten tekemää arviota, nimetään nämä arvioijat. Kun liiketoiminnan tavoitteet ja vaatimukset ovat tiedossa, niistä voidaan johtaa data-analyysin teknisemmät vaatimukset ja onnistumisen kriteerit.

Samalla selvitetään muut tekijät, jotka vaikuttavat analyysitavoitteen asettamiseen ja projektisuunnitelman tekoon. Tällaisia ovat esimerkiksi käytettävissä olevat resurssit (ihmiset, data, tietojärjestelmät), milloin analyysin tulisi olla valmis, tietosuoja- ja tietoturvavaatimukset, lain tuomat ja eettiset rajoitteet, muut rajaukset, tehdyt oletukset ja riskit. Jo tässä vaiheessa olisi hyvä perustaa myös projektin sanasto, jossa huomioidaan niin liiketoiminnan käyttämä termistö kuin data-analyysiin ja koneoppimiseen liittyvä sanasto.

Kerättyjen tietojen perusteella valmistellaan data-analyysin projektisuunnitelma. Tähän sisältyy myös kustannus-hyöty-analyysin tekeminen, jossa projektin kustannuksia verrataan potentiaalisiin liiketoimintahyötyihin. Koska älykäs data-analyysi on luonteeltaan tutkivaa, on projektisuunnitelmaa tärkeä tarkistaa jokaisen vaiheen päätteeksi.

2.2. Datan ymmärtäminen

Datan ymmärtämisvaihe aloitetaan keräämällä ensimmäinen datasetti. Yleensä tämä vaatii datan kopioinnin ympäristöön, jossa data-analyysi on tarkoitus suorittaa. Sekä ympäristön että datan on siis täytettävä tietosuoja- ja tietoturvavaatimukset asianmukaisesti. Jotta datan kerääminen olisi mahdollista, jo tämä vaihe saattaa vaatia onnistuakseen datan valmistelua.

Kerättyyn dataan tutustutaan ja sen sisältö kuvataan. Dataa voidaan tutkia esimerkiksi visualisoimalla ja laskemalla siitä erilaisia tunnuslukuja. Erityisesti kannattaa panostaa laatuongelmien tutkimiseen. Datassa voi olla virheellisiä tietoja tai tietoja voi puuttua. Data ei ehkä sisälläkään tietoja kaikista tarvittavista tapauksista tai se voi olla tiedoiltaan vanhentunutta.

Tavoitteena on siis tehdä ensivaiheen varmistus, että käytössä on riittävästi laadukasta dataa, jotta data-analyysille asetettu tavoite on mahdollista saavuttaa. Mikäli saatavilla oleva data ei sovellu projektin tarpeisiin, on projekti joko keskeytettävä tai projektin tavoitteita tarkistettava.

Datan ymmärtämisvaiheeseen liittyvän dokumentaation tulee sisältää ainakin listaus hankituista dataseteistä formaatteineen ja tapausten lukumäärineen, millä toimenpiteillä ja mistä ne hankittiin sekä mitä ongelmia datan keräämisessä ja laadunvarmistuksessa havaittiin ja miten ne ratkaistiin.

2.3. Datan valmistelu

Datan valmisteluvaiheessa tehdään kaikki vaadittavat toimet, jotta vaadittava data saadaan muutettua koneoppimismallien hyödynnettävissä olevaan muotoon. Tässä vaiheessa dataa myös tarvittaessa korjataan. Datan ymmärrys- ja valmisteluvaiheet vievät usein merkittävän osan koko projektin aikataulusta.

Aina ei kannata käyttää kaikkea saatavilla olevaa dataa. Datasetit ovat usein muodoltaan taulukoita, joissa sarakkeet edustavat eri muuttujia ja rivit eri tapauksia. Mikäli kaikki muuttujat eivät ole analyysin kannalta merkittäviä tai niiden laadussa on vakavia puutteita, kannattaa ne laskenta- ja muistikapasiteetin säästämiseksi jättää pois analysoitavasta datasetistä.

Tapausten eli rivien valitsemisessa on oltava tarkkana. Jos tapauksia on syytä valikoida niiden liiallisen määrän vuoksi, kannattaa satunnaisotoksen ottamisessa varmistaa, että esimerkiksi luokittelutehtävän ollessa kyseessä kaikkia luokkia edelleen löytyy valitusta datasetistä sopiva määrä. Jos tietyillä riveillä on laatuongelmia esimerkiksi puuttuvien tietojen vuoksi, helppo ratkaisu olisi poistaa tällaiset rivit. Tämä voi kuitenkin johtaa pahasti vinoutuneeseen aineistoon, mikäli tietojen

puuttumiseen liittyy systematiikkaa. Tämän vuoksi puuttuvia tietoja usein mieluummin korvataankin tietyillä oletusarvoilla tai arvioimalla oikea arvo mallintamalla.

Viimeistään ennen koneoppimisalgoritmien ajoa datasetti jaetaan kolmeen osaan. Suurin osa (esimerkiksi noin 80%) tapauksista erotetaan satunnaisotoksella varsinaiseksi koulutusdataksi. Loput tapauksista jaetaan satunnaisesti validointi- ja evaluointidataksi. Validointidatasettiä käytetään mallien parametrien optimoimiseen ja keskinäiseen vertailuun, jotta voidaan valita niistä paras. Evaluointidatasettiä käytetään parhaimmaksi valitun mallin luotettavuuden arviointiin. Jos tähän käytettäisiin validointiaineistoa, voisi luotettavuus näyttää liian hyvältä, koska mallin valinta olisi tehty sillä perusteella, että se toimii kyseisellä datasetillä parhaiten.

Kun käytettävät datasetit on valittu, ne muutetaan koneoppimismallien hyödynnettävissä olevaan muotoon. Yleensä tämä tarkoittaa datasetin numeerista vektorisointia. Mikäli vektorisoinnin muunnossäännöt riippuvat käytettävän datan sisällön arvoista, näiden sääntöjen määrittely tulee tehdä ainoastaan koulutusdataan perustuen. Validointi- ja evaluointidatasetit vektorisoidaan hyödyntäen koulutusdatalla opetettuja sääntöjä. Näin simuloidaan todellista tuotannon tilannetta. Jos esimerkiksi mallin tulisi osata ennustaa tuotannossa sähköpostitse tulleen palvelupyynnön oikea vastaanottaja, sähköpostin sisältö vektorisoidaan koulutusdatalla opittujen sääntöjen mukaisesti.

Samaakin koneoppimismallia voidaan kokeilla datasta eri tavoin muodostetuilla syötteillä. Siksi mallinnusvaiheesta on usein tarpeen palata tähän datan valmisteluvaiheeseen. Palaan tarkemmin tähän aiheeseen luonnollisen kielen käsittelyn osalta luvussa 3.

2.4. Mallinnus

Mallinnusvaiheessa käytetään erilaisia koneoppimismalleja, joiden parametrit optimoidaan. Ensin valitaan siis käytettävät koneoppimismallit. Koneoppimismalliperheitä on useita erilaisia, mutta nykyisin neuroverkkojen käyttö on tullut yhä suositummaksi. Neuroverkkojakin on kuitenkin ääretön määrä, joten kannattaa kokeilla useita erilaisia rakenteita.

Käytännössä mahdollisimman hyvän koneoppimismallin löytäminen on tutkivaa työtä. Lähes jokaisella koneoppimismallilla on omia etukäteen asetettavia, mallin ominaisuuksia säätteleviä hyperparametrejä, joiden arvoja muuttamalla mallin kyvykkyyteen voidaan vaikuttaa. Kilpailevia koneoppimismalleja koulutetaan koulutusdatalla ja niiden kyvykkyyttä arvioidaan käyttämällä validointidataa. Kaikki kokeilujen tuloksena syntyneet mallit parametreineen on hyvä dokumentoida ja listata validointidatalla arvioituun paremmuusjärjestykseen.

2.5. Evaluointi

Kun projektissa on päästy niin pitkälle, että on saatu rakennettua malli, joka näyttäisi olevan validointidatalla arvioiden laadukkain, on tärkeää arvioida malli perusteellisesti ja tarkistaa, että kyseisellä mallilla on mahdollista saavuttaa liiketoiminnan tavoitteet.

Parhaan mallin ennustekyvykkyys arvioidaan käyttämällä evaluointidataa. Kun evaluointi tehdään datalla, jota koneoppimismalli ei ole missään aikaisemmassa vaiheessa ”nähnyt”, saadaan kyvykkyys arvioitua mahdollisimman luotettavasti.

Kun parhaan mallin kyvykkyys on selvillä, arvioidaan, onko se riittävä liiketoiminnan tarpeen kannalta. Monimutkaisessa maailmassamme mikään malli ei ole täydellinen, vaan jokainen malli tekee joitain virheitä. Erilaisten virheiden vaikutusten vakavuudessa voi kuitenkin olla suuria eroja.

Mallin tarkoitus voi olla esimerkiksi päätellä sähköpostitse tulleen palvelupyynnön sisällön perusteella, mihin työjonoon pyyntö pitäisi ohjata. Tällöin kiireellistä kriittistä toimenpidettä vaativien sähköpostien ohjautuminen väärään, mahdollisesti hitaaseen työjonoon voi olla erittäin vakava ongelma, vaikka näin kävisi vain harvoin. Toisaalta ei-kiireellisen palvelupyynnön ohjautuminen väärin ei ole niin vakavaa, kunhan niiden suhteellinen lukumäärä jää niin alhaiseksi, että kokonaishyöty pysyy suurempana kuin kokonaiskustannukset.

Tässä vaiheessa on tärkeää katselmoida koko edeltävä prosessi, jotta voidaan varmistua, että tehdyt toimenpiteet ja koodi, olettamukset, valinnat, suunnitelmat ja laskelmat ovat edelleen perusteltuja ja realistisia eikä niistä löydy merkittäviä puutteita.

Evaluoinnin tulosten perusteella tehdään päätös jatkosta. Parhaassa tapauksessa arvioinnin tulos on, että kehitetty malli täyttää liiketoiminnan tarpeet ja sen kanssa on mahdollista edetä käyttöönottoon. Voi kuitenkin käydä niinkin, ettei mallin kyvykkyys riitä asetettujen tavoitteiden saavuttamiseen. Silloin voi olla järkevää päättää projekti. Yleensä älykkään data-analyysin prosessi on joka tapauksessa tuottanut uutta tietoa ja ymmärrystä aihekentästä. Tämän uuden ymmärryksen valossa on mahdollista tarkistaa asetettuja tavoitteita tai ideoida täysin uusia mahdollisuuksia hyödyntää data-analytiikkaa.

2.6. Käyttöönotto

Kun malli on luotu ja sen laatu on osoitettu riittäväksi, voidaan siirtyä käyttöönottovaiheeseen. Tämä tarkoittaa usein mallin hyödyntämistä osana pidempää prosessia eli mallia pitää voida kutsua muista

järjestelmistä. Käytännössä käyttöönotto voikin vaatia oman järjestelmäprojektin. Siinä tapauksessa projektin kustannukset tulee olla arvioitu jo osana älykkään data-analyysin prosessissa tehtyä kustannus-hyötyanalyysiä.

Projektin päätyttyä kirjoitetaan projektin loppuraportti. Loppuraportissa kannattaa erityisesti pohtia, mitä oppeja olisi annettavissa seuraaville vastaaville projekteille.

2.7. Monitorointi ja ylläpito

Kun koneoppimismalli otetaan osaksi päivittäistä toimintaa, täytyy sen ylläpidosta huolehtia. Maailma ympärillämme ja liiketoimintaprosessit muuttuvat jatkuvasti, joten myös koneoppimis-mallin kyvykkyyttä on monitoroitava jatkuvasti.

Muutos voi olla vähitellen tapahtuvaa, jolloin saattaa riittää, että mallia koulutetaan säännöllisin väliajoin uudelleen. Liiketoimintaan voidaan kuitenkin tehdä myös koneoppimismallin kannalta isoja kertaluonteisia muutoksia. Jos esimerkiksi koneoppimismallin pitäisi ennustaa palvelupyynnön oikea työjono, uusien työjonojen perustaminen tai vanhojen poisto vaatii isomman muutoksen myös malliin. Koska koneoppiminen perustuu datasta oppimiseen ja koska mallin yleensä toivotaan toimivan oikein heti muutosajankohdasta alkaen, on sille tällaisessa tilanteessa todennäköisesti valmisteltava keinotekoista opetusaineistoa.

Mallin jatkuvan ylläpidon ja muutoksiin varautumisen kustannukset ja niiden vaatimat resurssit ovat oleellinen osa kustannus-hyötyanalyysiä. Hallitsemattomat mallin kyvykkyyteen vaikuttavat muutokset voivat aiheuttaa erilaisia riskejä ja niitä olisikin hyvä analysoida osana jatkuvaa riskienhallintaa.

3. Luonnollisen kielen käsittelyn perusteet

Luonnollisella kielellä tarkoitetaan kieltä, jonka alkuperää ei tiedetä (esimerkiksi suomi, ruotsi ja englanti) [6]. Vastaavasti esimerkiksi ohjelmointikielet ovat keinotekoisia kieliä. Luonnollisen kielen käsittelyllä (jäljempänä ”NLP”, Natural Language Processing) tarkoitetaan luonnollisen kielen analysoimiseen ja tuottamiseen tarkoitettuja ohjelmallisia menetelmiä. NLP on laaja tieteenala, jonka menetelmiä käytetään muun muassa oikeinkirjoituksen ja kieliopin tarkastamiseen, konekääntämiseen, puheen tunnistukseen, tekstin muuttamiseen puheeksi, tekstin tiivistämiseen, tiedon hakemiseen hakukoneiden avulla, uutisten aihemallinnukseen, asiakaspalautteiden sentimenttianalyysiin ja tekstidokumenttien luokittelemiseen.

Koneoppimismenetelmät eivät osaa hyödyntää tekstimuotoista dataa sellaisenaan. Jotta luonnollista kieltä voitaisiin analysoida ja käsitellä koneoppimismenetelmin, siitä on ensin johdettava numeerisia piirteitä. Toisaalta moderni NLP on itsessään pääosin soveltavaa koneoppimista. Tämän luvun ymmärtäminen voi siis olla helpompaa, kun on lukenut myös seuraavat luvut 4 (Koneoppimisen perusteet) ja 5 (Neuroverkkojen ja syväoppimisen perusteet).

Tutustutaan ensin tärkeimpiin NLP:n peruskäsitteisiin, jotka valottavat luonnollisen kielen käsittelyn osa-alueita. Olen kerännyt peruskäsitteistön määritelmiä pääosin lähteestä Hakala, Kanerva [6]. Tämän jälkeen esittelen eräitä keskeisiä menetelmiä, miten tekstimuotoinen aineisto voidaan muuttaa numeerisiksi vektorimuotoisiksi piirteiksi. Lopuksi kerron keskeisistä luonnollisen kielen ja erityisesti suomen kielen käsittelyyn kehitetyistä avoimen lähdekoodin työkaluista.

Olen aiemmin tehnyt erikoistyön tekstidokumenttien luokittelusta, jossa olen kooditasolla esitellyt joitain vektorisointitapoja ja luokittelumalleja konkreettisten esimerkkien avulla [7]. Erikoistyö on liitteenä.

3.1. Luonnollisen kielen käsittelyn peruskäsitteitä

Tokenisointi tarkoittaa tekstin jakamista sanoihin. Otetaan esimerkiksi virke:

> ”Kissa nimeltä Maija meni bussilla kauppakeskus Selloon, joka on Espoon suurin :D.”

Tokenisoinnin jälkeen virke näyttäisi tältä:

> [’Kissa’, ’nimeltä’, ’Maija’, ’meni’, ’bussilla’, ’kauppakeskus’, ’Selloon’, ’,’, ’joka’, ’on’, ’Espoon’, ’suurin’, ’:’, ’D’, ’.’]

Jotta Kissa ja kissa tunnistettaisiin jatkossa samaksi sanaksi, muutetaan kaikki isot kirjaimet pieniksi:

> [’kissa’, ’nimeltä’, ’maija’, ’meni’, ’bussilla’, ’kauppakeskus’, ’selloon’, ’,’, ’joka’, ’on’, ’espoon’, ’suurin’, ’:’, ’d’, ’.’]

Huomataan, että samalla myös erisnimien isot kirjaimet katosivat. *Nimettyjen entiteettien* eli esimerkiksi ihmisten, eläinten, organisaatioiden, paikkakuntien ja virhekoodien tunnistaminen (NER, Named Entity Recognition) on yksi NLP:n aktiivisista tutkimus- ja kehityskohteista.

Nimettyjen entiteettien tunnistaminen on tärkeä tiedon uuttamisen alitehtävä. *Tiedon uuttamisen* tavoitteena on poimia ei-rakenteellisista aineistoista oleellisia faktoja ja esittää ne rakenteellisessa muodossa. *Säännöllisten lausekkeiden* avulla on mahdollista määritellä ohjelmallisesti etsittävän merkkijonon muoto ilman, että tarvitsee antaa tarkkaa merkkijonoa. Niiden avulla on helppo löytää tekstistä esimerkiksi päivämäärät tai tietyn muotoiset tuotekoodit.

Sanaston pienentämiseksi on usein tarve poistaa välimerkit sekä hyvin usein toistuvat merkityksettömät sanat eli niin sanotut *pysäytyssanat* (stopwords). Esimerkkivirkkeemme näyttää siis nyt tältä:

> [’kissa’, ’nimeltä’, ’maija’, ’meni’, ’bussilla’, ’kauppakeskus’, ’selloon’, ’espoon’, ’suurin’]

Morfologialla tarkoitetaan kielitieteissä yksittäisten sanamuotojen koostumuksen kuvaamista. Suomen kieli on kaikkine sijamuotoineen morfologisesti erittäin rikas kieli. Esimerkiksi sana ’kissassannekaan’ voidaan jakaa neljään pienempään merkitysyksikköön: kissa, ssa, nne, kaan.

Morfologisessa analyysissä sanalle määritellään sen kaikki mahdolliset *lemmat* eli sanojen perusmuodot, sanaluokat ja morfologiset piirteet. *Sanaluokkaleimaamisessa* (POS tagging, part-of-speech tagging) määritellään sanalle oikea lemma, sanaluokka ja morfologiset piirteet annetussa kontekstissa. Esimerkiksi tekstissä ilmaantuvan sanan ’haavan’ lemma voi olla joko haapa tai haava riippuen kontekstista. Taivutetun sanan muuttamista perusmuotoon kutsutaan *lemmatisoinniksi*.

Lemmatisoinnin jälkeen esimerkkivirkkeemme on saanut muodon:

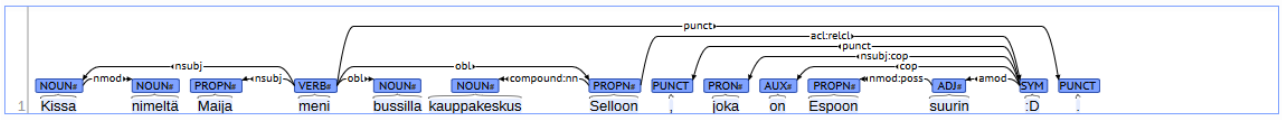
> [’kissa’, ’nimi’, ’maija’, ’mennä’, ’bussi’, ’kauppakeskus’, ’sello’, ’espoo’, ’suuri’]

N-grammeilla tarkoitetaan n:n peräkkäisen sanan tai n:n peräkkäisen merkin yhdistelmiä. Esimerkkilauseemme sanojen 2-grammit ovat:

> [’kissa nimi’, ’nimi maija’, ’maija mennä’, ’mennä bussi’, ’bussi kauppakeskus’, ’kauppakeskus sello’, ’sello espoo’, ’espoo suuri’]

Erityisesti morfologisesti suomen kieltä huomattavasti yksinkertaisempien kielten, esimerkiksi englannin kielen, käsittelyssä saatetaan käyttää lemmatisoinnin sijasta yksinkertaisempaa stemmausta. *Stemmauksessa* sanasta jätetään jäljelle vain sen vartalo. Esimerkiksi sanasta ’meni’ jäisi stemmauksen jälkeen jäljelle vain ’men’.

Kun morfologia tutkii sanojen muodostumista, *syntaksi* eli lauseoppi tutkii lauserakenteita. Siinä sanoja luokitellaan lauseenjäseniksi. Lauseenjäseniä ovat esimerkiksi peruskoulun äidinkielentunneilta tutut subjekti, objekti, predikaatti, attribuutti ja adverbiaali. *Syntaktisessa jäsentämisessä* (syntactic parsing) analysoidaan virkkeelle sen oikea syntaktinen rakenne. Sanojen väliset riippuvuudet kuvataan nuolina sanojen välillä. Näin syntyy *jäsennyspuu*. Työkaluja, jotka analysoivat virkkeen jäsennyspuun automaattisesti, kutsutaan *jäsentimiksi*. Kuvassa 3.1 on esitetty esimerkkilauseemme jäsennyspuu.



Kuva 3.1: Esimerkki jäsennyspuusta. Jäsentimenä on käytetty Turku NLP Groupin Online Parser Demoa [8].

Semantiikka tutkii sanojen ja lauseiden merkitystä, ja *fonologia* puhutun kielen äänteitä.

Strukturoidusti koottuja ja dokumentoituja aitoja tekstiaineistoja kutsutaan *korpuksiksi*. Laadukkaat korpukset ovat välttämätön edellytys luonnollisen kielen automaattisen käsittelyn eri osa-alueiden kehittämiseksi. Osa korpuksista on käsin *annotoituja*, eli niihin on manuaalisesti merkitty tarvittavat kieliopilliset tunnukset.

3.2. Bag-of-words-menetelmä

Yksinkertaisin tapa muodostaa teksteistä piirrevektoreita on niin sanottu bag-of-words-menetelmä. Siinä jokaiselle opetusdatasetissä esiintyvälle eri sanalle annetaan ensin oma indeksi. Tämän jälkeen jokaiselle dokumentille muodostetaan vektori laskemalla kunkin sanan esiintymismäärä kyseisessä dokumentissa ja tallentamalla tämä luku vektoriin aina kyseistä sanaa vastaavan indeksin kohdalle. Bag-of-words -menetelmä on siis kiinnostunut vain sanojen lukumääristä, se ei ota huomioon sanajärjestystä.

Otetaan esimerkiksi kaksi tekstiä:

Teksti 1: ”Kissa nimeltä Maija meni bussilla kauppakeskus Selloon, joka on Espoon suurin.”

Teksti 2: ”Matti soittaa selloa Espoon musiikkiopistossa. Matti soittaa hyvin.”

Tokenisoinnin, lemmatisoinnin ja muun datan esivalmistelun jälkeen, tekstit näyttävät tältä:

Teksti 1: [’kissa’, ’nimi’, ’maija’, ’mennä’, ’bussi’, ’kauppakeskus’, ’sello’, ’espoo’, ’suuri’]

Teksti 2: [’matti’, ’soittaa’, ’sello’, ’espoo’, ’musiikkiopisto’, ’matti’, ’soittaa’, ’hyvä’]

Kun kaikki erilliset sanat tallennetaan järjestykseen, saadaan 13 sanan lista:

[’kissa’, ’nimi’, ’maija’, ’mennä’, ’bussi’, ’kauppakeskus’, ’sello’, ’espoo’, ’suuri’, ’matti’, ’soittaa’, ’musiikkiopisto’, ’hyvä’]

Tekstien 13 merkkiä pitkät bag-of-word vektorit näyttäisivät siis tältä:

Teksti 1: [1 1 1 1 1 1 1 1 1 0 0 0 0]

Teksti 2: [0 0 0 0 0 0 1 1 0 2 2 1 1]

Molempien vektoreiden seitsemäs luku vastaa sanaa ’sello’ riippumatta siitä, merkitseekö sana kauppakeskusta vai soitinta. Tässä yhteydessä on hyvä muistaa, että erillisten sanojen listaus ja indeksointi tehdään ainoastaan koulutusdatan perusteella. Validointi- ja evaluointiaineistojen vektorisointi tehdään perustuen tähän koulutusdatalla muodostettuun indeksointiin.

Jos esimerkiksi validointidatasetin sisältö olisi:

Validointiteksti: ”Kirsin sello on suuri ja kaunis”

sen esivalmisteltu sisältö olisi:

Validointiteksti: [’kirsi’, ’sello’, suuri’, ’kaunis’]

ja bag-of-words-vektori:

Validointiteksti: [0 0 0 0 0 0 1 0 1 0 0 0 0]

Sanat ’kirsi’ ja ’kaunis’ puuttuvat indeksilistalta, joten ne jäisivät vektorisoinnissa kokonaan huomioimatta. Kattavan koulutusaineiston rooli on siis tässäkin olennainen.

Jos koulutusaineistossa olisi tekstejä suuri määrä ja ne käsittelisivät eri aiheita, erillisten sanojen lista pitenisi nopeasti. Jos esimerkiksi erillisiä sanoja olisi yhteensä 400 000, jokaisen tekstin bag-of-word-vektori olisi 400 000 luvun mittaisia. Tekstiä 1 vastaavassa bag-of-words-vektorissa olisi edelleen yhdeksän ensimmäistä numeroa ykkösiä, mutta sen perässä tulisi 399 991 nollaa. Muistin säästämiseksi käytetäänkin yleensä harvoja matriiseja, jotka tallentavat vain nollasta poikkeavat tiedot.

Vektorisoinnissa voidaan ottaa mukaan myös esimerkiksi kahden ja kolmen peräkkäisen sanan yhdistelmät eli 2- ja 3-grammit. Tällöin vektorit luonnollisestikin pitenevät entisestään.

3.3. Termifrekvenssi, dokumenttifrekvenssi ja näiden yhdistelmä

Termifrekvenssi (tf) ilmoittaa tietyn sanan eli termin lukumäärän dokumentissa. Ajatus on, että mitä useammin sana esiintyy dokumentissa, sitä paremmin se kuvaa kyseisen dokumentin sisältöä. Sanojen lukumääristä muodostetut piirrevektorit eivät kuitenkaan välttämättä ole sellaisenaan riittävän kuvaavia. Yleensä dokumentit ovat eri pituisia, jolloin sanojen keskimääräiset lukumäärät voivat vaihdella paljonkin, vaikka dokumentit edustaisivat samaa aihepiiriä. Siksi kunkin sanan lukumäärä dokumentissa kannattaa jakaa dokumentin kaikkien sanojen lukumäärällä.

Dokumenttifrekvenssi (df) puolestaan ilmaisee niiden dokumenttien lukumäärän, joissa kyseinen termi esiintyy. Eli jos sanan dokumenttifrekvenssi on pieni, sana esiintyy vain harvoissa dokumenteissa. Tällainen semanttisesti keskittynyt termi voi kuitenkin esiintyä yhdessä dokumentissa useita kertoja. Jos taas dokumenttifrekvenssi on suuri, kyseinen sana esiintyy monissa dokumenteissa. Tällaisen sanan käyttö piirrevektorissa ei juurikaan tuo luokittelulle hyödyllistä lisäinformaatiota.

Kun termifrekvenssi jaetaan dokumenttifrekvenssillä (kerrotaan käänteisellä dokumenttifrekvenssillä), saadaan laskettua paljon käytetty vektorisointimalli tf_idf.

3.4. Sanaupotusmallit

Edellä esitetyissä bag-of-words-tyyppisissä vektorisoinneissa kaikkia sanoja pidetään toisistaan täysin riippumattomina. Avaruudellisesti ajatellen ne muodostavat vektoriavaruuden, jossa kaikki sanat ovat ortogonaalisessa suhteessa toisiinsa ja siis yhtä kaukana toisistaan. Sanojen jakaumiin pohjaavan merkitysopin (distributional semantics) perusideana on, että sanan merkitys voidaan määrittää sen perusteella, minkä muiden sanojen yhteydessä se esiintyy. Näin semanttisesti toisiaan lähellä olevat sanat sijoittuvat myös vektoriavaruudessa lähelle toisiaan. Esimerkiksi mustikka ja mansikka ovat sanaupotusavaruudessa lähellä toisiaan, samoin kissa ja koira.

Sanaupotukset ovat sanan merkitystä kuvaavia reaalilukuarvoisia vektoreita. Bag-of-words-mallilla muodostetut sanavektorit ovat yleensä hyvin pitkiä (koko sanaston mittaisia, esimerkiksi kaksi miljoonaa) ja harvoja eli lähes kaikki arvot ovat nollia. Sanaupotusvektoreista sen sijaan tulee huomattavasti lyhyempiä, esimerkiksi 50-1000 luvun mittaisia.

Sanojen keskinäinen läheisyys voidaan laskea käyttäen kosini samankaltaisuutta. Suure saa suurimman arvonsa 1, kun vektoreiden välinen kulma on 0 astetta, ja pienimmän arvonsa 0, kun vektorit ovat ortogonaalisia suhteessa toisiinsa. Esimerkiksi fastTextin valmiiksi koulutetussa suomenkielisessä mallissa sanojen 'jalkapallo' ja 'jääkiekko' kosini samankaltaisuuden arvo on sanaupotusvektoreiden perusteella 0.71136767. Vastaavasti sanojen 'jalkapallo' ja 'jää' kosini samankaltaisuuden arvo on vain 0.1774352. [7]

Sanaupotuksia on mahdollista opettaa valmiilla neuroverkkomalleilla omasta aineistosta. Tällaisia työkaluja ovat ainakin Googlen tutkijoiden kehittämät word2vec [9] ja kontekstiriippuvien vektoreiden tuottamiseen pystyvä BERT [10], Facebookin tutkijoiden kehittämä fastText [11, 12, 13] ja Stanford Universityn tutkijoiden kehittämä GloVe [14]. Tämä vaatii kuitenkin suuren datamäärän sekä runsaasti laskentakapasiteettia. Siksi usein käytetäänkin valmiiksi opetettuja (jollakin kattavalla aineistolla) sanaupotuksia.

Sanaupotusvektoreiden kouluttamisen katsotaan olevan ohjaamatonta koneoppimista (unsupervised learing), sillä se ei vaadi koulutusdatana käytetyn syötetekstin annotointia. Käytännössä käytetyt algoritmit voivat kuitenkin hyödyntää ohjattua koneoppimista (supervised learning). Termit ohjaamaton ja ohjattu koneoppiminen on selitetty tarkemmin luvun neljä alussa.

Esimerkiksi Skip-gram on toinen word2vec-työkalun käyttämistä algoritmeista. Skip-gramin idea on, että sen sijaan että laskisimme, kuinka usein mikäkin sana s esiintyy esimerkiksi sanan ’koira’ lähellä, luokittelija opetetaan ennustamaan, kuinka todennäköisesti sana s esiintyy sanan ’koira’ yhteydessä. Sanaupotusvektorit muodostetaan näin opetettujen luokittelijoiden painoarvoista. Algoritmin hienous on, että se voi hyödyntää ohjattua koneoppimista, koska oikeat vastaukset löytyvät samasta aineistosta ilman, että koulutusdataa tarvitsisi erikseen annotoida. [15]

Tarkastellaan seuraavassa luvussa hieman sanaupotuksiin liittyvää eettistä haastetta. Tämän jälkeen kerron, miten valmiiksi koulutettuja sanaupotusvektoreita on mahdollista hyödyntää. Tutustutaan myös lähemmin kaikista uusimpaan ja siten edistyneimpään kielimalliin BERTiin (Bidirectional Encoder Representations from Transformers), joka osaa ottaa huomioon myös sanojen kontekstin.

3.4.1. Sanaupotuksiin liittyvä eettinen haaste

Tekoälyn eettisistä haasteista puhutaan paljon. Yksi eettisten haasteiden kokonaisuus liittyy siihen, että koneoppimismenetelmät oppivat toistamaan datan vääristymiä. Näiden vääristymien tunnistaminen ja välttäminen voi olla todella vaikeaa. Sanaupotukset ovat tästä hyvä esimerkki. Myös niiden käyttöön liittyy eettisiä näkökulmia, joita ei ehkä heti tule ajatelleeksi.

Sanaupotukset mallintavat sanojen suhteellisia merkityksiä. Esimerkiksi sanat mies ja nainen ovat lähellä toisiaan, koska molemmat sanat kuvaavat ihmistä. Toisaalta niitä voidaan pitää myös vastakkaisina, koska ne korostavat asiaa, joka erottaa ihmiset toisistaan. Kun tällaisia sanojen suhteellisia eroja tutkitaan, ovat vektorierot naisen ja miehen, kuningattaren ja kuninkaan sekä siskon ja veljen välillä suunnilleen samanlaisia. Näin ollen esimerkiksi yhtälön vektori(‘kunigas’) – vektori(‘mies’) + vektori(’nainen) tuloksena saatava vektori on lähellä vektori(’kuningatar’):ta. Vastaavasti vektori(’Pariisi’) - vektori(’Ranska’) + vektori(’Italia’) on lähellä vektori(’Rooma’):a. [14]

Valitettavasti sanaupotukset kuitenkin mallintavat samalla myös tekstissä piilossa olevia hyvinkin loukkaavia stereotypioita ja rasismia. Kun tutkijaryhmä analysoi suosittua, vapaasti saatavilla olevaa, Googlen uutisista koostuvasta englanninkielisestä korpuksesta word2vec-ohjelmistolla valmiiksi koulutettua sanaupotusmallia, se havaitsi, että sanaupotusmallissa esimerkiksi yhtälön vektori(‘mies’) – vektori(’ohjelmoija’) + vektori(’nainen’) tuloksena saatava vektori on noin vektori(’kotiäiti’). Samoin yhtälön vektori(‘isä’) – vektori(‘lääkäri’) + vektori(’äiti’) tulos on noin vektori(’hoitaja’). [16]

Koska valmiiksi koulutettuja sanaupotusmalleja käytetään hyvinkin laajasti perusominaisuuksina eri sovelluksissa, stereotypioita heijastavat sanaupotukset voivat myös vahvistaa niitä. Esimerkiksi hakukonealgoritmi, jonka tehtävänä on etsiä potentiaalisia ohjelmoijia tai lääkäreitä, ja joka käyttää stereotypioita sisältävää sanaupotusmallia, saattaa virheellisesti priorisoida nettisivuja, joissa esiintyy miesten nimiä ja vastaavasti aliarvostaa sivuja, joissa esiintyy naisten nimiä. [15,16]

Kammottava esimerkki sanaupotuksiin sisältyvästä rasismista on, että afroamerikkalaisten nimien (esimerkiksi Leroy ja Shaniqua) sanaupotusvektoreiden kosini samankaltaisuus oli tutkittaessa suurempi epämiellyttävien sanojen kanssa, kun taas eurooppalaisamerikkalaisten nimien (esimerkiksi Brad, Greg tai Courtney) kosini samankaltaisuus oli suurempi miellyttävien sanojen kanssa. [15]

3.4.2. Valmiiksi koulutettujen kontekstiriippumattomien sanaupotusvektoreiden hyödyntäminen

Kontekstiriippumattomia sanaupotusmalleja ovat esimerkiksi word2vec, fastText ja GloVe. Kaikkien näiden hyödyntäminen tapahtuu samalla tavalla. Morfologisesti erittäin rikkaassa suomen kielessämme sanoilla on useita taivutusmuotoja ja sanajohdoksia. Käytän tässä esimerkkinä fastText:iä, joka sopii hyvin suomenkielisten sanojen vektoriesitysten muodostukseen, koska se hyödyntää mallissaan myös sanojen alimerkkijonoja.

Ensin ladataan valmiiksi koulutetut sanaupotusmallit. Esimerkiksi fastTextin sivustolta [11] voi ladata fastTextin valmiiksi koulutetut sanaupotusvektorit 157 kielelle. Sieltä voi siis ladata myös

suomenkielisten sanojen vektoriesitykset tekstimuodossa sanaupotusmallin pohjaksi. Tässä valmiiksi koulutetussa mallissa on yhteensä kaksi miljoonaa sanaa. Sanat esitetään 300-dimensioisen avaruuden vektoreina.

Kun esimerkiksi sähköpostiviesteistä koostuva tekstiaineisto halutaan vektorisoida käyttäen hyväksi valmiiksi koulutettuja sanaupotusvektoreita, aloitetaan prosessi koulutusdatan tokenisoinnilla ja indeksoinnilla. Koska fastText on hyödyntänyt sanaupotuksia laskiessaan myös sanojen alimerkkijonoja, ei lemmatisointia eli sanojen perusmuotoistamista tarvita, varsinkaan mikäli koulutusdataa on runsaasti.

Kun sanasto on indeksoitu koulutusdatan perusteella, muodostetaan koulutus-, validointi- ja evaluointiaineistojen jokaisesta dokumentista lista kokonaislukuja niin, että jokainen dokumentin sana korvataan kyseistä sanaa vastaavalla indeksillä.

Seuraavaksi muodostetaan matriisi, joka yhdistää dokumenteista muodostetun sanaston sanat sanaupotusvektoreihin. Eli muodostettavan matriisin riville 1 tulee viesteistä muodostetun sanaston indeksiä 1 vastaavan sanan sanaupotusvektori (esimerkiksi 300-dimensioinen fastText-sanaupotus-vektori). Jos kyseiselle sanalle ei löydy valmista upotusvektoria, sitä vastaavan vektorin kaikki luvut jäävät matriisissa arvoon 0. Käytännössä malli ottaa siis huomioon vain sellaiset sanat, jotka sisältyvät koulutusaineistoon ja joille löytyy valmiiksi koulutettu sanaupotusvektori.

Tämän prosessin tuloksena muodostettua sanaupotusmatriisia pystytään nyt hyödyntämään neuroverkon sanaupotuskerroksen painokertoimien arvoina. Neuroverkkomallia muodostettaessa voidaan päättää, pidetäänkö sanaupotusmatriisin painokertoimia lopullisina vai halutaanko niitä edelleen kouluttaa analysoitavan aineiston perusteella.

3.4.3. BERT-kielimalli

Word2vecin, fastTextin ja GloVen heikkoutena on, että ne antavat samalle sanalle aina saman vektoriesityksen riippumatta siitä, missä kontekstissa sanaa on käytetty. Esimerkiksi sanalle ’kuusi’ on aina sama vektoriesitys, vaikka se voi tarkoittaa joko numeroa tai puulajia. BERT-kielimalli ottaa huomioon myös sanojen kontekstin, joten sen avulla voidaan opettaa myös sanojen kontekstisidonnaisia upotusvektoreita. Kielimalleja on muitakin, mutta esittelen nimenomaan BERT-kielimallin, koska sitä on tarkoitus hyödyntää Valtorin tekoälyratkaisussa.

Perinteiset kielimallit on kehitetty ennustamaan seuraava sana, kun tiedetään edeltävät sanat. Niitä on kehitetty perustuen erilaisiin todennäköisyysmalleihin, mutta neuroverkkoja hyödyntävät mallit ovat osoittautuneet taitavimmiksi. Kielimalleja käytetään muun muassa puheen tunnistukseen, konekääntämiseen, tekstin tiivistämiseen ja muihin tekstin ymmärtämiseen liittyviin tehtäviin.

BERT on lyhenne sanoista Bidirectional Encoder Representations from Transformers. Kaksisuuntaisuus (bidirectional) viittaa siihen, että BERTiä edeltävät kontekstin huomioivat kielimallit ovat pystyneet analysoimaan tekstiä ainoastaan yksisuuntaisesti. Otetaan esimerkiksi lause ’Sain haavan lehden’. Yksisuuntainen malli vasemmalta oikealle näkisi vain sanat ’Sain _’, kun sen pitäisi ennustaa sanan ’haavan’ vektoriesitys. Malli ennustaisi todennäköisemmin sanan ’haava’ kuin ’haapa’ taivutusmuodon. BERTin kaksisuuntainen malli näkee molemmat suunnat. Eli esimerkissämme puuttuvan sanan ’haavan’ vektoriesitys muodostettaisiin perustuen sekä oikeaan että vasempaan kontekstiin: ’Sain _ lehden’. Jotta BERT voisi ottaa huomioon sekä sanan vasemman- että oikeanpuoleisen kontekstin, se käyttää niin sanottua maskattua kielimallia (masked language model). Käytännössä 15 % mallin sisään syötettävistä sanoista maskataan ja ainoastaan nämä maskatut sanat ennustetaan. [10]

Maskattujen sanojen ennustamisen lisäksi BERT oppii korpuksesta virkkeiden välisiä suhteita. Opittava ennustamistehtävä on: kun otetaan kaksi virkettä A ja B, onko B seuraava virke, joka tulee A:n jälkeen, vai vain satunnainen virke korpuksesta? [17]

BERT hyödyntää transformeri-neuroverkkomalliarkkitehtuuria (transformer), joka perustuu niin sanottuun huomiomekanismiin (attention), joka mallintaa syötteiden ja tuotosten välisiä riippuvuuksia. Transformeri sisältää kaksi erillistä mekanismia, enkooderin (encoder), joka lukee tekstinsyötön, ja dekooderin, joka tuottaa tehtävän ennusteen. Koska BERTin tavoitteena on luoda kielimalli, se hyödyntää ainoastaan enkooderi-mekanismia. Transformer-arkkitehtuuri on kuvattu tarkemmin Vaswani et al.:in artikkelissa [18]. Attention-mekanismiin voi perehtyä myös animaatiota hyödyntävän Alammarin ei-akateemisen nettiartikkelin avulla [19]. Sama henkilö on havainnollistanut kuvilla myös koko BERT-kielimallia ja sen perusteita [20].

BERT-kielimallin koulutus vaatii valtavan korpuksen ja huikean määrän laskentaa. Käytännössä hyödynnetäänkin esikoulutettuja BERT-malleja. Esikoulutetun BERT-mallin päälle neuroverkkoon lisätään sovelluskohteen tehtävään sopiva ennustekerros. Tämän täydennetyn neuroverkkomallin kouluttamista kutsutaan hienosäädöksi (fine tuning).

Googlen esikouluttamista BERT-kielimalleista ei löydy puhtaasti suomen kielellä koulutettua mallia, vaan suomen kieltä käytettäessä Google nojautuu monikieliseen kielimalliin (multilingual BERT, M-BERT). M-BERT on koulutettu 104:ää eri kieltä sisältävällä yhteisaineistolla. Googlella ei ole suunnitelmissa julkaista enempää yksikielisiä malleja. Sen sijaan se saattaa jossain vaiheessa julkaista myös laajemman esikoulutetun version M-BERTistä. [21]

Googlen tutkijat ovat osoittaneet, että M-BERT toimii yllättävän hyvin, mutta siinä on järjestelmällisiä puutteita tiettyjen kieliparien välillä. Kielimallin siirto toimii parhaiten typologisesti

samankaltaisten kielten välillä, mutta on mahdollista myös erilaisten kielten välillä. Tutkimuksessa ei raportoitu suomen kieltä koskevia tuloksia. [22]

Turun yliopiston tutkijat ovat itse kouluttaneet suomenkielisen kielimallin FinBERTin, joka on avoimella lisenssillä julkisesti saatavilla. He myös vertasivat M-BERTillä ja FinBERTillä saavutettavaa laatutasoa erilaisissa luonnollisen kielen käsittelyn tehtävissä (sanaluokkaleimaaminen, nimettyjen entiteettien tunnistaminen, jäsentäminen ja luokittelu). Tulosten mukaan FinBERT suoriutui kaikista tehtävistä paremmin kuin M-BERT. Se suoriutui myös paremmin kuin aiemmat parhaat mallit. [23, 24]

3.5. Suomen kielen käsittelyyn kehitettyjä työkaluja ja korpuksia

Luonnollisen kielen käsittelyyn on kehitetty lukemattomia avoimella lisenssillä käytettävissä olevia koodikirjastoja ja muita työkaluja. Suurinta osaa voidaan käyttää ainakin osittain myös suomen kielen käsittelyyn. Niiden käytössä tulee kuitenkin huomioida suomen kielen erilaisuus esimerkiksi suhteessa morfologisesti huomattavasti yksinkertaisempaan englannin kieleen. Kaikki mallit, jotka toimivat hienosti englanninkielisen aineiston kanssa, eivät välttämättä olekaan hyviä tapoja käsitellä suomen kieltä.

Tunnetuin kotimaisten korpusten kokoelma on FIN-CLARIN-konsortion ylläpitämä Kielipankki-palvelu. Kielipankin sivustolta löytyy kattava lista saatavilla olevista aineistoista ja niihin liittyvistä lisensseistä. [25]

Valtion kehitysyhtiö Vake Oy, Helsingin yliopisto ja Yle ovat käynnistäneet Lahjoita puhetta -kampanjan, jossa tavoitteena on kerätä 10 000 tuntia vapaasti puhuttua suomen kieltä. Myös tämä puhekorpus tullaan tallentamaan Kielipankkiin tutkijoiden ja sovelluskehittäjien hyödynnettäväksi. [26, 27]

Kielipankin sivustolle on listattu myös suomen kielen käsittelyyn kehitettyjä työkaluja. Listasta pääsee linkillä siirtymään kunkin työkalun lähdesivustolle.

Esikoulutettujen mallien on hyvä olla koulutettu suomenkielisellä aineistolla. Edellä tutustuimmekin jo suomenkieliseen FinBERT-kielimalliin, jonka avulla on saavutettu parempia tuloksia kuin Googlen esikouluttamalla monikielisellä M-BERTillä. Samoin tutustuimme edellä Facebookin kouluttamiin suomenkielisiin fastText-sanaupotusvektoreihin.

FinBERTin pohjalta on kehitetty myös nimettyjen entiteettien tunnistamisen systeemi Finnish NER [28]. Näiden lisäksi Turun yliopiston Turku NLP -ryhmä on kehittänyt jäsennystyökalun Turku

Neural Parser Pipeline, joka on suomen kielen lisäksi koulutettu yli 50 muulla kielellä [29] sekä selaimella käytettävän työkalun, jolla voidaan tutkia sanojen semanttista samankaltaisuutta ja analogioita (word2vec) [30]. Jäsennystyökalullekin on tehty oma demo, jota edellä jo esimerkin yhteydessä hyödynnettiinkin [8].

Suomen kansalliskirjasto on kehittänyt Annif-työkalun aiheiden automaattista indeksointia ja luokittelua varten. Annif:ia on mahdollista opettaa myös omalla aineistolla. Annif hyödyntää olemassa olevien NLP:n ja koneoppimisen mallien yhdistelmää (Maui, Omikuji, fastText ja Gensim). [31]

FISKMÖ (finsk-svensk korpus & maskinöversättning) on Helsingin ja Turun yliopistojen sekä Kitesin (Suomen kielisektorin kattojärjestö) yhteinen projekti. Projektin rahoittaja on Svenska kulturfonden. Projektin tavoitteena on koota erittäin laaja suomen- ja ruotsinkielisiä tekstejä sisältävä rinnakkaiskorpus sekä näiden pohjalta julkinen konekäännöspalvelu suomen ja ruotsin kielten välisiä käännöksiä varten. Käännöspalvelua on mahdollista käyttää selaimella julkisesta verkosta tai osana käännösmuistiohjelmaa. [32]

Erityisesti suomen kielen käsittelyyn on siis kehitetty useita hyödyllisiä työkaluja ja korpuksia, joita kannattaa projekteissa hyödyntää ja joiden kehittäjien kanssa on yleensä mahdollista verkostoitua. Varsinkin yhteisten kotimaisten kielten korpusten kokoamiseen olisi kaikkien julkisten toimijoiden tärkeää osallistua lahjoittamalla aineistojaan mahdollisuuksien mukaisesti.

4. Koneoppimisen perusteet

Koneoppimisella tarkoitetaan järjestelmiä, jotka parantavat suorituskykyään tietyssä tehtävässä sitä mukaa, kun lisää kokemusta tai dataa kertyy [3] Koneoppiminen voidaan jakaa karkeasti kolmeen eri kategoriaan: ohjattuun oppimiseen (supervised learning), ohjaamattomaan oppimiseen (unsupervised learning) sekä vahvistusoppimiseen (reinforcement learning). Näistä ohjatun oppimisen mallit ovat käytetyimpiä [33].

Ohjatun oppimisen malleja on mahdollista käyttää, mikäli koulutusdatasta löytyy ennalta kullekin näytteelle oikea tulos (esimerkiksi luokka tai arvo). Malli oppii koulutusdatan perusteella ennustamaan tuloksen uusille näytteille, joiden oikeaa tulosta ei ole tiedossa. Luokittelumallit ennustavat oikean luokan (esimerkiksi sairausdiagnoosin) ja regressiomallit oikean reaaliluvun (esimerkiksi asunnon arvon).

Ohjaamattoman oppimisen malleja käytetään datan tutkimiseen, kun ei ole olemassa valmiita oikeita vastauksia. Klusteroinnissa, kuten esimerkiksi asiakassegmentoinnissa, datasta muodostetaan samankaltaisten tapausten ryhmiä. Assosiaatioanalyysissä tutkitaan todennäköisyyksiä, että tietyt kokoelman kohteet esiintyvät samanaikaisesti. Tätä käytetään muun muassa ostoskorianalyyseissä ja nettikauppojen suosittelusysteemeissä. Poikkeama-analyysiä voidaan käyttää, kun halutaan etsiä poikkeuksellisesti esiintyviä tai käyttäytyviä osajoukkoja. Ohjaamattoman oppimisen työkaluja voidaan käyttää myös datan visualisointiin ja esikäsittelyyn. Tästä esimerkkinä on moniulotteisen datan projisointi tasoon. Kuten edellä opimme, myös sanaupotusvektoreiden kouluttamista tekstikorpuksesta pidetään ohjaamattomana oppimisena, koska se ei vaadi annotoitua syöteaineistoa.

Vahvistusoppimisessa positiivinen tai negatiivinen palaute annetaan vasta usean vaiheen jälkeen. Esimerkiksi pelien opettamiseen tekoälysovellukselle käytetään vahvistusoppimista. Pelin voitto tai häviö ratkeaa vasta usean siirron jälkeen. Tekoälysovellukselle ei siis välittömästi osoiteta oikeaa vastausta, vaan sen täytyy kokeilla useita erilaisia siirtoketjua, kunnes se oppii valitsemaan siirrot oikein eli voittamaan.

Kerron seuraavassa aliluvussa niin sanotuista perinteisistä koneoppimismalleista. Tämän jälkeen selitän tarkemmin, mitä koneoppimismallien oppiminen tarkoittaa, miksi mallikoosteiden käyttö yleensä parantaa ennustekyvykkyyttä ja miten koneoppimismallien ennustekyvykkyyttä voidaan mitata.

4.1. Perinteiset koneoppimismallit

Yksinkertaisten koneoppimismallien etu on, että niiden laskentakapasiteettitarve on yleensä minimaalinen verrattuna varsinkin monimutkaisiin neuroverkkoihin. Niiden avulla onkin usein nopeaa ja helppoa laskea ensimmäinen arvio, kuinka hyvin koneoppimisella on mahdollista ratkaista käsillä oleva oppimishaaste. Neuroverkkoja käyttäen päästään yleensä ainakin jonkin verran parempiin tuloksiin, mutta niiden opettaminen on laskennallisesti huomattavasti raskaampaa. Yksinkertaisemmilla malleilla saadaan siis tehokkaasti selvitettyä koneoppimismallien minimikyvykkyystaso, joka neuroverkkomallilla pitää saada ylitettyä.

Perinteisten koneoppimismallien käyttämisen haasteena on kuvaavien piirteiden muodostaminen (feature engineering). Esimerkiksi värikuviin liittyvä data koostuu pikselikohtaisista vihreän, punaisen ja sinisen värin määrää kuvaavista luvuista. Näiden lukujen käyttämisestä sellaisenaan perinteisen koneoppimismallin syötteenä ei olisi mitään hyötyä. Tämän vuoksi on kehitetty erilaisia metodeja, miten raakadatasta saadaan muodostettua sellaisia piirteitä, että koneoppimismallit voisivat niiden avulla selvitä niille asetetuista tehtävistä mahdollisimman luotettavasti. Tietyillä soveltamisalueilla hyödyllisten piirteiden muodostaminen saattaakin muodostua pullonkaulaksi perinteisten koneoppimismallien hyödyntämiselle [34].

Erilaisia koneoppimismalleja on lukuisia. Koska neuroverkot ovat vain yksi koneoppimisen alalaji, kerron esimerkinomaisesti lyhyesti myös muutamasta muusta yleisesti käytetystä koneoppimismallista. Niiden tiedot ovat pääosin lähteestä Heikkonen [34].

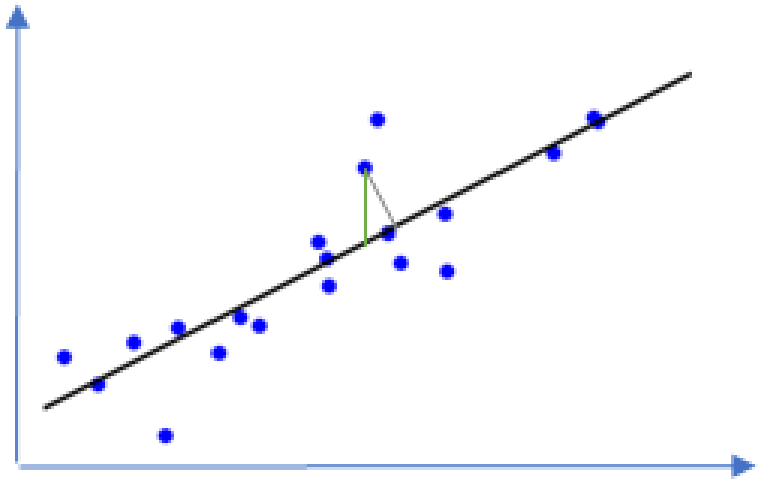
4.1.1. Esimerkkejä ohjatun oppimisen malleista

Aloitetaan yksinkertaisesta todennäköisyyksiin perustuvasta Naive Bayes -luokittelumenetelmästä. Bayesilaisen päätösteorian mukainen luokittelija suosittelee päätöksiä, jotka minimoivat odotetun kokonaisriskin. Riskinä voi olla esimerkiksi virheen tekeminen luokittelussa. Naive Bayes -menetelmässä tehdään laskentaa merkittävästi yksinkertaistava oletus, että datasta johdetut piirteet olisivat keskenään riippumattomia. Vaikka tämä oletus yksinkertaistaa yleensä mallia liikaakin, esimerkiksi tekstin luokittelussa Naive Bayesilla saadaan usein yllättävän hyviä tuloksia edistyneempiin menetelmiin nähden erittäin vähäisellä laskentakapasiteetilla.

Toinen erittäin yksikertainen koneoppimismalli on k lähintä naapuria (k nearest neighbors). Siinä luokittelu tehdään k:n lähimmän opetusnäytteen arvojen perusteella. Vaikka metodi on yksinkertainen, se pystyy oppimaan monimutkaisia epälineaarisia funktioita. Jos kyseessä on luokittelutehtävä, malli ennustaa uudelle näytteelle sen luokan, joka on yleisin sen k:n lähimmän

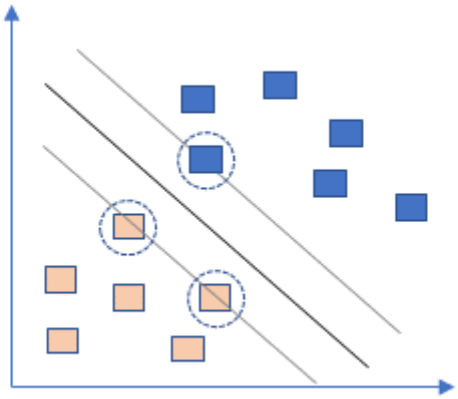
naapurin keskuudessa. Jos taas kyseessä on regressiotehtävä, malli laskee k:n lähimmän naapurin keskiarvon. Yli- ja alisovittumista voidaan säädellä muuttamalla hyperparametrin k arvoa. Palaan yli- ja alisovittumisen haasteeseen tarkemmin luvussa 4.3.

Yksinkertaisin regressiomalli on lineaarinen regressio, jossa datapisteisiin sovitetaan suora esimerkiksi minimoimalla kunkin datapisteen etäisyyksien neliöiden summa (least squares error) suhteessa tähän suoraan. Kuvassa 4.1 on esimerkki lineaarisesta regressiosta. Siinä on esitelty myös kaksi tapaa määritellä pisteen etäisyys suorasta. Tarvittaessa lineaarista regressiota on mahdollista käyttää myös epälineaaristen mallien luontiin lisäämällä koulutusdataan piirteitä eli sarakkeita, joissa muuttujia on kuvattu yhtälöiden avulla korkeampaan dimensioon. [35]



Kuva 4.1: Esimerkki lineaarisesta regressiosta.

Tukivektorikone (support vector machine) [36, 37] on luokittelija, joka pyrkii erottamaan kaksi luokkaa toisistaan sovittamalla niiden väliin päätöshypertason ja tämän tason kanssa yhdensuuntaiset toisistaan mahdollisimman etäälle mutta yhtä kauas päätöstasosta sijoittuvat marginaalihypertasot. Parhaassa tapauksessa marginaalitasot pystytään muodostamaan niin, ettei niiden väliin jää yhtään datapistettä. Silloin marginaalitasojen paikan määräävät ne datapisteet, jotka ovat lähimpänä päätöstasoa. Näitä datapisteitä kutsutaan tukivektoreiksi. Kuvassa 4.2 on esimerkki tällaisista lineaarisista marginaalitasoista. Kuvassa tukivektorit on merkitty ympyröimällä. Usein eri luokkia ei pystytä erottamaan toisistaan tasolla täydellisesti toisistaan. Silloin käytetään ns. joustavan marginaalin luokitinta, mutta silloinkin tukivektoreiksi kutsutaan niitä datapisteitä, jotka määrittävät marginaalitasojen paikan. Mikäli luokat eivät ole erotettavissa toisistaan lineaarisilla tasoilla, voidaan tässäkin menetelmässä piirteitä lisätä kuvaamalla muuttujia yhtälöiden avulla korkeampaan dimensioon.



Kuva 4.2: Esimerkki lineaarisista marginaalitasoista. Tukivektorit on merkitty ympyröimällä.

4.1.2. Esimerkkejä ohjaamattoman oppimisen malleista

Helppo tapa tehdä klusterointia, on käyttää k:n keskiarvon klusterointimallia (k-means clustering). Ensin valitaan, kuinka monta (k) klusteria halutaan muodostaa ja alustetaan niin monen klusterikeskuksen paikka satunnaisesti. Tämän jälkeen määritellään kunkin datapisteen klusteri sen perusteella, mikä klusterikeskus on sitä lähinnä. Uudet klusterikeskusten paikat lasketaan kuhunkin klusteriin kuuluvien datapisteiden keskiarvona. Näin jatketaan niin kauan, etteivät klustereiden keskukset enää liiku. K:n keskiarvon klusterointi on esimerkki vektorikvantisointimenetelmistä, joita voidaan käyttää muun muassa datan kompressointiin. Sen avulla on myös mahdollista muuttaa regressiotehtävä luokittelutehtäväksi.

Suomalaisen professorin Teuvo Korhosen SOM (self-organizing map) [38] on k:n keskiarvon klusterointia kehittyneempi menetelmä, jota käytetään työkaluna tutkivassa data-analyysissä. Klusteroinnin lisäksi SOM:in avulla voidaan visualisoida datapisteiden välisiä topologisia suhteita. SOM:in tuloksena muodostuu ruudukko, joka organisoi ja yhdistää viereiset klusterit. Näin datapisteiden moniulotteisia epälineaarisia riippuvuussuhteita voidaan esittää esimerkiksi kaksiulotteisena kuvana. SOM on esimerkki ohjaamattoman oppimisen neuroverkosta.

Yksinkertaisempi, muun muassa visualisoinnin helpottamiseen usein käytetty, menetelmä on pääkomponenttianalyysi (principal components analysis, PCA). Visualisoinnissa tavoitteena on esittää moniulotteinen data kaksi- tai kolmiulotteisena. Tätä varten datasta pyritään muodostamaan kaksi tai kolme sellaista komponenttia, joiden avulla data voidaan kuvata mahdollisimman hyvin. Ideana on, että koska alkuperäiset muuttujat saattavat korreloida keskenään, niistä muodostetaan sellaisia lineaarisia kombinaatioita, jotka eivät korreloi keskenään. PCA pyrkii mahdollisimman hyvin säilyttämään datassa esiintyvän varianssin. Näin ollen visualisointiin käytettäviksi pääkomponenteiksi valitaan ne, joiden varianssi on suurin.

4.2. Mitä koneoppimismallien oppiminen tarkoittaa

Koneoppimismalleihin liittyy kahdenlaisia parametrejä: hyperparametreja ja algoritmeilla optimoitavia mallin sisäisiä parametrejä. Hyperparametrien arvoilla säädellään mallin kompleksisuutta ja muita ominaisuuksia. Niiden arvot valitaan jo mallia luotaessa. Samastakin koneoppimismallista voidaan siis luoda useita kilpailevia malleja käyttämällä eri hyperparametrien arvoja. Tämän jälkeen luotu koneoppimismalli koulutetaan käyttäen koneoppimisalgoritmia, joka optimoi mallin sisäiset parametrit niin, että käytettävän koneoppimisalgoritmin häviötä (virhettä tai virheen kustannusta) laskevan funktion arvo koulutusdatalla laskettuna minimoituu. Tätä mallin sisäisten parametrien säätymistä optimointialgoritmin työn tuloksena kutsutaan mallin oppimiseksi.

Oppimisvaihe saattaa vaatia varsinkin neuroverkkoja käytettäessä suurtakin laskentakapasiteettia. Valmiiksi koulutetussa mallissa parametrit ovat vakioita, joten valmiin mallin hyödyntäminen on laskennallisesti huomattavasti kevyempää.

4.3. Mallin yleistyminen, ylisovittuminen, alisovittuminen ja regularisointi

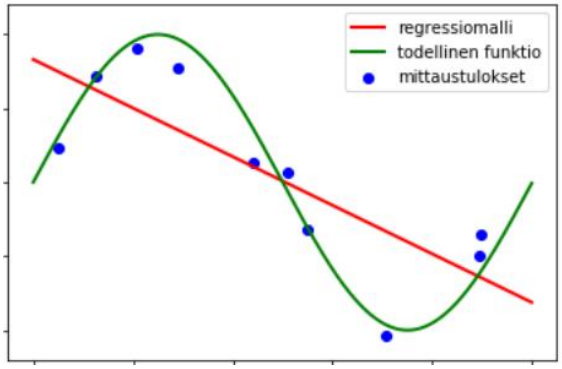
Koneoppimismallien kouluttamisen tavoitteena on, että mallit toimivat mahdollisimman oikein uusilla näytteillä, joita koneoppimisalgoritmi ei ole voinut hyödyntää oppimisvaiheessa. Mallin kykyä ennustaa oikein uudella datalla kutsutaan generalisoitumiseksi eli yleistymiseksi (generalization).

Koneoppimisen yhtenä haasteena on ylisovittuminen (overfitting). Mitä monimutkaisemman mallin valitsee, sitä varmemmin malli oppii ennustamaan oikein koulutusdatalla. Liian monimutkaisen mallin riski on, että se ylisovittuu koulutusdatan esimerkkeihin. Kun samaa mallia kutsutaan uudella datalla, jota ei ole käytetty kouluttamiseen, malli ei generalisoidukaan siihen. Alisovittunut (underfitting) malli on vastaavasti liian yksinkertainen. Sillä ei ole riittävää selityskykyä.

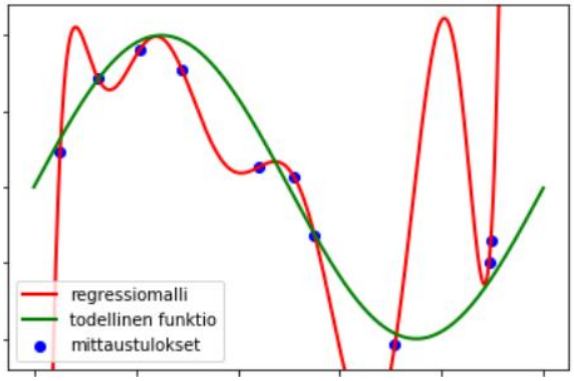
Kuvissa 4.3, 4.4 ja 4.5 on havainnollistettu tätä problematiikkaa yksinkertaisen regressioesimerkin avulla. Jokaisessa kuvassa vihreä käyrä edustaa todellista funktiota ilmiön takana. Pisteet edustavat mittaustuloksia. Niiden arvoissa on mittausepätarkkuudesta johtuen pientä heittelyä, joten ne eivät täydellisesti osu vihreän funktion arvoihin. Data-analyysin tekijä ei tiedä todellista funktiota, joten hän yrittää kokeilemalla löytää mahdollisimman hyvin ilmiötä ennustavan regressiomallin (punainen käyrä).

Kuvassa 4.3 datapisteisiin on sovitettu suora. Näemme, että malli on alisovittunut eli liian yksinkertainen. Kuvassa 4.4 datapisteisiin on vastaavasti sovitettu yhdeksännen asteen polynomi.

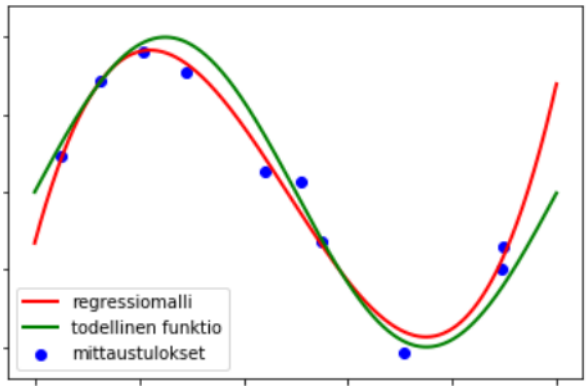
Malli on ylisovittunut mittauspisteisiin. Näiden kahden ensimmäisen regressiomallin ennustekyvykkyys uusille datapisteille on varsin huono. Kuvassa 4.5 regressiomallina on käytetty kolmannen asteen polynomia. Se mallintaa todellista ilmiötä parhaiten. [35]



Kuva 4.3: Esimerkki alisovittuneesta regressiomallista (ensimmäisen asteen polynomi eli suora).



Kuva 4.4: Esimerkki ylisovittuneesta regressiomallista (yhdeksännen asteen polynomi).



Kuva 4.5: Esimerkki hyvin yleistyvästä regressiomallista (kolmannen asteen polynomi).

Mallin generalisointikykyä voidaan usein parantaa lisäämällä koulutusdataa ja käyttämällä yksinkertaisempia malleja. Regularisointi on yksi tapa ehkäistä koneoppimismallien ylisovittumista koulutusdataan. Siinä minimoitavaan kustannusfunktioon lisätään regularisaatiotermi, joka rankaisee liian monimutkaisista malleista. Regressiomallien yhteydessä tällaisia malleja kutsutaan käytetystä regularisaatiotavasta riippuen ridge regressioniksi [39], lassoksi [40] tai elastic netiksi [41].

4.4. Automatisoitu koneoppiminen (AutoML)

Automatisoitu koneoppiminen eli AutoML tarkoittaa koneoppimismallien kehittämiseen liittyvän prosessin automatisointia. Tavoitteena on tehdä prosessista niin automatisoitu, ettei sen käynnistämiseen ja ajamiseen tarvita erityisosaamista. Käytännössä tätä kunnianhimoista tavoitetta on helpompi lähestyä automatisoimalla tiettyjä prosessin vaiheita.

Tyypillinen automatisoitava prosessin vaihe on erilaisten kilpailevien mallien kouluttaminen. Tällöin aloitustietoina annetaan esimerkiksi koneoppimismallin tyyppi sekä kyseisen koneoppimismallin käyttämien hyperparametrien mahdolliset arvot listoina. Automatisoitu prosessi kouluttaa kyseisen koneoppimismallin joko kaikilla tai algoritmin valitsemilla hyperparametrien arvokombinaatioilla, laskee kaikkien mallien kyvykkyysmetriikat validointidatalla ja asettaa näin mallit paremmuus-järjestykseen. Näin automatisoitu prosessi on mahdollista jättää ajautumaan esimerkiksi yön yli itsekseen ja kehittäjä voi analysoida tuloksia aamulla töihin tullessaan.

4.5. Mallikoosteiden hyödyntäminen

Mallikoosteiden (ensemble learning) hyödyntäminen perustuu teoriaan, että kun yhdistetään eri luokittelijoita, joista jokainen ennustaa paremmin kuin satunnainen arvaaminen, satunnaiset virheet kumoavat toisensa ja oikeat päätökset vahvistuvat [34].

Erilaisia luokittelijoita voidaan opettaa käyttämällä eri koneoppimismalleja tai samaakin mallia eri hyperparametrien arvoilla. Mallien tulosten yhdistämiseen voi käyttää esimerkiksi painotettua äänestämistä (weighted voting).

Toinen lähestymistapa on käyttää mallin kouluttamiseen erilaisia otoksia koulutusdatasta. Tällaisia menetelmiä ovat esimerkiksi bagging [42] ja boosting [43]. Bagging on näistä yksinkertaisempi. Siinä koulutusdatasta otetaan aina sopiva satunnainen otos niin, että samat näytteet on mahdollista valita yhä uudelleen. Otoksia muodostetaan siis yhtä monta kuin erillisiä malleja halutaan kouluttaa.

Boostingissa datan näytteitä painotetaan. Jokaisessa iteraatiossa koulutetaan uusi malli samalla algoritmilla. Iteraation jälkeen datan näytteitä painotetaan sen mukaan, mitkä näytteet iteraation malli ennusti väärin. Lopullinen luokittelu tehdään käyttämällä kaikkien iteraatioissa koulutettujen mallien painotettua äänestystä niin, että parhaiten ennustaneiden luokittelijoiden ääniä painotetaan enemmän.

4.6. Koneoppimismallien ennustekyvykkyyden arviointi

Koneoppimismallien kyvykkyyttä arvioidaan kahdessa eri älykkään data-analyysiprosessin vaiheessa ja siksi niitä varten tarvitaan myös kaksi erillistä datasettiä: validointidata ja evaluointidata.

Mahdollisimman hyvän koneoppimismallin löytäminen on tutkivaa työtä, joten mallinnusvaiheessa koulutetaan useampia kilpailevia malleja. Kouluttamiseen käytetään koulutusdataa. Lähes jokaisella koneoppimismallilla on omia hyperparametrejä, joita säätämällä mallin kyvykkyyteen voidaan vaikuttaa.

Kilpailevia koneoppimismalleja syntyy siis useita. Koska tavoitteena on löytää malli, joka generalisoituu mahdollisimman hyvin uusiin näytteisiin, kilpailevien mallien keskinäistä kyvykkyyttä vertaillaan käyttämällä erillistä validointidataa. Evaluointivaiheessa, kun paras malli on valittu, sen kyvykkyys arvioidaan jälleen täysin erillisellä datalla eli evaluointidatalla. Näin voidaan varmistua evaluointitulosten objektiivisuudesta datan osalta.

Kun ennustekyvykkyyttä arvioidaan, on tärkeää laskea datasta sellainen vertailuarvo, joka on mahdollista saavuttaa ilman, että koneoppimismalli on varsinaisesti oppinut vielä mitään. Palaan näihin jokaisen arviointitavan esittelyn yhteydessä. Olen käyttänyt seuraavissa kappaleissa lähteenä pääosin Airolaa [35].

4.6.1. Ristiin validointi

Jos käytettävissä olevaa dataa on runsaasti, ei koulutus-, validointi- ja evaluointidatan erottaminen toisistaan ole ongelma. Joskus koneoppimisella ratkaistava ongelma on kuitenkin sellainen, että dataa on käytettävissä vain vähän (esimerkiksi 100 näytettä). Silloin voidaan käyttää ristiin validointia (cross validation).

Ristiin validointitapoja on jätä-yksi-pois (leave-one-out) ja k-ositettu (K-Fold) ristiin validointi. K-ositetussa ristiin validoinnissa näytteet jaetaan k:hon osaan. Tämän jälkeen erotetaan yksi osa kerrallaan validointidataksi. Malli koulutetaan lopuilla näytteillä ja sen jälkeen ennuste tehdään ulos jätetyllä validointidatalla. Kun tämä on toistettu kaikilla näytteillä eli k kertaa, lopputuloksena on saatu tehtyä ennuste jokaiselle näytteelle. Näiden tulosten perusteella on mahdollista laskea todellisten arvojen ja ennustettujen arvojen välinen virhe koko mallille.

Jätä-yksi-pois ristiin validointi on k-ositetun ristiin validoinnin erikoistapaus, jossa k on yhtä suuri kuin näytteiden määrä. Mallin koulutuksessa jätetään siis aina yksi näyte kerrallaan ulos ja vastaavasti ennuste tehdään vain sille yhdelle ulos jätetylle näytteelle.

Koska sekä validointi että evaluointi on tärkeää tehdä riippumattomilla datoilla, on mahdollista käyttää sisäkkäistä ristiin validointia. Siinä sisäinen silmukka hoitaa validoinnin eli parhaan mallin valitsemisen, ja ulkoinen silmukka vastaa evaluoinnista.

4.6.2. Regressiomallin tarkkuus

Regressiomallien tavoitteena on ennustaa vastaus reaalilukuna. Yksittäiseen näytteeseen liittyvä virhe tarkoittaa siis todellisen arvon ja ennustetun arvon välistä eroa.

Yleinen tapa arvioida regressiomallin tarkkuutta on käyttää virheiden neliöiden keskiarvoa (mean squared error). Tämän laskentamallin ongelmana on, että se korostaa isojen virheiden merkitystä (koska ne korotetaan toiseen potenssiin). Näin yksittäinen iso ennustevirhe johtaa suureen kokonaisvirheen arvoon, vaikka malli toimisi muilla näytteillä hyvinkin tarkasti. Jotta voidaan arvioida, kuinka paljon koneoppimismalli on todellisuudessa oppinut, käytetään vertailukohteena mallia, joka ennustaa aina koulutusaineiston oikeiden arvojen keskiarvoa.

Toinen tapa arvioida regressiomallin kyvykkyyttä on laskea virheiden itseisarvojen keskiarvo (mean absolute error) ja verrata tätä lukuun, joka saadaan, jos malli ennustaisi aina koulutusdatan oikeiden arvojen mediaania.

4.6.3. Luokittelumallin tarkkuus

Luokittelumallin tarkkuus (accuracy) voidaan mitata laskemalla mikä osuus ennusteista osui oikeaan. Tarkkuus saa siis aina arvon nollan ja ykkösen väliltä. Jos tarkkuus on 1, malli ennusti kaikkien näytteiden luokat oikein, ja jos tarkkuus on 0, kaikki ennusteet menivät pieleen. Jotta saataisiin selvitettyä, onko malli oppinut mitään, tarkkuutta verrataan lukuun, joka saataisiin, jos olisi aina ennustettu suurinta luokkaa. Tämä on tärkeä asia sisäistää, ettei erehdy luulemaan mallia hyväksi pelkästään sen perusteella, että sen tarkkuus on lähellä yhtä.

Otetaan esimerkiksi koneoppimismalli, jonka pitäisi ennustaa onko sähköposti roskapostia vai ei. Oletetaan, että 1% sähköposteista on roskapostia. Jos malli ennustaisi aina, ettei kyseessä ole roskaposti, mallin tarkkuus olisi 0.99. Tarkkuus näyttäisi siis hyvältä, vaikka todellisuudessa malli ei olisi oppinut mitään.

Vastaavasti voidaan laskea väärinluokitteluaste (misclassification rate), joka on virheellisesti luokiteltujen näytteiden osuus. Tarkkuuden ja väärinluokitteluasteen summa on siis yksi.

4.6.4. Kustannusmatriisi

Joskus luokitteluvirheen tekemisen seurausten hinta tai vakavuus voi vaihdella luokkakohtaisesti. Otetaan esimerkiksi koneoppimismalli, jonka pitäisi ennustaa, onko tehtaalla valmistettu tuote viallinen vai ehjä. Jos malli ennustaa, että ehjä tuote on viallinen, tästä syntyy kustannuksia, koska tuote täytyy valmistaa uudelleen. Jos taas malli ennustaa, että viallinen tuote on ehjä, tästä syntyy vaikeammin hallittavia kustannuksia liittyen viallisen tuotteen logistiikkaan ja mahdolliseen maineen menetykseen, mikäli viallinen tuote on ehtinyt asiakkaalle asti. Kustannusmatriisia on havainnollistettu alla kuvassa 4.6.

		Ennustettu luokka	
		Ehjä	Viallinen
Todellinen luokka	Ehjä	0	kustannus 1
	Viallinen	kustannus 2	0

Kuva 4.6: Esimerkki kustannusmatriisista (cost matrix).

Tilanteessa, jossa luokitteluvirheiden kustannukset riippuvat siitä, mitkä luokat ovat sekoittuneet keskenään ja nämä erilaisten virheiden kustannukset on voitu arvioida, voidaan virheellisille ennusteille laskea kustannusten odotusarvo (expected loss).

4.6.5. Sekaannusmatriisi

Luokittelutehtävän onnistumista kuvataan usein sekaannusmatriisina (confusion matrix). Se muistuttaa rakenteeltaan kustannusmatriisia eli matriisin rivit edustavat todellisia luokkia ja sarakkeet ennustettuja luokkia. Tällä kertaa matriisin arvot kuvaavat näytteiden lukumääriä.

Otetaan esimerkiksi kolmeluokkainen luokittelutehtävä. Testiaineistossa jokaisesta luokasta on 100 näytettä. Näin ollen jokaisen rivin summan on oltava 100. Oikein ennustettujen näytteiden lukumäärät sijaitsevat matriisin diagonaalilla eli niissä todellinen luokka ja ennustettu luokka ovat samat. Jos oikeasti luokkaa 3 edustavista näytteistä 20 on virheellisesti ennustettu edustavan luokkaa 2, tämä luku 20 näkyy rivin kolme sarakkeessa kaksi. Sekaannusmatriisia on havainnollistettu kuvassa 4.7.

		Ennustettu luokka		
		luokka 1	luokka 2	luokka 3
Todellinen luokka	luokka 1	90	0	10
	luokka 2	0	70	30
	luokka 3	0	20	80

Kuva 4.7: Esimerkki sekaannusmatriisista (confusion matrix)

4.6.6. Binääristen luokittelutehtävien kyvykkyysmittareita

Binäärisessä luokittelutehtävässä luokkia on vain kaksi: positiivinen ja negatiivinen. Tällöin sekaannusmatriisissa on neljä kenttää: Oikea positiivinen (True Positive, TP), väärä positiivinen (False Positive, FP), väärä negatiivinen (False Negative, FN) ja oikea negatiivinen (True Negative, TN). Näiden perusteella on määritelty joukko yleisesti käytettyjä kyvykkyysmittareita. Binäärisen luokittelutehtävän sekaannusmatriisia on havainnollistettu kuvassa 4.8.

		Ennustettu luokka	
		Positiivinen	Negatiivinen
Todellinen luokka	Positiivinen	Oikea positiivinen (True Positive, TP)	Väärä negatiivinen (False Negative, FN)
	Negatiivinen	Väärä positiivinen (False Positive, FP)	Oikea negatiivinen (True Negative, TN)

Kuva 4.8: Binäärisen luokittelutehtävän sekaannusmatriisi.

Muun muassa tiedonhaussa käytetään mittareita precision (tarkkuus) ja recall (saanti). Precision ilmaisee haun tuottamien relevanttien dokumenttien osuuden tietokannassa olevista kaikista relevanteista dokumenteista. Se lasketaan siis TP / (TP + FP). Recall ilmaisee haun tuottamien relevanttien dokumenttien osuuden tietokannassa olevista kaikista relevanteista dokumenteista ja lasketaan TP / (TP + FN).

Precisionin ja recallin arvoja on helppo manipuloida toistensa kustannuksella. Precisionin voi maksimoida luokittelemalla ainoastaan ne näytteet positiivisiksi, joiden oikeellisuudesta ollaan varmimpia. Vastaavasti recallin voi maksimoida luokittelemalla kaiken positiiviseksi. Tämän vuoksi

käytetäänkin usein näiden yhdistelmää F_1-scorea, joka lasketaan 2*precision*recall / (precision + recall). Kun mallin kyvykkyyttä arvioidaan F_1-scoren perusteella, sitä kannattaa verrata malliin, jossa ennustetaan aina positiivista luokkaa, sillä silloin recall saa arvon 100%.

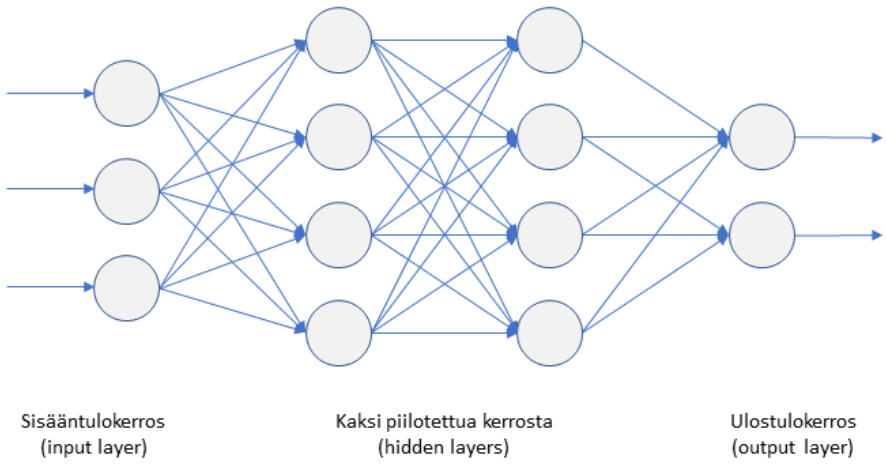
Varsinkin lääketieteellisissä diagnooseissa käytetään perinteisesti ROC-käyriä. ROC-käyrän muodostamiseksi käytetään suureita True Positive Rate (sensitivity), joka on kaavana ihan sama suure kuin recall, ja False Positive Rate (1-specificity), joka lasketaan FP / (FP + TN). ROC-käyrä piirtää True Positive Raten suhteessa False Positive Rateen. Se siis kuvaa suhteellista vaihtokauppaa hyötyjen (oikeat positiiviset) ja kustannusten (väärät positiiviset) välillä. Täysin satunnaisten arvausten tuloksena ROC-käyrä olisi siis viiva vasemmasta alanurkasta oikeaan ylänurkkaan. Mitä korkeammalla ennustemallin ROC-käyrä kulkee suhteessa tähän viivaan, sitä paremmin malli toimii.

Erilaisten mallien ROC-käyrien vertailuun käytetään suuretta AUC (area under curve), joka nimensä mukaisesti laskee ROC-käyrän alle jäävän alueen pinta-alan. Satunnaisen arvauksen mallin tuottama AUC-arvo olisi siten 0,5 ja täydellisen mallin AUC-arvo olisi 1.

5. Neuroverkkojen ja syväoppimisen perusteet

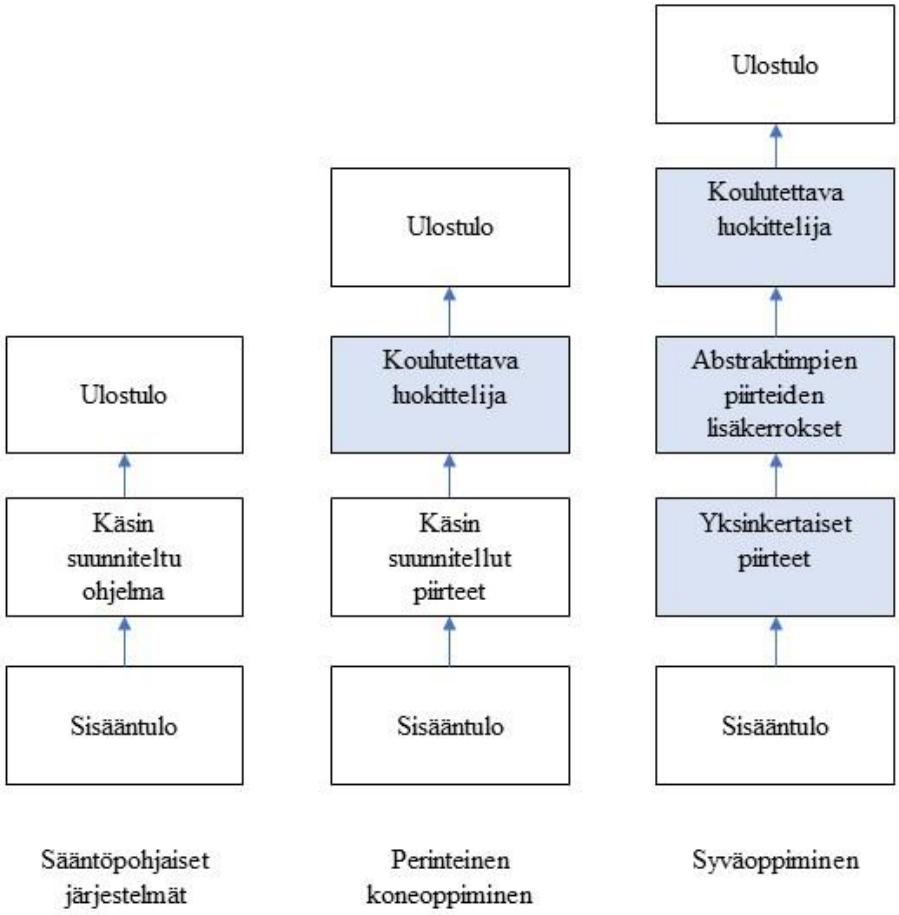
Tässä luvussa keskitytään ohjattuun oppimiseen käytettäviin neuroverkkoihin. Syvät neuroverkot muodostuvat kerroksista (layers), joista jokainen sisältää useita neuroneja. Ensimmäistä kerrosta kutsutaan sisääntulokerrokseksi (input layer) ja viimeistä kerrosta ulostulokerrokseksi (output layer). Näiden välissä voi olla useita erilaisia piilotettuja kerroksia (hidden layers). Verkko voi olla haarautuva ja sisään- ja ulostuloja voi olla useampia. Syväoppiminen on yksi koneoppimisen alalaji, jolla tarkoitetaan syvien neuroverkkojen oppimista. Syväoppimisen sanalla ’syvä’ viitataan piilotettujen kerrosten lukumäärään.

Eteenpäin kytketyssä neuroverkossa (feedforward neural network tai multilayer percepton, MLP), neuronit vastaanottavat syötteitä vain edeltävien kerrosten neuroneilta. Informaatio liikkuu verkossa siis koko ajan eteenpäin. Kuvassa 5.1 on esimerkki eteenpäin kytketystä neuroverkosta. Jos eteenpäin kytkettyyn neuroverkkoon lisätään silmukoita, niitä kutsutaan toistuviksi neuroverkoiksi (recurrent neural network). Jätän toistuvat neuroverkot pääsääntöisesti tämän perehdytysmateriaalin ulkopuolelle.



Kuva 5.1: Esimerkki eteenpäin kytketystä neuroverkosta (feedforward neural network)

Edellä luvussa 4.1 kerroin, miten kuvaavien piirteiden muodostamisesta voi muodostua perinteisten koneoppimismallien hyödyntämisen pullonkaula. Syvien neuroverkkojen vahvuus on, että ne osaavat muodostaa yksinkertaisista konsepteista monimutkaisia konsepteja [44]. Näin piirteiden oppiminen tapahtuu osana neuroverkkomallin oppimista. Sääntöpohjaisten järjestelmien, perinteisen koneoppimisen ja syväoppimisen prosessien vertautumista toisiinsa on havainnollistettu kuvassa 5.2.



Kuva 5.2: Sääntöpohjaisten järjestelmien, perinteisen koneoppimisen ja syväoppimisen prosessien komponenttien vertautuminen toisiinsa. Siniset laatikot kuvaavat komponentteja, jotka osaavat oppia datasta. Kuva on muokattu Goodfellow et al.:in kirjan [45] sivulla 10 esitetystä kuvasta.

Otetaan kuvantunnistus esimerkiksi syväoppimisen hierarkkisesta etenemisestä neuroverkon eri tasoilla. Siinä ensimmäinen piilotettu kerros etsii reunoja vertailemalla vierekkäisten pikseleiden kirkkautta tosiinsa. Toinen piilotettu kerros etsii kulmia ja muita ääriviivoja yhdistämällä ensimmäisen kerroksen löytämiä reunoja. Kolmas piilotettu kerros etsii kokonaisia objekteja yhdistämällä toisen kerroksen löytämiä kulmia ja ääriviivoja. Lopulta näiden kolmannen kerroksen muodostamat piirteet annetaan syötteenä koulutettavalle luokittelijalle, jonka tehtävänä on ennustaa, mitä kuva esittää. [45]

Neuroverkkojen avulla on saavutettu huimia edistysaskelia verrattuna perinteisiin koneoppimismalleihin. Neuroverkot ovat näyttäneet voimansa erityisesti kuvantunnistuksessa [46], puheentunnistuksessa [47] ja kuten jo edellä luvussa kolme tutustuimme, NLP:ssä. Jos esimerkiksi Googlen hakukenttään kirjoittaa ”AI cancer”, saa tuloksena lukuisia hakutuloksia, joissa kerrotaan, miten syöpädiagnostiikkaa on pystytty kehittämään kuvantunnistuksen avulla.

Monimutkaisten neuroverkkojen kouluttamiseen vaaditaan usein huomattavasti suurempia määriä dataa kuin perinteisten koneoppimismallien kouluttamiseen. Jos dataa on vain vähän saatavilla, perinteiset koneoppimismenetelmät toimivat usein vähintään yhtä luotettavasti. [48]

Toinen neuroverkkoihin liittyvä haaste on niiden vaatima suuri laskentakapasiteetti. Neuroverkkojen laskenta perustuu tensorilaskentaan ja laitteistojen kehityksessä onkin hyödynnetty näytönohjaimista tuttuja graafisia suorittimia (Graphical Processing Unit, GPU). Tämän lisäksi Google on lanseerannut neuroverkkojen opettamiseen erityisesti kehitetyn tensorisuorittimen (Tensor Processing Unit, TPU), jota voi käyttää yhdessä Googlen TensorFlown kanssa. TPU:n on raportoitu olevan jopa 10 kertaa nopeampi ja energiatehokkaampi kuin parhaimmat GPU:t. [49]

Kun neuroverkkoja lähdetään opettamaan, tarvitaan dataa, neuroverkkomalli, optimoitava häviöfunktio (loss function tai cost function) ja optimointialgoritmi (optimizer). Kerron näistä seuraavissa aliluvuissa tarkemmin. Yksi merkittävimmistä tekijöistä neuroverkkojen voittokululle on ollut siirto-oppiminen (transfer learning). Kerron siitä viimeisessä aliluvussa. Olen käyttänyt seuraavissa aliluvuissa yleislähteenä Knuutilaa [48].

5.1. Tensorimuotoinen data

Tensorit ovat neuroverkkojen käyttämiä monidimensioisia datamalleja. Tensoreissa voi olla periaatteessa mielivaltainen määrä dimensioita, mutta yleisimmin ne ovat maksimissaan viisidimensioisia. Dimensioiden hahmottamista helpottaa, kun niitä miettii esimerkkien kautta.

Vektoreissa on vain yksi dimensio, matriiseissa niitä on kaksi. Matriisi- eli taulukkomuotoinen data on tuttu datan esitysmuoto esimerkiksi Excelistä. Matriisin joka rivillä on tietyn näytteen tiedot, ja näytteiden eri piirteet sijaitsevat matriisin sarakkeissa.

Jos käytettävä data sisältää aikasarjoja (tai muita sarjoja), datamalliin tarvitaan yksi dimensio lisää. Data voi silloin muodostaa esimerkiksi kuution, jonka akselit ovat näytteet, aika-askeleet ja piirteet. Myös yksi kuva voidaan esittää kolmiulotteisena tensorina. Yhden kuvan pikselit muodostavat matriisin, ja kun siihen lisätään joka pikselin väriä kuvaavat tiedot, on tuloksena kolmiulotteinen tensori. Mustavalkokuvissa pikselin tummuus ilmoitetaan yhdellä luvulla, värikuvissa kolmella (punainen, vihreä ja sininen, eli RGB).

Useita kuvia sisältävästä datasta muodostuu siis nelidimensioisia tensoreita. Sen akselit ovat näytteet, korkeus, leveys ja värit. Koska me ihmiset elämme kolmiulotteisessa maailmassa, sen yli menevien dimensioiden ymmärtäminen vaatii jo abstraktimpaa avaruudellista hahmottamista.

Videot muodostuvat peräkkäisistä kuvista eli kehyksistä (frame), joten jokainen video muodostaa nelidimensioisen tensorin. Kun data sisältää useita videonäytteitä, on dimensioita jo yhteensä viisi (näytteet, aika, korkeus, leveys ja värit).

5.2. Neuroverkkomallit

Eteenpäin kytketty neuroverkko koostuu kerroksista ja kerrokset neuroneista. Kun neuroverkkomallia lähdetään suunnittelemaan, valitaan ensin, kuinka monesta ja minkälaisista kerroksista se rakennetaan, ja kuinka monta neuronia kullekin kerrokselle sijoitetaan. Lisäksi valitaan, miten nämä neuronit kytkeytyvät toisiinsa. Kerrosten lukumäärää kutsutaan syvyydeksi ja neuronien määrää kussakin kerroksessa niiden leveydeksi.

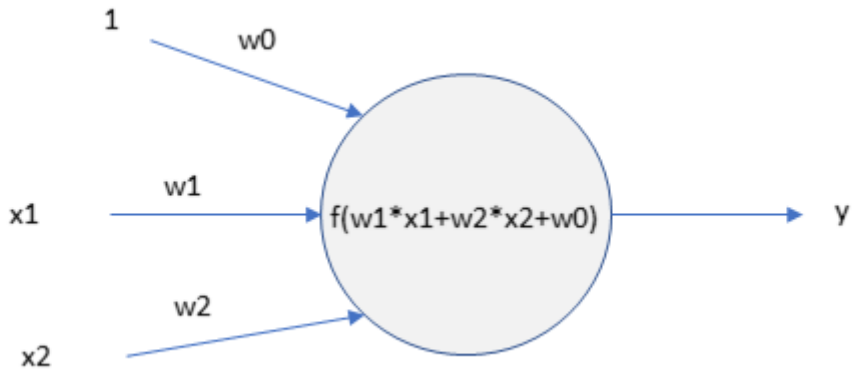
Kuvassa 5.1 oli esimerkki täysin yhdistetystä (fully connected) eteenpäin kytketyn neuroverkon arkkitehtuurista, jossa kaikki edellisen kerroksen neuronit on yhdistetty seuraavan kerroksen neuroneihin. Jokaista verkon neuroneja yhdistävää nuolta vastaa matemaattisessa mallissa yksi optimoitava parametri. Monissa erikoistuneissa neuroverkkomalleissa, kuten esimerkiksi konvoluutioneuroverkoissa, yhteyksiä on vähemmän, jolloin myös optimoitavia parametreja ja laskentatarvetta on vähemmän.

Seuraavissa aliluvuissa on esitelty neuronien toimintaa sekä Valtorin tekoälyratkaisussa käytettävät erilaiset neuroverkon kerrokset.

5.2.1. Neuroneista koostuva peruskerros

Yksittäisen neuronin toiminta on varsin yksinkertaista. Se vastaanottaa syötteitä muilta neuroneilta ja laskee niiden perusteella uuden arvon, jonka se lähettää eteenpäin syötteenä seuraaville neuroneille. Käytännön laskennan tasolla neuroneita ei kuitenkaan esiinny, vaan tässä kuvaustavassa on kyse abstarktiosta, jonka avulla matemaattista mallia yritetään tehdä ymmärrettävämmäksi.

Yksittäisen neuronin toimintaa on kuvattu kuvassa 5.3. Jokaiselle syötteelle on oma painokerroin (w1, w2, w0), jonka arvo määräytyy kyseisen syötteen suhteellisen merkityksen perusteella muihin syötteisiin nähden. Neuroni laskee ensin kaikkien vastaanottamiensa syötteiden painotetun summan ja sen jälkeen sijoittaa painotetun summan arvon parametrinä aktivointifunktion (f) lausekkeeseen. Tämän aktivointifunktion tuloksen (y) neuroni lähettää eteenpäin seuraaville neuroneille.



Kuva 5.3: Yksittäisen neuronin toiminta.

Jos painotettuja summia käytettäisiin sellaisenaan, neuroni voisi oppia vain lineaarisesti (suoralla, tasolla tai hypertasolla) toisistaan erotettavissa olevia luokkia. Tämän vuoksi tarvitaan lisäksi epälineaarinen aktivointifunktio (f). Aktivointifunktioita on lukuisia, mutta tällä hetkellä suositelluin piilotettujen kerrosten aktivointifunktio on ReLU (Rectified Linear Unit) [50, 51, 52], joka lasketaan kaavalla f(x)=max(0, x). ReLU saa siis arvon 0, aina kun x on negatiivinen ja muuten se saa arvon x. ReLUn suosio perustuukin sen tehokkaaseen laskentaan ja sen avulla helposti tehtävään optimointiin. Muita yleisesti käytettyjä aktivointifunktioita ovat muun muassa Logistic Sigmoid ja Tanh. Logistic Sigmoid kutistaa reaaliluvut nollan ja ykkösen välille. Se lasketaan kaavalla $\sigma(x) = 1 / (1 + \exp(-x))$. Tanh puolestaan voi saada arvoja väliltä -1 ja 1. Sen kaava on $\tanh(x) = 2\sigma(2x) - 1$.

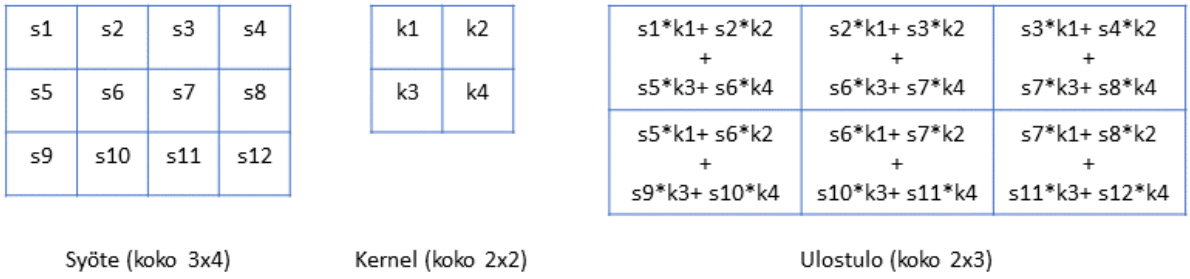
Yksittäinen neuroni voi muodostaa yhden päätösrajan (decision boundary) eli se pystyy erottamaan syötteen kahteen luokkaan. Näin ollen n neuronia pystyy luokittelemaan jo 2^n luokkaan. Mikä tahansa funktio on mahdollista approksimoida monen paikallisen funktion summana. Teoriassa jo kaksi piilotettua tasoa sisältävällä eteenpäin kytketyllä neuroverkolla, joissa on riittävä määrä epälineaarisia neuroneja, voidaan kuvata mitkä tahansa monimutkaiset päätösalueet [53].

Neuroverkoilla on siis teoriassa rajaton selityskapasiteetti. Haasteena onkin, että ne voivat liiankin helposti "oppia ulkoa" opetusdatan ja ylisovittua siihen oppimalla sellaisiakin piirteitä opetusdatasta, jotka eivät ole yleistettävissä uuteen dataan. Tällainen ylisovittunut malli ei siis toimi luotettavasti esimerkiksi validointidatalla. Tämän vuoksi, kun neuroverkkoja opetetaan, validointidatalla usein seurataan saatavien tulosten kehittymistä ja lopetetaan opettaminen, kun validointidatalla lasketut tulokset alkavat huonontua. Tätä kutsutaan early stopping -menetelmäksi.

5.2.2. Konvoluutiokerros

Konvoluutioneuroverkoiksi (convolutional neural networks, CNN) [54] kutsutaan neuroverkkoja, joissa vähintään yksi tavallisista neuroneista koostuva kerros on korvattu konvoluutioita laskevilla neuroneilla. Konvoluutioneuroverkot ovat erikoistuneet ruudukkomuotoisen (grid-like) datan käsittelyyn kuten kuviin [54] ja tekstiin [55]. Konvoluutiolla viitataan matemaattiseen lineaariseen operaatioon, konvoluutioon. Konvoluutiokerroksen ideana on, että malli oppii, onko datassa tiettyä piirrettä. Se oppii tunnistamaan kyseisen piirteen siitä riippumatta, missä kohtaa esimerkiksi kuvaa tai tekstiä kyseinen piirre sijaitsee (translational invariance). Tämä ominaisuus ei kuitenkaan koske esimerkiksi kuvan skaalausta tai rotaatiota.

Konvoluution laskeminen on helpoin ymmärtää yksinkertaisen esimerkin kautta (kuva 5.4). Oletetaan, että sisään tulevan syötteen koko on 3x4, ja sen yli liu'utetaan kerneliä, jonka koko on 2x2. Kerneliä liu'utetaan syötteen päällä kaikkiin sellaisiin kohtiin, joihin se mahtuu kokonaisuudessaan. Tämän kokoinen kernel mahtuu kokonaisena syötteen "päälle" yhteensä kuuteen eri kohtaan (kolmeen vierekkäiseen kohtaan kahden ylimmän rivin päälle ja kolmeen vierekkäiseen kohtaan kahden alemman rivin päälle). Ulostulosolujen arvo lasketaan kullekin kernelin paikalle syötteen päällä niin, että päällekkäisten solujen arvot kerrotaan keskenään ja nämä kaikki lasketaan yhteen. Näin ollen konvoluution lopputulos on kokoa 2x3.



Kuva 5.4: Esimerkki konvoluution laskemisesta.

Joskus syötteeseen lisätään ympärille täytteenä (padding) esimerkiksi 0-arvoisia syötteitä, jotta kernel saadaan liu'utettua myös niiden kohtien yli, joihin se ei muuten mahtuisi. Toisaalta joskus kernelin halutaan hyppäävän esimerkiksi joka toisen kohdan yli, silloin määritellään askeleen (stride) pituudeksi 2.

Täysin yhdistetyssä neuroverkossa, kaikilla edellisen kerroksen neuroneilla on yhteys kaikkien seuraavan kerroksen neuronien kanssa (dense). Näin ollen optimoitavia parametrejä on yhtä monta kuin näitä neuronipareja. Konvoluutiokerroksessa liu'utetaan samaa kerneliä joka kohdassa, joten yhteyksistä tulee paljon harvempia (sparse) ja jäljelle jäävissä yhteyksissä käytetään yhteisiä

parametrejä (parameter sharing) (kuvassa 5.4: k1, k2, k3 ja k4). Konvoluutiota hyödyntävä malli tarvitsee siis vähemmän muistia ja on tilastollisesti tehokkaampi. Pienempi parametrien määrä yleensä johtaa myös parempaan generalisoitumiseen. [48]

Konvoluutiokerros muodostuu yleensä kolmesta alikerroksesta. Ensimmäisessä alikerroksessa lasketaan konvoluutiot. Toisessa alikerroksessa konvoluution tulos sijoitetaan aktivointifunktion parametriksi. Myös tässä tapauksessa aktivointifunktio siis tarvitaan, jotta malli osaisi mallintaa epälineaarisia päätösrajoja.

Kolmannessa alikerroksessa saatuja tuloksia yhdistetään (pooling) laskemalla niistä yhteenvetostatistiikkaa. Jälleen kerrokseen tulevan syötteen päällä voidaan ajatella liukuvan tietyn kokoinen filtteri (esimerkiksi kokoa 2x2 oleva filtteri, joka askelpituudella 2 hyppää joka toisen paikan yli). Ulostulosolujen arvo lasketaan tällä kertaa jokaiselle filtterin paikalle syötteen päällä valitsemalla kyseisen alueen (local pooling) maksimi- tai keskiarvo. Global poolingissa tulevasta syötteestä lasketaan vain yksi maksimi- tai keskiarvo. Näin on mahdollista vähentää piirteiden dimensiota. Tällöin ei haittaa, vaikka sisään tulevan datan koko vaihtelisi (esimerkiksi lauseen pituus).

5.2.3. Long short-term memory (LSTM) -kerros

Long short-term memory (LSTM) [56, 57] on laajasti käytetty toistuvan neurovekon (recurrent neural network) kerrosarkkitehtuuri. Kun eteenpäin kytketyssä neuroverkossa (feedforward neural network), neuronit vastaanottavat syötteitä vain edeltävien kerrosten neuroneilta, toistuvassa neuroverkossa voi olla lisäksi silmukoita. Siksi se pystyy analysoimaan yksittäisten datapisteiden lisäksi myös sarjoja, eli esimerkiksi yksittäisten sanojen lisäksi lauseita.

LSTM-moduuli koostuu neljästä neuroverkon kerroksesta, jotka ovat vuorovaikutuksessa keskenään. Solun tila (state) pitää muistissaan juoksevasti informaation arvoja. Sitä verrataankin usein liukuhihnaan. Solun tilaan voidaan lisätä tai siitä voidaan poistaa informaatiota. Tätä säädellään sisääntulo- (input gate), ulostulo- (output gate) ja unohdaporttien (forget gate) avulla.

Syötevektorin arvot (input) lasketaan tavallisen neuroverkon neuronin avulla käyttäen haluttua aktivointifunktiota. Sen arvot lisätään solun tilaan ainoastaan niiltä osin, mitä sisääntuloportti sallii. Unohdaportti säätelee mitä solun tilan pitäisi unohtaa aikaisemmasta informaatiosta. Lopuksi ulostuloportti määrittelee mitkä tiedot solun tilasta päästetään ulos.

Kaikissa porttisoluissa käytetään Sigmoid-aktivointifunktiota, joka puristaa niiden arvot 0:n ja 1:n välille. Nämä arvot kuvaavat, kuinka paljon kutakin komponenttia tulisi päästää portista läpi. Arvo

0 siis tarkoittaa, ettei portti päästä mitään läpi, ja arvo 1 tarkoittaa, että kaikki päästetään portista läpi.

5.2.4. Dropout-kerros

Neuroverkkojen ylisovittumista pyritään ennaltaehkäisemään dropout-kerroksella [58]. Yksi tapa luoda erilaisia luokittelijoita, on käyttää luokittelijoiden kouluttamiseen erilaisia otoksia koulutusdatasta. Neuroverkkojen tapauksessa käytännöllinen tapa muodostaa lukuisia erilaisia aliverkkoja on lisätä dropout-kerros, joka maskaa jokaisessa oppimisiteraatiossa valitun osuuden kerroksen solmuista pois kertomalla ulostuloarvon nollalla.

5.2.5. Erän normalisointi -kerros

Erän normalisointi -kerrosta (batch normalization) [59] käytetään stabiloimaan piilotettuihin kerroksiin tulevien syötteiden jakaumaa. Kerros standardoi siihen tulevat syötteet normaalijakaumaksi, jonka keskiarvo on 0 ja keskihajonta 1. Näin optimoitavasta ”maastosta” saadaan tasaisempi.

Syötteiden normalisointi mahdollistaa huomattavasti isompien oppimisnopeuksien (learning rate) käytön, koska silloin aktivointifunktion arvot pysyvät maltillisina ja optimointi nopeutuu. Mallista tulee myös robustimpi hyperparametrien valinnalle sekä parametrien aloitusarvojen asettamiselle (initialization). Lisäksi malli generalisoituu paremmin. [59]

5.3. Optimoitavat häviöfunktiot

Häviöfunktion (loss function tai cost function) valinta riippuu ennustetehtävän luonteesta. Sen tehtävänä on mitata mallin tuottamien ennusteiden ja oikeiden vastausten välistä eroa. Neuroverkon oppiminen tarkoittaa neuronien välisten yhteyksien painojen säätymistä niin, että häviöfunktio saa pienimmän mahdollisimman arvon. Oikean häviöfunktion valinta on siis erittäin tärkeää.

Yleisesti käytetyt luokittelu- ja regressiotehtävien häviöfunktiot on johdettu perustuen suurimman uskottavuuden estimaatin (maximum likelihood estimate, MLE) teoriaan.

5.4. Optimointialgoritmi

Neuroverkon oppiminen tarkoittaa neuronien välisten yhteyksien painojen säätymistä niin, että häviöfunktio saa pienimmän mahdollisimman arvon. Tämä tapahtuu usean iteratiivisen oppimiskierroksen tuloksena. Yksi oppimiskierros sisältää viisi askelta [49]:

1. Koulutusaineistosta otetaan erä (batch) näytteitä, joka sisältää näytteiden piirteet ja oikeat vastaukset.
2. Näyte-erä ajetaan neuroverkon läpi (forward pass), joka laskee näytteille ennusteet.
3. Ennusteiden ja oikeiden vastausten ero lasketaan häviöfunktiota käyttäen.
4. Lasketaan häviöfunktion paikallinen gradientti (backward pass tai backpropagation). Gradientti muodostuu vektorin kaikista osittaisderivaateista.
5. Kaikkia neuroverkon painoja säädetään niin, että häviöfunktion arvo hiukan pienenisi (eli vastakkaiseen suuntaan kuin gradientti).

Funktion arvon iteratiivista minimoimista (tai maksimoimista) perustuen paikallisen gradientin arvon laskentaan ja painojen säätämiseen liikkumalla pitkin gradienttia kutsutaan gradienttimenetelmäksi (gradient descent) [60].

Häviöfunktion minimointitehtävää voidaan ajatella maisemana, josta etsitään syvintä kohtaa. Oikeasti maisemassa voisi olla lukuisia dimensioita, mutta ajatusleikissämme voimme rajoittua meille tuttuun kolmiulotteiseen maan pintaan. Siinä tapauksessa korkeus edustaa häviöfunktion arvoa. Maisemassa voi olla useita kukkuloita, laaksoja, jyrkänteitä, solia jne. Jokaisessa tasaisessa kohdassa, jossa alamäki muuttuu ylämäeksi tai toisin päin, gradientti saa arvon 0. Kääntäen, kun gradientti saa arvon 0, kyseessä voi olla lokaali tai globaali minimi tai maksimi tai satulapiste (toiseen suuntaan minimi ja toiseen maksimi).

Tavoitteena on löytää globaali minimi käyttäen numeerista optimointia. Nyt siis meidät tiputetaan parametrien satunnaisen alustamisen tuloksena vieraaseen maastoon johonkin satunnaiseen kohtaan. Tiedämme koordinaatit (missä kohtaa olemme leveys- ja pituuspiireillä, eli parametrien arvot, sekä korkeutemme), mutta näemme maisemasta vain juuri sen kohdan, jossa seisomme. Koska etsimme syvintä kohtaa, otamme suunnaksi alamäen (eli vastakkaisen suunnan kuin paikallinen gradientti).

Tässä vaiheessa joudumme sanomaan algoritmille etukäteen, kuinka pitkän matkan kuljemme valitsemaamme suuntaan. Tätä etukäteen määriteltyä matkan pituutta kutsutaan oppimisnopeudeksi (learning rate). Hyvällä tuurilla päädymme alemmaksi ja satumaisen hyvällä tuurilla satuimme olemaan juuri siinä alamäessä, joka vie kohti globaalia minimiä. Mutta voi käydä niinkin, että lähestymme vain pientä kuoppaa ja globaali minimi olisi ihan muualla. Tai edessä onkin seinämä,

mutta koska kuljemme siihen suuntaan etukäteen ilmoitetun matkan, saatamme päätyä hyvinkin korkealle.

Edellä esitettyihin moniulotteisen numeerisen optimoinnin haasteisiin on yritetty kehittää monenlaisia algoritmejä. Koska ne menevät jo aika syvälle matematiikkaan, jätän niiden käsittelyn tällä kertaa tähän. Käytännössä kuitenkin on siis mahdollista kokeilla optimointia erilaisilla algoritmeillä ja niiden hyperparametreillä. Yksi tärkeimmistä hyperparametreista on oppimisnopeus (learning rate), joka säätelee, kuinka paljon parametreja säädetään yhdellä oppimiskierroksella.

5.5. Siirto-oppiminen

Neuroverkkojen opettamisessa tarvitaan paljon dataa ja laskentaa. Siirto-oppimisella (transfer learning) [61] pystytään hyödyntämään aiemmin opetettuja neuroverkkoja oman neuroverkon pohjana. Edellä olemme jo tutustuneet luonnollisen kielen käsittelyn yhteydessä esikoulutettuihin sanapotusvektoreihin sekä BERT-kielimalliin. Nämä ovat hyviä esimerkkejä siitä, miten esikouluttamalla malli ensin ohjaamattomassa tehtävässä, jossa on paljon dataa, voidaan saavuttaa parempia tuloksia ohjatussa tehtävässä, jossa dataa on käytettävissä vähemmän [55].

Usein organisaatiolla itsellään ei edes ole riittävästi dataa käytettävissä täysin oman mallin kouluttamiseen. Kun alemman tason piirteet on esikoulutettu isolla datamäärällä, voi omaan tarpeeseen räätälöidyn neuroverkon opettaminen onnistua pienemmälläkin aineistolla.

Luonnollisen kielen käsittelyn lisäksi siirto-oppimisen hyödyntäminen on hyvin yleistä kuvantunnistuksen tehtävissä. Kuvantunnistukseen on vuosien varrella kehitetty yhä parempia konvoluutioneuroverkkoarkkitehtuureja. Niistä monet ovat avoimesti käytettävissä osana kirjastoja sekä malliarkkitehtuureina että esiopetettuina verkkoina. Koska neuroverkot oppivat piirteiden spatiaalisia hierarkioita, alemman tason piirteet voivat olla hyvinkin hyödyllisiä, vaikka räätälöitävä luokittelutehtävä koostuisi aivan eri aiheen kuvista.

6. Case 1: Tekstidokumenttien luokittelu

Tässä luvussa esittelen tekstidokumenttien luokitteluun käytettävien ennustemallien kehittämistä tutkivana prosessina konkreettisen esimerkin avulla. Olen aiemmin tehnyt Jupyter Notebookilla [62] työkirjan, jossa esittelen tekstidokumenttien luokittelua myös kooditasolla. Työkirja on kokonaisuudessaan liitteenä sekä lisäksi luettavissa ja ladattavissa lähdeluettelosta löytyvän linkin kautta [7].

6.1. Luokitteluun käytetty data

Demodatana on käytetty Kielipankin CC-BY-NC 4.0 lisenssillä jakamaa Ylilauta-korpusta (http://urn.fi/urn:nbn:fi:lb-2016101210), jota Turun yliopiston kieli- ja puheteknologian apulaisprofessori Sampo Pyysalo on edelleen muokannut yksinkertaiseen (aihe, teksti) TSV-muotoon ja tuottanut siitä menetelmien vertailuun tasapainotetun kymmenen luokan datasetin (https://github.com/spyysalo/ylilauta-corpus).

Ylilauta on anonyymi keskustelufoorumi, joka ei vaadi rekisteröitymistä tai nimimerkin käyttöä. Ylilauta-korpukseen on tallennettu kyseisen keskustelufoorumin keskustelupalstoja vuosilta 2014-2016. Tässä työssä käytettyyn suppeampaan aineistoon on otettu sama määrä viestejä kymmenestä yleisimmin esiintyvästä aiheluokasta. Nämä aiheluokat ovat:

1. Ajoneuvot
2. Hikky (Hikikomero, masentuneiden ja sosiaalisesti syrjäytyneiden vertaistukiryhmä)
3. Kuntosali
4. Muoti
5. Pelit
6. Penkkiurheilu
7. Politiikka
8. Seksuaalisuus
9. Sota
10. Televisio

Aineisto on jaettu valmiiksi kolmeen osaan:

- Koulutusdata erilaisten mallien kouluttamiseen (10 000 kpl/aihe, yht 100.000 kpl)

- Validointidata mallien parametrien optimoimiseen ja keskinäiseen vertailuun, jotta voidaan valita näistä paras (1 000 kpl/ aihe, yhteensä 10 000 kpl)
- Testidata, jota käytetään parhaimmaksi valitun mallin luotettavuuden arviointiin (1 000 kpl/ aihe, yhteensä 10 000 kpl)

6.2. Erilaisten mallien tutkiva kehittäminen

Tavoitteena on kehittää malli, joka osaa mahdollisimman hyvin ennustaa viestien aiheluokan niiden tekstisisällön perusteella. Tutkiva prosessi aloitetaan yksinkertaisemmista malleista, joiden opettaminen onnistuu nopeasti vaikka tavallisella kannettavalla tietokoneella. Näin saadaan asetettua ennustekyvykkyyden lähtötaso, jota yritetään enemmän laskentatehoa vaativilla malleilla parantaa.

Jokaista koneoppimismallityyppiä koulutetaan useilla eri hyperparametrien kombinaatioilla ja jokaisen koulutetun mallin ennustekyvykkyys analysoidaan käyttäen validointidataa. Kyseisen mallityypin voittavaksi malliksi valitaan se malli, jonka hyperparametreillä saavutetaan paras validointitulos. Koska validointiaineistoa on käytetty hyperparametrien optimoimiseen ja parhaan mallin valitsemiseen, evaluoidaan vielä parhaiden mallien luotettavuus täysin erillisellä testidatalla.

Lopullisen tuotannossa käytettävän koneoppimismallin valintaan vaikuttaa sekä mallin ennustekyvykkyys että sen vaatima laskentakapasiteetti. Mikäli erot eri mallien kyvykkyyksissä jäävät pieniksi, eikä erolla ole merkittävää liiketoiminnallista vaikutusta, tai käytettävissä ei ole jatkuvasti tehokasta laskentaympäristöä, ja mallia pitäisi tuotannossa kouluttaa jatkuvasti, kannattaa valita laskennallisesti yksinkertaisempi malli.

6.2.1. Naive Bayes

Aloitin yksinkertaisella todennäköisyyksiin perustuvalla Naive Bayes -luokittelumenetelmällä. Käytin tekstin vektorisointiin tf_idf-menetelmää. Koulutin eri hyperparametrien arvoilla yhteensä noin 20 erilaista malliversiota. Muunneltavia parametrejä olivat: käytetäänkö yksittäisiä sanoja vai myös 2- ja/tai 3-grammeja, poistetaanko pysäytyssanat sekä kuinka harvinaiset sanat poistetaan. Eri kombinaatioilla tarkkuus validointidatalla vaihteli 70,6 %:n ja 73,1 %:n välillä. Parhaan tarkkuuden (73,1%) saavutin käyttämällä yksittäisiä sanoja sekä poistamalla pysäytyssanat ja erittäin harvinaiset sanat (min_df-arvolla 0.00003).

Tutkin myös kuvassa 6.1 näkyvää sekaannusmatriisia apua käyttäen, miten eri luokat erottuvat toisistaan. Matriisin rivit edustavat oikeita luokkia ja sarakkeet ennusteita. Diagonaalilla näkyy siis kyseisen luokan oikein luokiteltujen viestien lukumäärät. Esim. ensimmäisellä rivillä näkyy niiden

dokumenttien luokitteluennusteet, joiden todellinen luokka on yksi eli 'Ajoneuvot'. Näistä 1000:sta luokkaan yksi kuuluvasta viestistä 772 on luokiteltu oikein, neljä on ennustettu virheellisesti luokkaan kaksi 'Hikky', jne. Sekaannusta näyttäisi tapahtuvan kaikkien luokkien välillä.

```
[772   4  13  11  17   9  11  11  24   5]
[ 42 741  69  66  72  49 107 112  70  55]
[ 31  20 766  41  35  44  14  49  23  31]
[ 29  13  30 753  19  23   7  33  23  15]
[ 19  11   8  20 690  62  11  12  25  33]
[  7   6  13   7  36 685  10   8  19   8]
[ 29  56  13  23  22  31 792  13 151  64]
[ 35 119  57  46  27  27  22 745  26  29]
[ 16   9  12  10  25  20   7  13 617  15]
[ 20  21  19  23  57  50  19   4  22 745]
```

Kuva 6.1: Sekaannusmatriisi.

Paras validointitulos käyttäen Naive Bayes -menetelmää jäi 73.1%:in tarkkuuteen. Täysin riippumattomalla testiaineistolla Naive Bayes -menetelmän tarkkuudeksi saatiin 71,6%

6.2.2. Tukivektorikone (support vector machine)

Seuraavaksi tutkin, minkälaiseen tarkkuuteen pääsen käyttämällä tukivektorikone-menetelmää. Koulutin ensin useita malliversioita käyttäen Scikit-learnin [63] SGDClassifier-mallia, mutta kaikki nämä mallit jäivät kyvykkyydeltään huonommiksi kuin Naive Bayes. Tämän vuoksi vaihdoin malliksi Scikit-lernin LinearSVC-mallin.

Tässä mallissa vaihtelin seuraavia hyperparametrien arvoja: poistetaanko pysäytyssanat, kuinka harvinaiset sanat poistetaan, käytetäänkö 2-grammeja, säätelyparametrin arvo sekä iteraatioiden maksimimäärä. Sain parhaan validointituloksen käyttäen koko sanastoa, ilman 2-grammeja ja säätelyparametrin arvolla 0.6. Validointidatalla tarkkuudeksi saatiin 74,0 % ja testidatalla 72,0 %.

6.2.3. Muita perinteisiä koneoppimismalleja

Dokumenttien vektorisoinnissa syntyy todella pitkiä vektoreita. Tämä rajaa merkittävästi, mitä koneoppimismenetelmiä voidaan tavallisella kannettavalla käyttää. Kokeilin kahta hyvin yleisesti käytettyä koneoppimismenetelmää (K nearest neighbors sekä Regularized linear model with ridge regression), mutta molemmat kaatuivat. Nearest neighbors ilmoitti kyseessä olevan muistin loppuminen, kun taas Ridge-menetelmän käyttö kaatoi Jupyter notebookin prosessin.

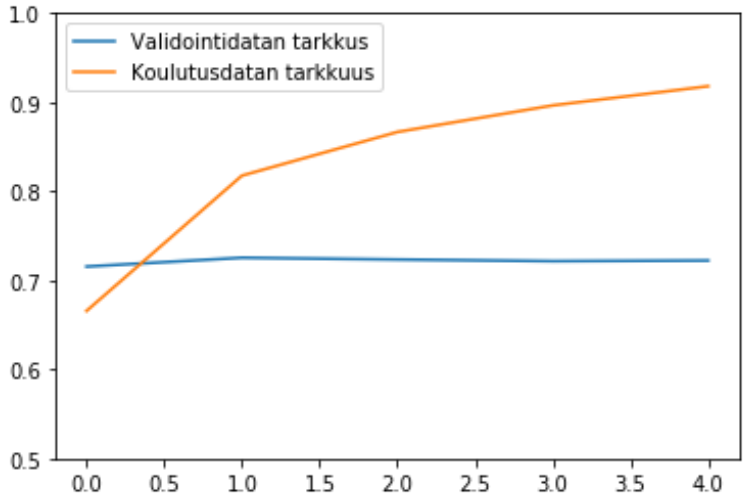
SGDClassifier-metodia voidaan käyttää myös muiden mallien kuin tukivektorikoneen käyttöön, mutta koska niiden alustavat tulokset eivät olleet kovin lupaavia, en lähtenyt tutkimaan näitä menetelmiä tarkemmin.

6.2.4. Täysin yhdistetty eteenpäin kytketty neuroverkko

Seuraavaksi tutkin yksinkertaisia täysin yhdistettyjä eteenpäin kytkettyjä neuroverkkoja, joissa on vain yksi tai kaksi piilotettua tasoa. Jos olisin ottanut kaikki sanat mukaan, vektoreista olisi tullut lähes 400 000:n pituisia. Koska käyttämäni tietokoneen kapasiteetti ei olisi riittänyt niin suuren tietomäärän työstämiseen, rajoitin piirteiden määrän 100 000 sanaan.

Kokeilin ensin neuroverkkomallia, jossa on yksi piilotettu taso, jossa on 128 neuronia. Käytin Early stopping -menetelmää, jossa joka optimointikierroksen jälkeen mallin virhemäärä lasketaan validointidatan avulla. Jos virhe ei pienene esim. kolmen viimeisen kierroksen aikana koulutusprosessi pysäytetään. Opetuksen tuloksena saatu malli käyttää koulutuksen aikana validoituja parhaita parametriarvoja.

Kuvassa 6.2. on esitetty kuvaaja, miten mallin tarkkuus kehittyi koulutuksen aikana. Oranssi käyrä näyttää mallin tarkkuuden kehittymisen koulutusdatalla ja sininen käyrä validointidatalla. Kuviosta huomataan, ettei validointidatan tarkkuus parane toisen kierroksen jälkeen (x-akselin indeksi 0 vastaa ensimmäistä validointitulosta, joka lasketaan ensimmäisen kierroksen jälkeen). Jos riittävän isoa neuroverkkoa koulutetaan riittävän kauan, sen tarkkuus koulutusdatalla lähenee aina 100 %:ia. Sen sijaan validointidatalla laskettu tarkkuus alkaa jossain vaiheessa huonontua, koska opittu malli ei enää ylisovittumisesta johtuen yleisty siihen.



Kuva 6.2: Mallin tarkkuuden kehittyminen koulutuksen aikana.

Tarkkuudeksi saatiin validointidatalla 72,6%, joka on huonompi kuin tukivektorikoneella tai Naive Bayesilla saatu tarkkuus.

Näinkin yksinkertaisen neuroverkon koulutus vei kannettavalla tietokoneellani noin 40 minuuttia. Neuroverkoilla olisi huikea selitysvoima, mutta niiden opettamiseen vaaditaan myös paljon

laskentatehoa. Tavallisella tietokoneella erilaisten yksikertaistenkin mallien koulutus ja niiden hyperparametrien virittäminen on helposti tuntien tai jopa päivien urakka.

Koulutin kymmenisen erilaista yksinkertaista neuroverkkoa muuttamalla vektorisointitapoja, vaihtelemalla kerrosten ja neuronien lukumääriä, tiputtamalla pois pysäytyssanoja ja hyvin harvinaisia sanoja, käyttämällä 2- ja 3-grammeja, säätämällä oppimisnopeutta, vaihtamalla optimointialgoritmia sekä lisäämällä dropout-kerroksen.

Parhaalla yksinkertaisella neuroverkkomallilla validointitarkkuudeksi saatiin 75.5 % ja testidatalla 73,6 %

6.2.5. Kannettavalla tietokoneella koulutettu konvoluutioneuroverkko käyttäen fastText-sanaupotuksia

Suomen kielessä sanoilla on useita taivutusmuotoja ja sanajohdoksia. FastText sopii hyvin suomenkielisten sanojen vektoriesitysten muodostukseen, koska se hyödyntää mallissaan myös sanojen alimerkkijonoja.

Tietokoneeni rajallisen muistikapasiteetin vuoksi otin malliin mukaan vain 100 000 yleisintä sanaa. Jos sanalle ei löydy fastTextin mallista valmiiksi opetettua upotusvektoria, sitä vastaavan vektorin kaikki luvut jäävät matriisissa arvoon 0. Tässä tapauksessa upotusvektori löytyi 86 570 sanalle.

Pelkkä sanaupotusten hyödyntäminen ei tuottanut parempaa mallia, joten lisäsin neuroverkkoon myös konvoluutio- (filttereiden lukumäärä 128 ja kernelin koko 5) ja dropout-kerrokset. Tällä mallilla tarkkuudeksi saatiin validointidatalla vain 69.4 %.

Koitin vielä kouluttaa saman mallin pienentämällä oppimisnopeutta. Paras validointitulos 70,1% saatiin viimeisellä kierroksella, joten tulos olisi voinut vielä hiukan parantua, mikäli koulutusta olisi jatkettu kauemmin. Mallin koulutus kesti näinkin yli 36 tuntia, joten neuroverkon hienosäätö tavallisella koneella alkoi käydä mahdottomaksi.

Testidatalla tarkkuudeksi saatiin 68,5%. Tulos jäi huonommaksi kuin muilla menetelmillä. Sanaupotusvektoreita oli koneeni muistirajoitteiden vuoksi käytössä vain 86 570 sanalle, kun koko sanaston koko olisi ollut lähes 400 000. Tämä on todennäköisin syy sille, ettei sanaupotusvektoreita hyödyntämällä päästy parempaan tulokseen. Edellä ilman esikoulutettuja sanaupotuksia koulutettu neuroverkko koulutettiin 100 000 sanalla ja sillä päästiin jo merkittävästi parempaan tarkkuuteen 73,6%.

6.2.6. CSC:n supertietokoneella koulutettu konvoluutioneuroverkko käyttäen fastText-sanaupotuksia

Kannettavan tietokoneeni kapasiteettirajoitteiden vuoksi sanaupotuksia hyödyntävien neuroverkkojen luokittelutarkkuus jäi edellä varsin pieneksi. Seuraavaksi tukin, millaisiin tuloksiin on mahdollista päästä CSC:n supertietokoneen Puhtin avulla, kun malliin voidaan ottaa mukaan kaikki koulutusdatan sanat ja hyperparametrien optimointikin on mahdollista järkevässä ajassa.

Kun upotussanamatriisiin otettiin mukaan kaikki koulutusaineiston 399 349 sanaa, FastText-upotusvektori löytyi 212 960 sanalle eli noin 53 %:ille. Tämä on huomattavasti enemmän kuin aikaisemmassa matriisissa, jossa upotusvektorit löytyivät vain 86 570 sanalle.

Koulutin 11 erilaista neuroverkkoa. Niiden opettaminen supertietokoneella kesti yhteensä muutaman tunnin. Halusin ensin kokeilla, mihin tarkkuuteen päästään samanlaisella neuroverkkomallilla kuin olin kannettavalla tietokoneella kouluttanut, mutta kun käytettävissä on kaikki sanat. Tarkkuudeksi validointidatalla tuli nyt 72,4 % eli tulos parani huomattavasti. Kun annoin neuroverkon jatkaa fastText:in esikoulutettujen upotussanavektoreiden edelleen kouluttamista luokittelutehtävän omalla koulutusaineistolla, tarkkuus validointidatalla nousi vielä muutaman prosenttiyksikön. Tämän jälkeen kokeilin vielä erilaisia malleja muuttamalla seuraavia hyperparametrien arvoja: kernelin koko, filttereiden lukumäärä ja oppimisnopeus.

Kun paras malli valitaan validointidatan perusteella, paras malli saatiin koulutettua arvoilla: filttereiden lukumäärällä 256, kernelin koolla 5 ja oppimisnopeudella 0.0001. Tämän mallin tarkkuudeksi testidatalla saatiin 74,2%

6.3. Yhteenveto kehitetyistä malleista

Eri koneoppimismalleilla päästiin seuraaviin ennustetarkkuuksiin testiaineistolla:

- Naive Bayes 71,6 %
- Tukivektorikone 72,0 %
- Täysin yhdistetty eteenpäin kytketty neuroverkko 100 000 sanalla 73,6 %
- Konvoluutioneuroverkko esikoulutetuilla sanaupotuksilla, 100 000 sanalla 68,5 %
- Konvoluutioneuroverkko esikoulutetuilla sanaupotuksilla, kaikilla sanoilla, sanaupotusten edelleen koulutuksella 74,2 %.

Paras ennustetarkkuus saavutettiin siis käyttäen konvoluutioneuroverkkoa esikoulutetuilla sanaupotuksilla, jossa neuroverkko vielä edelleen koulutti sanaupotuksia. Eri mallien koulutuksen

vaatimassa laskentakapasiteettitarpeessa on kuitenkin todella isoja eroja. Naive Bayes -mallien kouluttaminen käy silmän räpäyksessä jopa tavallisella kannettavalla. Sen sijaan erilaisten isojen neuroverkkojen opettaminen järjellisessä ajassa vaatii jo hyvin tehokasta laskentaa.

Vertailun vuoksi voidaan todeta, että samalle aineistolle on tehty myös FinBERT-kielimallia hyödyntävä luokittelumalli, jonka tarkkuudeksi on evaluoitu 82,8 % [23]. Samassa tutkimuksessa M-BERT-kielimallilla tarkkuudeksi saatiin 77,4 % ja fastTextillä 74,7 %.

7. Valtorin tekoälyalusta

Valtori on hankkinut koneoppimiseen pohjautuvien käyttötapausten alustaksi tekoälyalustan, joka koostuu IT-infrastruktuurista sekä sen päällä ajettavasta tekoälyratkaisusta. Ratkaisun avulla on mahdollista kehittää koneoppimismalleja ja tarjota niitä rajapinnan kautta palveluina. Ratkaisu mahdollistaa myös käyttötapauksiin liittyvien työnkulkujen toteuttamisen. Tavoitteena on kehittää yhteinen ratkaisukehys, jota hyödyntäen uusien käyttötapausten toteuttaminen olisi mahdollisimman yksinkertaista. Tekoälyalustaa on tulevaisuudessa tarkoitus käyttää myös Valtorin asiakkaiden koneoppimiseen pohjautuvien käyttötapausten alustana.

Käytössä on neljä ympäristöä: kehitys-, testi-, hyväksymistesti- (QA, quality assurance) ja tuotantoympäristö. Näistä kehitys- ja testiympäristö sijaitsevat pilvipalvelussa, ja hyväksymistesti- ja tuotantoympäristö Valtorin hallinnoimassa konesalissa virtuaalipalvelinratkaisuna. Tuotanto-ympäristössä tuotantodata on käytössä integraatioiden kautta. Koska pilviympäristössä ei ole tietoturva- ja tietosuoja syistä lupaa käyttää tuotannon dataa salaamattomana, käyttötapauskohtaisten koneoppismallien kouluttaminen tapahtuu QA-ympäristössä.

Tekoälyratkaisu hyödyntää mikropalveluarkkitehtuuria. Ratkaisun komponentteja ajetaan konttiteknologiaa hyödyntävän Kubernetes-kerroksen päällä. Palveluiden asentaminen tapahtuu automaattisten CI/CD-putkien kautta (Continuous Integration/Continuous Delivery). Versionhallinta- ja CI/CD-alustana käytetään GitLabia.

Kerron seuraavissa aliluvuissa Valtorin tekoälyratkaisusta sillä tarkkuudella, kuin se on julkisessa dokumentissa mahdollista. Tekoälyratkaisu koostuu kahdesta pääosasta: AINO:sta (AI no) ja AION:ista (AI on). AINO edustaa perinteistä tietojärjestelmäpuolta, kun taas AION:in avulla on mahdollista kehittää koneoppimismalleja.

7.1. AINO

AINO edustaa tekoälyratkaisun perinteistä tietojärjestelmäpuolta, johon voidaan koodata käyttötapausten työnkulut. Lisäksi se sisältää käyttötapausten hyödyntämiä yleispalveluita kuten auditointilokitus-, autentikointi-, koodisto-, parametri- ja dokumenttipalvelu. Koodistopalvelussa koodeille (esimerkiksi maakoodi) ja koodiarvojen selkokielisille selityksille eri kielillä voidaan antaa voimassaoloaikoja. Parametripalvelussa voidaan muokata käyttötapausten toimintaan vaikuttavien

parametrien arvoja ilman, että koodiin täytyy tehdä muutoksia. Dokumenttipalvelu osaa generoida PDF-dokumentteja. AINO:n koodauskielenä käytetään pääsääntöisesti Javaa.

Tekoälyratkaisun ulkopuoliset järjestelmät kutsuvat koneoppimista hyödyntäviä käyttötapauksia integraatioiden kautta. Pääsääntöisesti integraatiot tapahtuvat Valtion Integraatioalustan VIA:n kautta. Protokollana käytetään JSON:ia HTTP(S):n yli. AINO:on toteutetaan uusia integraatio-palveluita käyttötapausten tarpeiden mukaisesti.

7.2. AION

AION on koko tekoälyalustan ydin. Sen avulla on mahdollista kehittää koneoppimismalleja ja tarjota niitä rajapinnan yli AINO:lle. Koneoppimismallien kehittäminen tapahtuu omassa kehitysputkessaan, jonka pohjana on Googlen kehittämä TensorFlow Extended (TFX) [64, 65]. Valtorin tekoäly-ratkaisussa koneoppimiseen liittyvien ajojen orkestrointiin Kubernetes-alustalla käytetään Kubeflowta [66]. TFX-putkea ajetaan siis Kubeflown päällä.

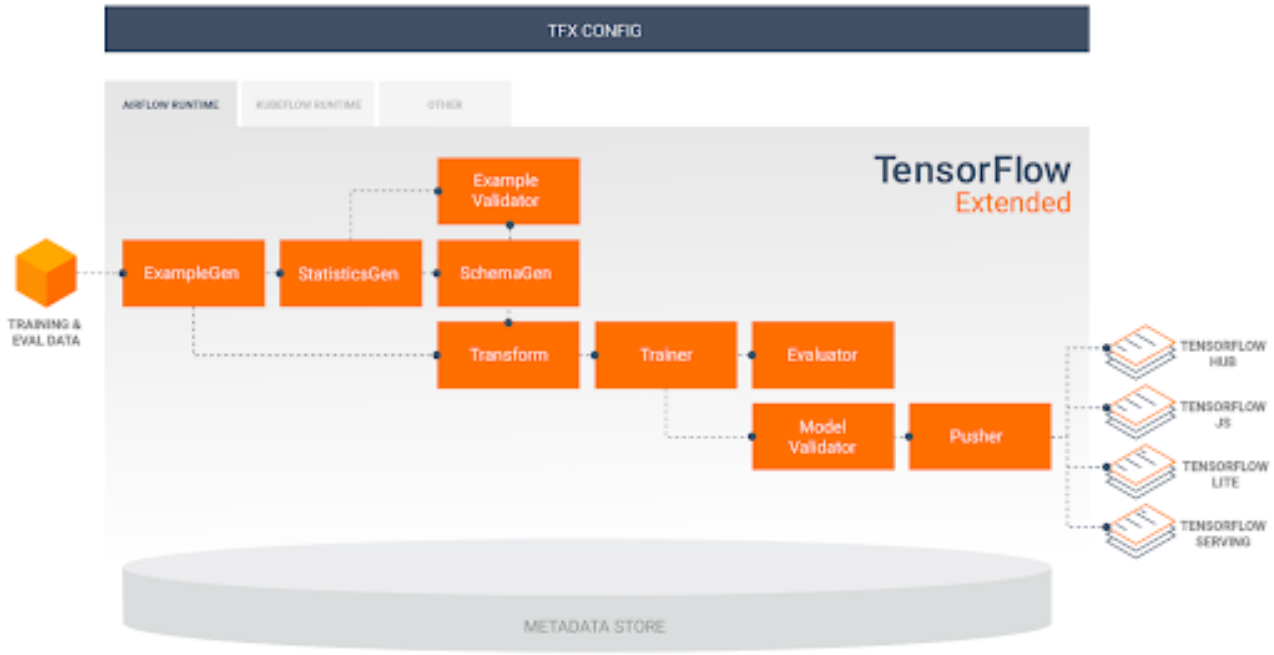
TensorFlow [67] on Googlen kehittämä ilmainen avoimen lähdekoodin koneoppimiskirjasto. Matemaattiset operaatiot on toteutettu TensorFlowssa C++:lla, mutta kirjaston hyödyntäminen onnistuu käyttäen Python-kieltä. Keras [68] on TensorFlown päälle kehitetty käyttäjäystävällisempi ohjelmointirajapinta (application programming interface, API). Vaikka sitä kutsutaan rajapinnaksi, se on tarkoitettu ihmisten (ei koneiden) käytettäväksi.

Koneoppimismallien kehittäminen ei välttämättä vaadi järeää alustaa. Varsinkin yksinkertaisten mallien kehittäminen ja kouluttaminen onnistuu vaikka kannettavalla tietokoneella. Mutta kun koneoppimismallia halutaan käyttää tuotannossa, sen pitää täyttää kaikki samat vaatimukset kuin muidenkin tietojärjestelmien. Tällaisia vaatimuksia ovat esimerkiksi skaalautuvuus, eheys, modulaarisuus, testattavuus, tietoturva ja tietosuoja. TFX on kehitetty ottamaan huomioon nämä vaatimukset.

TensorFlow Extended eli TFX-pipeline on TensorFlow:hun perustuva kokonaisvaltainen tuotantoputki koneoppimismallien kehittämiseksi, tuotantoon viemiseksi ja monitoroinniksi. TFX-pipelinen komponentit on rakennettu käyttäen neljää TFX-kirjastoa, joista jokaista on mahdollista käyttää myös erikseen. TensorFlow Data Validation -kirjasto on kehitetty datan tutkimiseen, validointiin ja monitorointiin. TensorFlow Transform -kirjastoa käytetään datan esikäsittelyyn ja vektorisointiin. TensorFlow Model Analysis -kirjasto mahdollistaa kehitettyjen mallien kyvykkyyden monipuolisen analysoinnin. TensorFlow Serving -kirjasto tukee mallien versioinnissa, mahdollisessa

palauttamisessa sekä usean mallin A/B-testauksessa, jossa tuotantoon viedään kaksi eri mallia, joista testijakson jälkeen valitaan paras. [64]

Luvussa kaksi tutustuimme älykkään data-analyysin prosessimalliin. Prosessin teknisiä kehitysvaiheita ovat datan sisään luku ja validointi, datan valmistelu, mallin kouluttaminen, mallin evaluointi sekä mallin käyttöönotto. TFX-putki koostuu komponenteista ja niiden parametreistä. Jokainen komponentti edustaa jotakin data-analyysiprosessin vaihetta. Kuvassa 7.1 on kuvattu TFX:n standardi komponentit. Kerron niistä tarkemmin seuraavissa aliluvuissa.



Kuva 7.1: TFX:n standardi komponentit. [65]

Jokainen komponentti koostuu kolmesta osasta: ajuri (driver), suorittaja (executor) ja julkaisija (publisher). Komponentit saavat syötteensä pääsääntöisesti kaikille komponenteille yhteisestä metatietovarastosta. Ajuri päättää metatietojen perusteella, mitä tehtäviä pitää tehdä, koordinoi komponentin työn suorittamista ja välittää metatiedot suorittajalle. Suorittaja suorittaa komponentin varsinaisen tehtävän. Julkaisija vastaanottaa tulokset suorittajalta ja päivittää tiedot metatietovarastoon. Ajurin ja julkaisijan koodit ovat lähes aina samoja komponentista riippumatta. Varsinainen räätälöitävä koodi löytyy siis toteuttajasta. Komponentit konfiguroidaan käyttäen Python-kieltä. [65]

Kehitysputki kootaan ja komponenttien suorittamista hallitaan Kubeflow-orkestraattorin avulla. Kubeflowssa on hallintaportaali, jonka avulla on mahdollista käynnistää ja seurata ajoja ja niiden komponentteja.

7.2.1. Metatietovarasto

TFX:n metatietovarasto on implementoitu käyttäen ML-Metadata-kirjastoa [69]. Metatietovarastoon tallennetaan kolmenlaista tietoa. Ensimmäinen tietoluokka koskee koulutettuja malleja, niiden kouluttamiseen käytettyä dataa sekä niiden evaluointituloksia. Tämän tyyppisiä tietoja kutsutaan artefakteiksi. Näiden tietojen osalta ei tallenneta varsinaista dataa, vaan ainoastaan varsinaisen datan sijaintitiedot ja ominaisuudet. Toisen tietoluokan muodostavat komponenttien suoritustiedot virheenkorjausta, toistettavuutta ja auditointia varten. Lisäksi tallennetaan dataobjektien alkuperä, kun ne virtaavat putken läpi. Näin on mahdollista tutkia millä tiedoilla malli on koulutettu tai miten tietty vektorisointitapa vaikutti evaluointituloksiin. [65]

Metatietovaraston käyttö mahdollistaa sen, ettei aina ole pakko ajaa koko putkea alusta alkaen. Jos on tehnyt muutoksia vain tiettyyn komponenttiin, riittää, että putken käynnistää kyseisestä vaiheesta uudelleen. Näin voi säästää paljon aikaa ja laskentakapasiteettia. Myös tietyn mallin koulutusta voi jatkaa siitä tilanteesta, mihin on viimeksi jäänyt. Tällainen tilanne voisi tulla vastaan esimerkiksi silloin, kun haluaa kokeilla ensin muutamaa mallia rajoitetulla kierrosmäärällä ja jatkaa vain tulosten perusteella lupaavimman mallin kouluttamista. Koko mallin kehityksen historian tallentaminen metatietoihin mahdollistaa mallin kyvykkyyden monitoroinnin ja tutkimisen myös pitkällä aikavälillä, kun mallia käytettään tuotannossa ehkä vuosia.

7.2.2. Datan sisään luku ja validointi

Valtorin tekoälyratkaisussa datan sisään luku, tutkiminen ja ihan ensimmäinen esikäsittely tapahtuu Data ingest -nimisessä Jupyter Notebookissa. Näihin tehtäviin on mahdollista käyttää muitakin kirjastoja, mutta TFX:n TensorFlow Data Validation -kirjasto on kehitetty datan tutkimiseen ja analysoimiseen. Data ingestin toiminta on osin rinnakkainen kuvassa 7.1 esitettyjen ExampleGen-, StatisticsGen-, SchemaGen- ja ExampleValidator-peruskomponenttien kanssa.

Data ingest -notebookissa data luetaan sisään eri lähteistä, muodostetaan tarvittaessa uusia sarakkeita ja tehdään muita muutoksia sekä valitaan ja nimetään mallin käyttämät sarakkeet. Tämän jälkeen kaikille valituille piirteille lasketaan niitä kuvailevat tilastotiedot (esimerkiksi puuttuvien tietojen sekä uniikkien arvojen prosenttiosuudet). Datan kuvailuun on mahdollista käyttää myös visualisointityökaluja. Näiden avulla päästään tutkimaan datan laatua ja ominaisuuksia. Skeema-työkalut tutkivat datasta laskettua statistiikkaa ja yrittävät näin päätellä piirteiden perusominaisuuksia kuten tietotyyppejä, arvoalueita ja luokkia. Skeemaa on mahdollista myös itse säätää esimerkiksi lisäämällä uusia luokkia. Tilastotietojen ja skeeman avulla tutkitaan datan laatua, kuten puuttuvia arvoja tai arvoja, jotka eivät sovi skeemaan. Myös datan vinoumista ja siirtymistä on mahdollista

saada raporttitietoa. Lopulta Data ingest tallentaa esikäsittelemänsä datan varsinaisen opetuspipelinen saataville.

Koska jokainen käyttötapaus on tehty eri tarkoitukseen ja ne käyttävät eri järjestelmissä muodostuvaa dataa, Data ingest räätälöidään käyttötapauskohtaisesti. On hyvä huomata, että Data ingestiä käytetään ainoastaan mallin kehittämiseen tarvittavan datan sisään lukuun ja käsittelemiseen. Tässä komponentissa ei saa tehdä sellaista datan esikäsittelyä, jota tarvitaan, kun valmista mallia kutsutaan rajapinnan kautta. Tällaiset esikäsittelyt tehdään vasta osana varsinaista TFX-pipelineä.

Kun TFX-pipeline käynnistetään, ExampleGen lukee Data ingest -notebookin tallentaman datan sisään ja jakaa sen erillisiin koulutus- ja evaluointidatasetteihin. StatisticsGen laskee tilastotiedot ja SchemaGen muodostaa piirteiden skeeman. ExampleValidator hyödyntää tilastotietoja ja skeemaa ja tekee näiden perusteella datan validoinnin. TFX-pipeline voidaan käynnistää Kubeflowsta, tai halutessaan voi varsinkin kehitysvaiheessa orkestraattorina käyttää myös interaktiivista Jupyter Notebookia, jolloin komponentteja voi ajaa ja niiden tuloksia seurata vielä havainnollisemmin.

7.2.3. Datan valmistelu

Datan varsinainen esikäsittely ja tarvittavat muunnokset tehdään Transform-komponentissa. Näiden koodauksessa hyödynnetään pääasiallisesti TensorFlow Transform -kirjastoa. Esimerkiksi luonnollisen kielen vektorisointi on mahdollista tehdä tässä komponentissa.

Transform-komponentti muodostaa koodissa määritellyt datamuunnosten säännöt koulutusdatan perusteella ja tallentaa ne metatietovarastoon. Koulutusdatan perusteella tallennettuja sääntöjä käytetään tämän jälkeen sekä koulutus-, evaluointi- että tuotantodatan vektorisointiin.

7.2.4. Mallinnus

Valtorin tekoälyratkaisussa TFX-pipelinen Transform- ja Trainer-peruskomponentin väliin on lisätty Tuner-komponentti. Tuner muodostaa automaattisesti esimerkiksi viisi erilaista neuroverkkoa, kouluttaa niitä jokaista esimerkiksi viisi kierrosta (epoch) ja valitsee niistä tämän perusteella parhaimman. Tämän jälkeen Trainer-komponentti kouluttaa Tunerin tallentaman parhaan mallin loppuun.

Tuner-komponentti perustuu Keras Tuner -kirjastoon [70], joka on kehitetty koneoppimismallien hyperparametrien optimointiin. Neuroverkkojen tapauksessa hyperparametreja ovat esimerkiksi kerrosten lukumäärä, kunkin kerroksen tyyppi ja leveys, oppimisalgoritmin oppimisnopeus (learning rate) sekä käytettävä aktivointifunktio ja optimointialgoritmi. Keras Tuneria olisi mahdollista käyttää

myös esimerkiksi Scikit-learn-kirjaston koneoppimismallien hyperparametrien säätöön, mutta Valtorin putkeen on toistaiseksi toteutettu ainoastaan neuroverkkojen rakenteen tuunaus.

7.2.5. Mallin evaluointi

Evaluointia on mahdollista tehdä kahdella eri työkalulla. Nämä työkalut ovat TensorBoard [71] ja TensorFlow Model Analysis -kirjaston [72] visualisointityökalut.

TensorBoardin avulla on mahdollista tarkastella mallin kouluttamisen aikaista kyvykkyyden kehittymistä. Seurattaviksi mittareiksi voi valita esimerkiksi tarkkuuden ja kustannusfunktion arvon. TensorBoardin avulla tuloksia analysoidaan koulutus- ja validointidatalla.

TFX:n Evaluator-komponentti sen sijaan laskee koulutetun mallin kyvykkyysmetriikat evaluointidatan perusteella. Tämä tekee mallien välisestä vertailusta luotettavampaa, koska eri malleja voidaan ajaa samalla kontrolloidulla testidatalla. Tässä komponentissa käytetään monipuolista TensorFlow Model Analysis -kirjastoa, jonka käytettävissä oleviin kyvykkyysmittareihin voi tutustua tarkemmin esimerkiksi TensorFlown tutoriaalimateriaalin [73] avulla. Model Analysis -visualisointityökalun avulla mallia voidaan myös tutkia pienempinä osakokonaisuuksina, niin sanottuina viipaleina.

Palaan molempiin visualisointityökaluihin hieman tarkemmin ja havainnollisemmin luvussa 8.5.

Jos tuotannossa on jo ennestään malli, ModelValidator-komponentti vertailee, onko uusi kehitetty malli parempi kuin jo tuotannossa oleva malli.

7.2.6. Mallin käyttöönotto

Pusher-komponentti tarkistaa, että malli on läpäissyt validoinnin. Tämän jälkeen se paketoi lopullisen mallin ja tallentaa sen oikeaan repositorioon. Ainakin toistaiseksi mallin varsinainen tuotantoon vienti tapahtuu Valtorin ympäristössä manuaalisesti.

Malli on mahdollista käyttöönottaa myös niin, että vanhaa ja uutta mallia käytetään tietyllä suhdeluvulla. Uuden mallin käyttöä voidaan lisätä näin vaiheittain.

7.2.7. Mallin monitorointi

On todella tärkeää, että mallin ennustekyvykkyyttä seurataan tuotannossa. TFX:n TensorFlow Model Analysis ja Data Validation -kirjastoista olisi saatavissa tähän tukea. Monitorointia ei kuitenkaan vielä ole toteutettu Valtorin TFX-ratkaisuun. TensorFlow Model Analysis -kirjaston avulla voidaan tutkia, mikä on mallin ennustekyky tuotantoon viennin jälkeen [74]. TensorFlow Data Validation -

kirjasto tukee koulutus- ja tuotannondatan välisen skeeman, piirteiden ja jakauman vinouman tunnistamista [75].

Skeeman tai piirteiden vinouma voisi syntyä esimerkiksi silloin, jos kehittäjä koodaisi ainoastaan Data ingest -notebookiin sellaista datan esikäsittelyä, joka pitäisi tehdä myös tuotannon datalle. Koska Data ingest käsittelee ainoastaan koulutus- ja evaluointidataa, siinä tehtyjä toimenpiteitä ei automaattisesti tehdä tuotannon datalle, vaan nämä käsittelyvaiheet pitää tehdä vasta varsinaisessa TFX-pipelinessä. Piirteiden vinouma syntyy myös, jos koulutusaineiston tapausten piirteiden arvo jostain syystä muuttuu opetusdatan hankinnan ja tuotannon aloittamisen välissä.

Koulutus- ja tuotantodatan välinen jakaumien vinouma voi syntyä, jos kouluttamiseen on käytetty vain osajoukkoa tuotannon tapauksista. Näin käy myös, jos tuotannon järjestelmiin tehdään muutoksia, joita ei oteta huomioon koulutusdatassa. Esimerkiksi uusien tuotteiden perustaminen johtaa vinoumaan, mikäli uusia tuotteita alkaa ilmestyä tuotannondataan ennen kuin niitä on sisällytetty koulutusdataan.

7.2.8. Muut käytettävissä olevat koneoppimiskirjastot

Valtorin TFX-pipelineä kehitetään sen mukaisesti, mitä kyvykkyyksiä kulloinkin rakennettavat käyttötapaukset vaativat. Ensimmäisessä vaiheessa on siis kehitetty ainoastaan sellaisia kyvykkyyksiä, mitä hankkeen ensimmäisen vaiheen käyttötapaukset tarvitsevat. Tällä hetkellä TFX-pipeline osaa lukea vain taulukkomuotoista dataa ja se käyttää ennustamiseen ainoastaan neuroverkkopohjaisia malleja.

Koska mallien pääasiallinen kouluttaminen tapahtuu Valtorin hallinnoimassa konesalissa, mallien kouluttamiseen voidaan käyttää ainoastaan sellaisia koneoppimiskirjastoja, jotka on asennettu konesalin palvelimille. Vaikka näiden kirjastojen käyttö ei vielä ole mahdollista osana Valtorin räätälöityä TFX-pipelineä, niitä voidaan kuitenkin jo käyttää mallien kokeiluun ja kehittämiseen Jupyter Notebookin avulla.

Nykyisissä käyttötapauksissa koneoppimiskirjastoina on käytetty lähinnä TensorFlown ja Kerasin kirjastoja. Konesaliin asennetut muut koneoppimiskirjastot käyttötarkoituksineen ovat:

- H2O, hajautettu koneoppimisen ja prediktiivisen analytiikan alusta ja työkalu [76]
- OpenCV, kuvankäsittelykirjasto (Open Source Computer Vision Library) [77]
- Scikit-image, kuvankäsittelykirjasto Pythonille [78]
- Scikit-learn, laajasti käytetty koneoppimiskirjasto Pythonille [63]
- Tesseract, tekstin tunnistaminen kuvasta [79]

8. Case 2: Tikettien luokittelu ja ohjaaminen oikeaan työjonoon

Valtorin tavoitteena on automatisoida yhä useampia manuaalisia prosesseja ja työtehtäviä. Valtorin ja Palkeiden yhteishankkeen tarkoituksena on automatisoida ja tehostaa palveluprosesseista tunnistettuja kehityskohteita tekoälyn avulla. Hankkeessa rakennetaan yhteinen tekoälyalusta sekä toteutetaan ensimmäiset valitut käyttötapaukset. Yksi Valtorin käyttötapauksista on ’Tikettien luokittelu ja ohjaaminen oikeaan työjonoon’, josta kerron tässä luvussa tarkemmin.

Hanke ja käyttötapausten kehittäminen on vielä tätä kirjoittaessani kesken. Tämän vuoksi en pysty vielä kattavasti esittelemään käytännön tuloksia. Uskon kuitenkin käyttötapauksen kehittämiseen liittyvän prosessin esittelyn kehityshuomioineen olevan opettavaista tästä huolimatta.

8.1. Liiketoiminnan ymmärtäminen

Hanketta varten Valtori ja Palkeet tekivät hankinnan, jonka kohteena oli tekoälyratkaisu, suunniteltujen käyttötapausten toteutus tekoälyratkaisun päälle, käyttöön tarvittavat ympäristöt, tuki- ja ylläpitopalvelut sekä asiantuntijatyö. Hankinnan tarjouspyyntömateriaali koostuu yhteensä 36 dokumentista, joissa hankinnan kohde ja vaatimukset on kuvattu varsin kattavasti.

Hankkeelle asetettiin ohjausryhmä, joka on vastuussa kokonaishankkeesta ja sen lopputuloksista. Lisäksi nimitettiin projektipäälliköt, jotka vastaavat hankkeen hallinnoinnista ja hankkeen vaiheiden edistymisestä. Käyttötapauksille on sovittu myös liiketoimintaomistajat, mutta heidän osallistumisensa hankkeeseen on ollut vähäistä. Jatkossa olisi tärkeää, että liiketoimintaomistajat pystyisivät allokoimaan käyttötapausten kehittämiselle enemmän aikaa ja toisaalta projektit pitäisivät heihin määrämuotoisemmin yhteyttä.

8.1.1. Aikataulu

Hankintavaiheessa aikataulutavoitteena oli, että toimitusprojekti saataisiin käynnistettyä toukokuun 2019 aikana ja käyttötapaukset olisi hyväksytty tuotantokäyttöön 31.10.2019 mennessä. Erilaisista, pääosin teknisistä haasteista, johtuen tuotantokäytön aloittaminen on kuitenkin siirtynyt aikaisintaan loppuvuoteen 2020.

Jatkossa, kun tekoälyalustan ratkaisukehys on valmis, uusien vastaavien käyttötapausten kehittäminen ja käyttöönotto voi hyvinkin olla mahdollista puolen vuoden aikaikkunoissa.

8.1.2. Kustannus-hyöty-analyysi

Hankinnan käynnistyessä on laskettu, että mikäli kaikkien kuuden toteutettavaksi suunnitellun käyttötapauksen tuomat hyödyt realisoituisivat täysimääräisesti, olisi taloudellinen hyöty vuositasolla noin kolme miljoonaa euroa [80]. Ihan näihin lukuihin ei tulla hankkeessa pääsemään, sillä yksi kuudesta käyttötapauksesta on rajattu kokonaan pois, ja muihinkin käyttötapauksiin on jouduttu tekemään muutoksia ja rajauksia.

Kustannus-hyöty-laskelmaa olisi tärkeä seurata ja päivittää hankkeen aikana jatkuvasti. Älykkään data-analyysin ja ketterän kehittämisen myötä ymmärrys sekä liiketoiminta-alueesta että koneoppimisen mahdollisuuksista ja rajoitteista kasvaa vaiheittain. Koska tavoitteena on saavuttaa kustannushyötyjä, kustannus-hyötylaskelman esilläpito voisi auttaa fokuksen säilyttämisessä olennaisissa asioissa ja antaisi myös eväitä mahdolliseen projektin keskeyttämispäätöksen tekemiseen.

Nyt kehitteillä olevat ensimmäiset käyttötapaukset toimivat referenssiratkaisuina tuleville käyttötapauksille. Näin ollen niistä voidaan katsoa olevan laajempaakin hyötyä sen lisäksi, mitä niiden lasketaan tuottavan kustannussäästöjä ja parempaa palvelua omissa kohdeprosesseissaan.

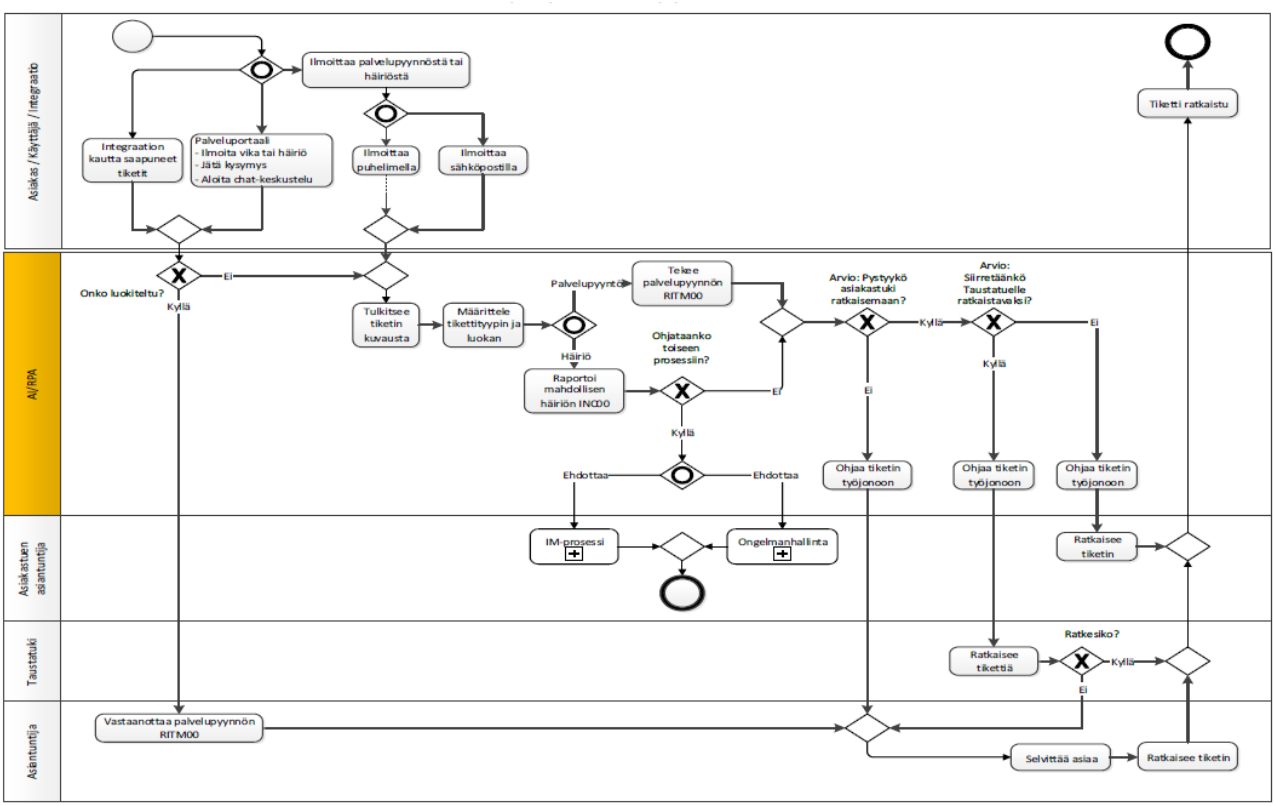
8.1.3. Tietoturvavaatimukset

Valtionhallinnon hankinnoille on leimallista korkeat tietoturvavaatimukset. Niistä oli kilpailutusasiakirjoihin koottu oma useita välilehtiä sisältävä excel-liite. Hankinnan kohteen kuvauksessa tietoturvavaatimuksista oli mainittu näin:

> ”Ratkaisun tulee perustua valtioneuvoston asetukseen tietoturvallisuudesta valtionhallinnossa (TTA 681/2010), tietoturvallisuudesta valtionhallinnossa annetun asetuksen täytäntöönpanosta laadittuun ohjeeseen (VAHTI 2/2010) ja ICT varautumisen vaatimuksiin (VAHTI 2/2012). Lisäksi ratkaisun tulee perustua KATAKRI 2015 STIV vaatimuksiin toiminnan kannalta merkittävin osin ja sovelluskehityksen tietoturvaohjeeseen (VAHTI 1/2013) sekä toimitilojen tietoturvaohjeeseen (VAHTI 2/2013). Ratkaisun tulee myös huomioida voimaan tuleva tiedonhallintalaki, joka on hyväksytty eduskunnassa 18.3.2019.
>
> Lisäksi myöhemmässä vaiheessa tulee huomioida keväällä 2019 julkaistava Kyberturvallisuuskeskuksen pilvipalveluiden turvallisuuden arviointikriteeristö (työstönimellä PiTuKri). Palvelun sisältämään asiakkaan dataan ei ole pääsyä ETA-alueen ulkopuolelta. [80]”

8.1.4. Käyttötapauksen kuvaus

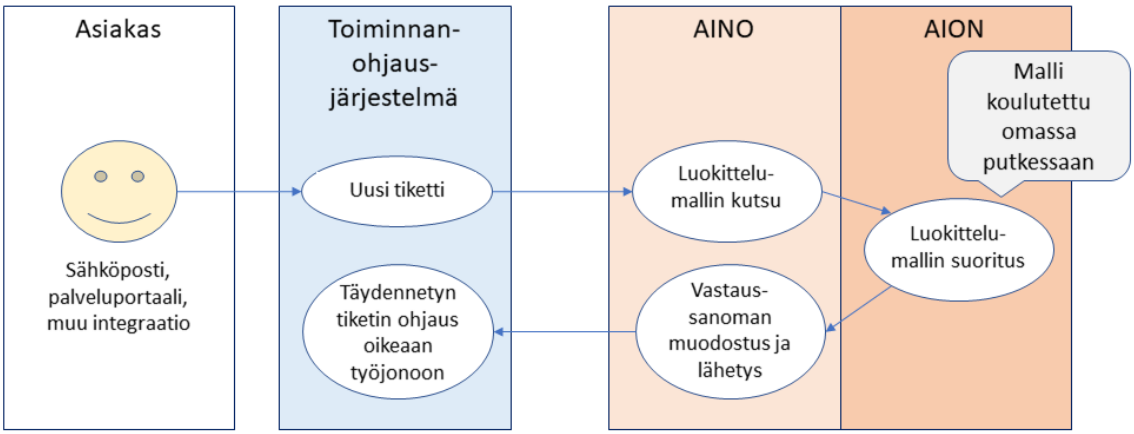
Hankinnan kohteen kuvauksessa ’Tikettien luokittelu ja ohjaaminen oikeaan työjonoon’ -käyttötapaus on kuvattu näin: ”Otetaan käyttöön tekstintunnistus tikettien luokittelussa ja ohjaamisessa oikeaan työjonoon. Hyödynnetään puheen- ja tekstintunnistusta tikettien automaattisessa muodostamisessa. Ratkaisu varmistaa myös tiketin luomisen kontaktista, joka jossain tapauksessa voisi jäädä kirjaamatta. [80]” Käyttötapauksesta on yhdeksi hankinta-asiakirjaliitteeksi tehty myös oma tarkempi dokumentti, josta kopioitu prosessikaavio on esitetty kuvassa 8.1 [81]. Hankkeen aikana puheentunnistusvaatimus on kuitenkin rajattu pois hankkeen ensimmäisen vaiheen laajuudesta.



Kuva 8.1: Tikettien luokittelu ja ohjaus oikeaan työjonoon. Hankintaa varten suunniteltu prosessi. [81].

Tiketti on yhteisnimitys palvelupyynnöille ja häiriöilmoituksille. Valtori käsittelee noin 30 000 tikettiä kuukaudessa. Tiketteihin liittyviä prosesseja hallinnoidaan toiminnanohjausjärjestelmän avulla. Valtorin käyttämä toiminnanohjausjärjestelmä perustuu ServiceNow -tuotteeseen ja sitä kutsutaan TOP:iksi. Tikettityyppejä on kolme: CALL, RITM ja INC. RITM (Requested Item) tarkoittaa palvelupyyntöä ja INC (Incident) häiriöilmoitusta. CALL-tyyppinen tiketti muutetaan palvelupyynnöksi tai häiriöksi heti, kun tiedetään kummasta on kyse.

Prosessiin osallistuvat järjestelmät ovat toiminnanohjausjärjestelmä TOP sekä AINO:sta ja AION:ista koostuva tekoälyratkaisu. AINO edustaa tekoälyratkaisun perinteistä tietojärjestelmäpuolta, johon koodataan käyttötapauksen työnkulut. AION:in avulla kehitetään koneoppimismallit ja tarjotaan niitä rajapinnan yli AINO:lle. AINO:n ja AION:in arkkitehtuuria ja toimintaa on kuvattu tarkemmin luvussa seitsemän. Kuvassa 8.2 on kuvattu ’Tiketin luokittelu ja ohjaus oikeaan työjonoon’ -prosessi yksinkertaistettuna, sekä kunkin järjestelmän rooli prosessissa.



Kuva 8.2: Eri järjestelmien roolit ’Tikettien luokittelu ja ohjaus oikeaan työjonoon’ -prosessissa.

Asiakas voi tehdä palvelupyynnön tai ilmoittaa häiriöstä joko lähettämällä sähköpostia tai täyttämällä lomakkeen palveluportaalissa. Toiminnanohjausjärjestelmä muodostaa näistä automaattisesti CALL-tyyppisen tiketin. Asiakas voi myös soittaa asiakastukeen, jolloin asiakastuen asiantuntija kirjaa tiketin toiminnanohjausjärjestelmään.

Uudet tiketit muodostetaan pääsääntöisesti Asiakastuen työjonoon. Kun Asiakastuen työjonoon on saapunut uusi tiketti, toiminnanohjausjärjestelmä lähettää rajapintakutsun AINO-järjestelmälle, joka luo siitä edelleen luokittelumallin kutsun AION-järjestelmälle. Jotta AION voisi tuottaa luokitteluennusteen, luokittelumalli on täytynyt ensin kouluttaa omassa putkessaan, osana järjestelmän kehitystä ja ylläpitoa.

Tässä käyttötapauksessa AION ennustaa todennäköisimmät luokka-arvot tiketin kentille tuoteratkaisu, komponentti, toimenpide ja työjono sekä lisäksi CALL-tiketeille tiketin tyypin (RITM tai INC). AION lähettää luokitteluennusteiden tulokset varmuusarvoineen AINO:lle. AINO vertaa varmuusarvoa käyttötapaukselle asetettuihin varmuusraja-arvoihin ja muodostaa tietojen perusteella vastaussanoman toiminnanohjausjärjestelmälle. Vastaussanomassa välitetään mallin antamat ennusteet perusteluineen ja nämä tallennetaan toiminnanohjausjärjestelmän työmuistiinpanoihin.

Mikäli ennusteen varmuusarvo on ollut suurempi kuin raja-arvo, toiminnanohjasjärjestelmä täydentää sanoman perusteella tiketin tiedot ja ohjaa tiketin oikeaan työjonoon. Muussa tapauksessa tiketti jää Asiakastuen työjonoon manuaalisesti käsiteltäväksi.

8.1.5. Mallin ennustekyvykkyyteen liittyvät riskit tavoiteltavien hyötyjen näkökulmasta

Käyttötapauksen tehtävänä on siis täydentää tiketin tiedot ja ohjata tiketti oikeaan työjonoon. Käyttötapauksella saavutettava hyöty on säästetty aika, kun näitä ei tarvitse tehdä käsin. Säästöt toteutuvat vain, mikäli koulutettu koneoppimismalli ennustaa tiketin tiedot oikein ja on ennusteestaan riittävän varma (mallin laskema varmuusarvo on vähintään yhtä suuri kuin parametripalveluun tallennetun käyttötapauksen varmuusrajan arvo).

Mikäli malli on liian epävarma ennusteestaan, tiketin kenttien tietoja ei päivitetä, joten tiketti joudutaan käsittelemään manuaalisesti. Säästöt eivät siis toteudu. Vakavampi riski on, että malli on varma ennusteestaan, mutta todellisuudessa ennuste on väärä. Silloin tiketille päivitetään väärät tiedot ja se ohjataan väärään työjonoon. Tällöin tiketti joudutaan manuaalisesti korjaamaan ja palauttamaan manuaaliseen ohjaukseen. Tällöin manuaalinen työ lisääntyy lähtötilanteeseen verrattuna eli kustannukset lisääntyvät.

8.1.6. Mallin ennustekyvykkyyden evaluointikriteerit

Projektissa pyritään kehittämään mahdollisimman luotettava ennustemalli sekä analysoimaan ja esittämään sen kyvykkyyden mittarit mahdollisimman selkeästi ohjausryhmälle ja käyttötapauksen liiketoimintaomistajalle. Lopullisen arvion ennustekyvykkyyden riittävyydestä liiketoiminnan kannalta tekevät nämä tahot yhdessä.

Toiminnanohjausjärjestelmästä ei tällä hetkellä ole saatavissa kovinkaan kattavia mittareita itse liiketoimintaprosessin kyvykkyydelle. Ei esimerkiksi tiedetä, mikä osuus tiketeistä ohjataan manuaalisessa prosessissa oikeaan työjonoon. Manuaaliprosessista ei siis ole saatavilla benchmark-arvoa, johon koneoppimismallin kyvykkyyttä voisi verrata.

8.2. Datan ymmärtäminen

Koneoppimismallin opettamiseen vaadittava data saadaan toiminnanohjausjärjestelmästä. Koska tekoälyalustaa on kehitetty samassa hankkeessa yhtä aikaa käyttötapausten kanssa, on käyttötapausten kehittäminen ja datan tutkiminen ollut hyvin pitkään mahdollista ainoastaan pilvessä

olevassa kehitysympäristössä. Koska pilviympäristöön ei ole lupaa siirtää tuotannon dataa, tehtiin kehityksen aloitusvaihetta varten hyvin pieni anonymisoitu 40 näytteen koulutusdatasetti.

Ilmeisesti, koska hankkeessa oli monenmoisia teknisiä haasteita ratkottavana, ja koska varsinaista koulutusdataa ei voinut tekoälyalustalla vielä tutkia, datan analysoiminen jäi pitkään hyvin vähälle huomiolle. Käyttötapauksen määrittely ja koodaus etenivät siis hyvin pitkälle ilman syvällistä ymmärrystä datasta. Kun varsinainen koulutusdata lopulta ajettiin tuotannonohjausjärjestelmästä, huomattiin, ettei käyttötapaus sittenkään ollut ihan optimaalisesti määritelty.

Käyttötapauksessa on esimerkiksi yksi varmuusraja RITM/INC-ennusteelle ja yksi yhteinen varmuusraja kaikille muille. Yksi erittäin vaikeasti ennustettava tietokenttä voi siis aiheuttaa sen, ettei muitakaan tietoja voida täydentää automaattisesti. Koska projekti oli siinä vaiheessa jo useita kuukausia myöhässä, käyttötapaukseen ei oltu halukkaita tekemään enää muutoksia. Tämän vuoksi on erittäin suuri riski, ettei tavoiteltavia kustannushyötyjä tulla saavuttamaan.

Tämä on yksi havainnollistava esimerkki siitä, kuinka olennaisen tärkeää on sisäistää, että älykäs data-analyysi on iteratiivinen prosessi, jonka joka vaiheessa kannattaa säilyttää mahdollisuus palata edeltäviin vaiheisiin uudesta tiedosta viisastuneena.

Uusia tikettejä syntyy noin 30 000 kuukaudessa. Tiedossa oli, että työjonoihin oli tehty isompi uudistus 11.6.2019, joten koulutusaineisto koottiin tämän jälkeen syntyneistä tiketeistä, joita oli kaikkiaan noin 350 000. Varsinaiseen koulutusaineistoon valittiin vain suljettuja tikettejä, jotta kenttien arvot olisivat mahdollisimman oikeita. Kun jäljelle jätettiin ainoastaan sähköpostilla, itsepalvelun tai integraation kautta tulleet tiketit, jäi jäljelle lopulta alle 200 000 tikettiä. Uudet tiketit saapuvat pääsääntöisesti Asiakastuen työjonoon. Niistä noin 28% on myös ratkaistu Asiakastuessa. Tekoälyn tehtävänä on käsitellä ainoastaan Asiakastukeen saapuneita uusia tikettejä.

Tekoäly ennustaa todennäköisimmät luokka-arvot tiketin kentille tuoteratkaisu, komponentti, toimenpide ja työjono sekä lisäksi CALL-tiketeille tiketin tyypin (RITM tai INC). Koulutusaineistossa noin 75% tiketeistä on palvelupyyntöjä (RITM) ja 25% häiriöilmoituksia (INC). Erilaisia komponentteja on noin 50, aktiivisia työjonoja noin 130 ja toimenpiteitä hiukan alle 500. Tuoteratkaisuja on noin 20 000, mutta kaikki niistä eivät ole aktiivisia.

Tekoälyn ei haluta ohjaavan tikettejä kaikkiin työjonoihin. Tällaiset tiketit jätetään ainakin toistaiseksi Asiakastuen työjonoon manuaaliseen prosessiin. Lisäksi koulutusdatasetin muodostamisen jälkeen toiminnanohjausjärjestelmän työjonoihin on tehty muutoksia. Osa jonoista on poistettu ja lisäksi on perustettu uusia jonoja. Näiden kaikkien osalta sovittiin, että ennen kuin niistä muodostuu tuotannossa riittävästi dataa koulutuskäyttöön, myös nämä ohjataan Asiakastukeen.

Lopulta sellaisia työjonoja, joihin tekoäly saa alkuvaiheessa ohjata tikettejä jäi jäljelle noin 80. Muunnokset tehtiin niin, että opetusdatan esikäsittelyyn lisättiin muuntotaulukkokäsittely, jossa kaikki kielletyt jonot muutettiin Asiakastuen työjonoksi.

Tuoteratkaisuja on noin 20 000, joista kaikki eivät ole aktiivisia. Näiden ennustaminen oikein olisi todella haastava tehtävä. Koska toiminnanohjausjärjestelmä osaa määritellä tuoteratkaisun tuotetiedon ja organisaation perusteella, muutettiin määrittelyä niin, että koneoppimismalli ennustaakin tuotteen eikä tuoteratkaisun. Tuotteiden määrä oli noin 95, mutta liiketoiminta päätti jakaa yhden tuotteen 15 eri tuotteeseen. Tässä tapauksessa datalle ei ollut mahdollista tehdä vastaavaa jakoa. Yksikin esimerkki koulutusdatassa riittää siihen, että siitä saadaan koneoppimismallille tunnettu luokka. Ennustetarkkuus jää kuitenkin huonoksi, jos ei esimerkkejä ole riittävästi. Jatkuvien muutosten vuoksi uusinta dataa voisi olla hyvä painottaa.

Datan ymmärtämisvaiheessa huomattiin, että koulutusdatan heikko laatu aiheuttaa merkittävän riskin koneoppimismallin käytön kannattavuudelle. RITM/INC- ja työjonotietojen arvot pitäisi olla suhteellisen luotettavia, mutta tuote, komponentti- ja toimenpidetietojen laatu on todennäköisesti huonoa. Koska koneoppimismallit oppivat datan perusteella, huonolaatuisella datalla saa kehitettyä vain huonosti ennustavia malleja.

Koska vain RITM/INC- ja työjono-tiedot ovat koulutusdatassa luotettavia, muita tietoja ei ehkä kannattaisi yrittääkään ensimmäisessä vaiheessa ennustaa. Nyt on hyvin todennäköistä, että niiden luokitteluennusteen huono tarkkuus estää myös työjonotietojen päivittymisen tiketeille.

Datan laatua pyritään jatkossa parantamaan ohjeistamalla työntekijöitä tarkastamaan tiedot aina huolellisesti ennen tiketin sulkemista. Näin datan laatu toivottavasti paranee ajan kanssa. Olisi hyvä, jos ihmisten korjaamaa tietoa voisi koulutusdatassa painottaa.

8.3. Datan valmistelu

Datan valmisteluvaiheessa tehdään kaikki vaadittavat toimet, jotta raakadata saadaan muutettua koneoppimismallien hyödynnettävissä olevaan muotoon. Tässä vaiheessa dataa myös tarvittaessa korjataan.

Toiminnanohjausjärjestelmästä saatavassa raakadatassa ei ole valmiina saraketta, joka kertoisi, onko kyseessä RITM- vai INC-tiketti. Sen sijaan siinä on sarake, jossa on tiketin tunniste, jonka alkuosa on joko RITM tai INC. Opetusdataan on siis luotava uusi sarake, joka ilmaisee tiketin tyypin.

Tämän jälkeen koulutusdataan valitaan ne sarakkeet, joita vastaavat tiedot saadaan tai luovutetaan rajapinnan kautta. Samalla sarakkeet uudelleennimetään rajapintamääritysten mukaisiksi. Koska rajapinnassa luovutetaan ainoastaan ennustettujen luokkien tunnistekoodeja, tulosten analysoinnin helpottamiseksi koulutusdataan sisällytetään myös niiden selkokieliset arvot. Tässä vaiheessa koulutusdataan tehdään myös datan ymmärtämisvaiheessa esitellyt työjonojen muutokset muuntotaulukkoa hyväksikäyttäen.

Koska datassa ei ole ollut viitteitä siitä, että ulostulokolumneissa oleviin tyhjiin arvoihin liittyisi systematiikkaa, datasta poistetaan kaikki nämä rivit. Samalla poistetaan duplikaatit.

Kuten luvussa kolme ’Luonnollisen kielen käsittelyn perusteet’ opimme, tekstimuotoista dataa voidaan muuntaa numeeriseen muotoon usealla eri tavalla. Valtorin tekoälyratkaisussa käytetään sekä kirjain- että sana-n-grammeja ja näistä edelleen johdettuja sanakirja-tyyppisiä vektorisointeja. Sanaupotusten ja BERT-kielimallin hyödyntäminen tulee näillä näkymin tapahtumaan Valtorin arkkitehtuurissa osana mallinnusvaiheen AutoML-osuutta.

8.4. Mallinnus

Valtorin tekoälyratkaisussa käytetään Kerasin Tuner -kirjastoon perustuvaa AutoML-komponenttia, joka muodostaa automaattisesti esimerkiksi viisi erilaista neuroverkkoa, kouluttaa niitä jokaista jonkin aikaa ja valitsee niistä tämän perusteella parhaimman. Tämän jälkeen Trainer-komponentti kouluttaa Tunerin tallentaman parhaan mallin loppuun.

Käytännössä mallinnusvaihe etenee iteratiivisesti niin, että ensin koulutetaan AutoML:ää hyödyntäen ensimmäinen malli ja evaluoidaan sen tulokset. Tämän jälkeen pyritään tutkivasti kehittämään yhä parempi ennustemalli, kunnes evaluoinnin tulosten perusteella mallin arvioidaan olevan riittävän hyvä tuotantokäyttöön. Mallin evaluointitulosten perusteella myös ennustekyvykkyyden raja-arvo pyritään säätämään optimaaliseksi.

Kun tämän käyttötapauksen ensimmäinen ennustemalli koulutettiin oikealla koulutusdatalla QA-ympäristössä, huomattiin, ettei evaluointivaihe toimi teknisesti. Jäätiin siis ilman ensimmäisen mallin varsinaista kyvykkyysanalyysiä. Koulutettu malli kuitenkin vietiin jo tässä vaiheessa hyväksymistestattavaksi osana käyttötapausta. Testeissä huomattiin, että malli tuotti aina saman ennusteen. Näin tiedettiin, ettei mallin ennustekyvykkyys ole riittävä. Havaintojen perusteella pienivolyymiset luokka-arvot jäivät isovolyymisten jalkoihin mallin opetuksessa. Tätä haastetta pyritään jatkossa taklaamaan malliarkkitehtuurin kehittämisen lisäksi lisäämällä kustannusfunktioon recallin painotusta sekä tasapainottamalla opetusdataa.

Koska QA-ympäristössä oli teknisiä haasteita, mallin kehittämistä jatkettiin jälleen pilvipohjaisessa kehitysympäristössä. Koska siellä ei tietosuoja- ja tietoturvasyistä ole mahdollista käyttää oikeaa tuotannondataa, käytetään toistaiseksi kehittämisen tukena avointa ServiceNow-tikettidatasettiä.

Ensimmäisen mallin koulutuksessa Tuner (AutoML) osasi muodostaa vain sellaisia malleja, joissa oli peräkkäin tai haarautuen tavallisia tiheitä (dense) kerroksia, sekä dropout- ja erän normalisointi -kerroksia. Valittavissa olevaa neuroverkkomalliarkkitehtuuria monipuolistettiin lisäämällä valikoimaan myös konvoluutio- , embedding- ja LSTM-kerrokset.

BERT-kielimallin hyödyntäminen on vielä toistaiseksi toteuttamatta. Sen sijaan lisätiin mahdollisuus käyttää yhtenä neuroverkon kerrostyyppinä embedding-kerrosta, joka muodostaa koulutusaineistolle oman sanaupotusmallin. Kunhan mallin analysointia ja kehittämistä päästään tekemään oikealla koulutusdatalla, tullaan mahdollisesti kokeilemaan myös fastText-sanaupotusvektoreiden vaikutusta kyvykkyyteen.

Neuroverkkomallien ylisovittumisen ehkäisemiseksi kerroksiin lisättiin myös elastic net -tyyppinen regularisointimahdollisuus. Lisäksi otettiin käyttöön mukautuva oppimisnopeus (learning rate).

8.5. Evaluointi

Jokaisen mallin kouluttamisen jälkeen sen kyvykkyys analysoidaan. Analysoinnin perusteella pyritään tekemään malliin parannuksia, jonka jälkeen uusi malli taas koulutetaan ja analysoidaan. Tätä jatketaan, kunnes mallin arvioidaan olevan mahdollisimman hyvä. Mallien analysointia tehdään evaluointityökaluilla.

Ideaalitilanteessa käyttötapauksen liiketoimintaomistaja on osannut antaa mallin evaluoinnissa käytettäville mittareille kynnysarvot, joiden täyttyessä malli voidaan viedä automaattisesti tuotantoon. Jos tuotannossa on malli jo ennestään, uusi malli viedään tuotantoon automaattisesti vain siinä tapauksessa, että se on entistä mallia parempi. Tällainen toimintatapa vaatisi kuitenkin varsin vakiintunutta koneoppimismallien kehitys- ja ylläpitotoimintaa. Käytännössä ainakin alkuvaiheessa evaluointivaiheen tulokset esitellään liiketoimintaomistajalle ja ohjausryhmälle, jotka tekevät niiden perusteella päätöksen mallin tuotantokelpoisuudesta.

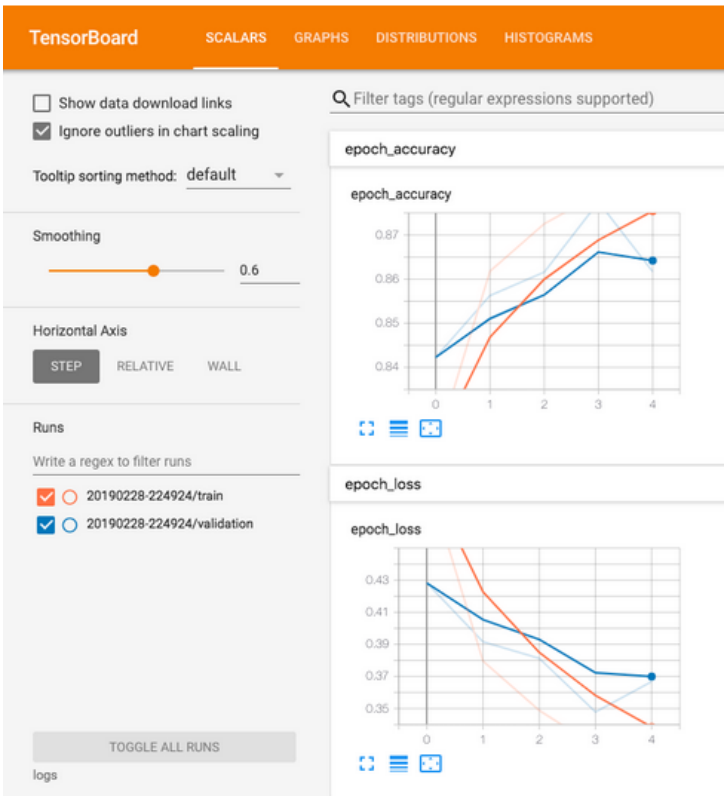
Koska mallien evaluointi ei teknisien ongelmien takia vielä toimi QA-ympäristössä, en pysty esittelemään oikealla datalla koulutettavien mallien evaluointituloksia. Sen sijaan esittelen tutoriaaleista otettujen kuvien avulla pari työkalua, joiden avulla evaluointia on tulevaisuudessa

mahdollista tehdä. Nämä työkalut ovat TensorBoard [71] ja TensorFlow Model Analysis -kirjaston [72] visualisointityökalut.

8.5.1. Mallien analysointi käyttäen TensorBoardia

Mallin kouluttamisen aikaista kyvykkyyden kehittymistä koulutus- ja validointidatalla on mahdollista seurata TensorBoardin avulla. Seurattaviksi mittariksi voi valita esimerkiksi tarkkuuden ja kustannusfunktion arvon. Kuvassa 8.3. on esimerkki tällaisesta näkymästä. Oranssit viivat esittävät mallin kehittymistä koulutusdatalla ja siniset viivat validointidatalla. Vaaleammat viivat on piirretty todellisten arvojen kuvaajina ja tummemmat viivat kuvaavat tasoitettuja arvoja. Tasoittamisen laskennassa käytettävän parametrin arvoa voi säädellä ’Smoothing’-liukukytkimen avulla.

Kuvaajista nähdään, että vaikka mallin tarkkuus koulutusdatalla paranee jatkuvasti, sen tarkkuus validointidatalla alkaa huonontua neljännen kierroksen jälkeen (indeksi 0 vastaa ensimmäistä kierrosta). Vastaavasti häviöfunktion arvo alkaa validointidatalla uudelleen kasvaa. Malli alkaa siis neljännen kierroksen jälkeen ylisovittua koulutusdataan eikä enää generalisoidu validointidataan. Mikäli mallin kouluttaminen lopetettaisiin neljänteen kierrokseen, olisi tasoitetulla kaavalla laskettu tarkkuus validointidatalla noin 0,866. Ilman tasoitusta laskettava tarkkuus olisi vähintään 0,875, mutta tätä arvoa ei kuvassa näy. Turvallisinta on evaluoida tarkkuus lisäksi täysin erillisellä evaluointidatalla. Tämä on mahdollista TensorFlow Model Analysis -kirjaston työkalujen avulla.



Kuva 8.3: Esimerkki TensorBoardin visualisointityökalun näkymästä. Kuva tutoriaalista [82]

TensorBoardin avulla on mahdollista tutkia myös sekaannusmatriiseja. Kuvassa 8.4. on esimerkki tästä. Rivit vastaavat oikeita luokkia ja sarakkeet ennustettuja luokkia. Tietyn rivin tietty arvo kuvaa siis sitä osuutta tapauksista, jotka ovat oikeasti rivin ilmoittamaa luokkaa ja on ennustettu edustavan sarakkeen ilmoittamaa luokkaa. Sekaannusmatriisin luettavuutta on parannettu värikoodauksella, jossa matriisin solun tummuusaste kuvaa kyseisen lukuarvon suuruutta. Ihannetilanteessa kaikki diagonaalin arvot olisivat ykkösiä ja siis taustaväriltään tummia, ja muut arvot nollia ja taustaväriltään vaaleita.

Sekaannusmatriisin yläpuolella olevan oranssin liukukytkimen avulla voi valita, minkä kierroksen tuloksia haluaa tarkastella. Näin on siis mahdollista tarkastella mallin kehittymistä koulutuskierrosten edetessä.

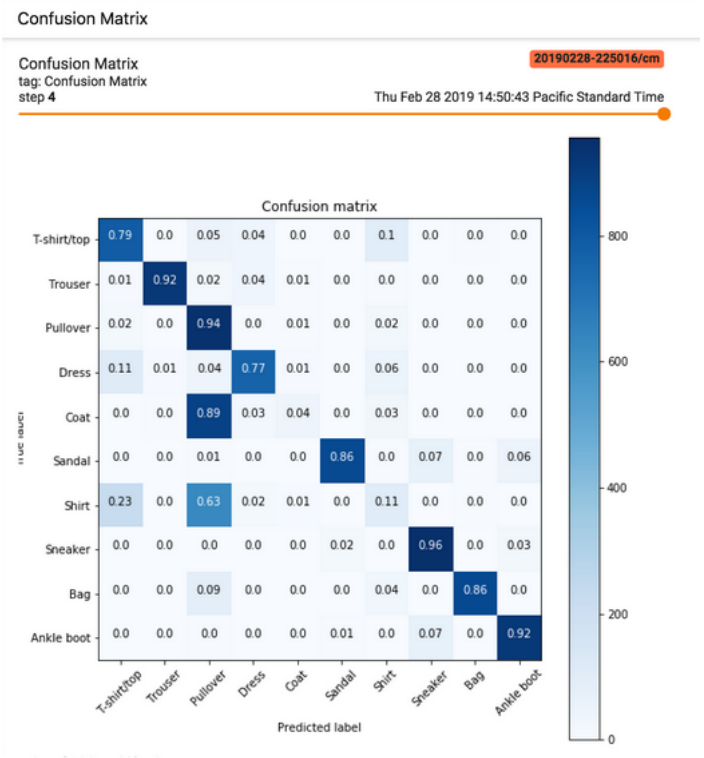
Kuvan demoesimerkin tapauksessa malli ennustaa muut luokat ihan kelvollisesti, mutta jostain syystä takkien (coat) ja paitojen (shirt) ennustetaan pääsääntöisesti olevan villapaitoja (pullover) eli malli ei osaa erottaa näitä omiksi luokikseen. Kuvitellaan, että nämä esimerkin luokat edustaisivat työjonoja. Siinä tapauksessa ennustemalli ohjaisi villapaita-työjonoon myös suurimman osan takki- ja paita-tiketeistä.

Mikäli mallia ei yrityksistä huolimatta saataisi opetettua erottamaan villapaita-, takki- ja paita-tikettejä luotettavammin toisistaan, pitäisi suunnitella, miten tämä mallin rajoite otetaan huomioon liiketoimintaprosessissa ja mallin kehityksessä.

Jos esimerkiksi villapaitatikettejä olisi 95% ja takki- ja paita tikettejä vain 5% näiden yhteenlasketusta kokonaismäärästä, ei ehkä olisi ongelmallista, vaikka villapaita-työjonoon tulisi näitä virheellisesti luokiteltuja tikettejä sekaan. Ne vain korjattaisiin manuaalisesti.

Jos suhdeluku olisi sama, mutta väärinluokittelu olisi vaikka prosessin seuraavan vaiheen automatisoinnin vuoksi todella riskialtista, yksi vaihtoehto voisi olla yhdistää takit ja paidat yhteen luokkaan ja kokeilla, oppisiko koneoppimismalli näin erottamaan ne luotettavammin villapaidoista.

Vaihtoehtoisia reagointitapoja olisi muitakin ja niiden valinta riippuisi niiden soveltamiskohteesta. Tämän vuoksi olisi tärkeää, että evaluointituloksia analysoitaisiin ja vaihtoehtoja pohdittaisiin liiketoiminnan omistajan ja asiantuntijoiden kanssa tiiviissä yhteistyössä.



Kuva 8.4: Esimerkki TensorBoardin sekaannusmatriisista. Kuva tutoriaalista [83]

TensorBoardia on mahdollista käyttää myös muuhun visualisointiin ja analysointiin. Sen avulla voi esimerkiksi tarkastella TensorFlow-neuroverkkomalleja graafeina tai vertailla eri hyperparametreilla koulutettuja malleja keskenään. Mallien vertailu käyttäen kontrolloidusti samaa validointidataa ei kuitenkaan ole mahdollista. Graafinäkymästä on helppo katsoa, minkälaisen mallin AutoML lopulta koulutti.

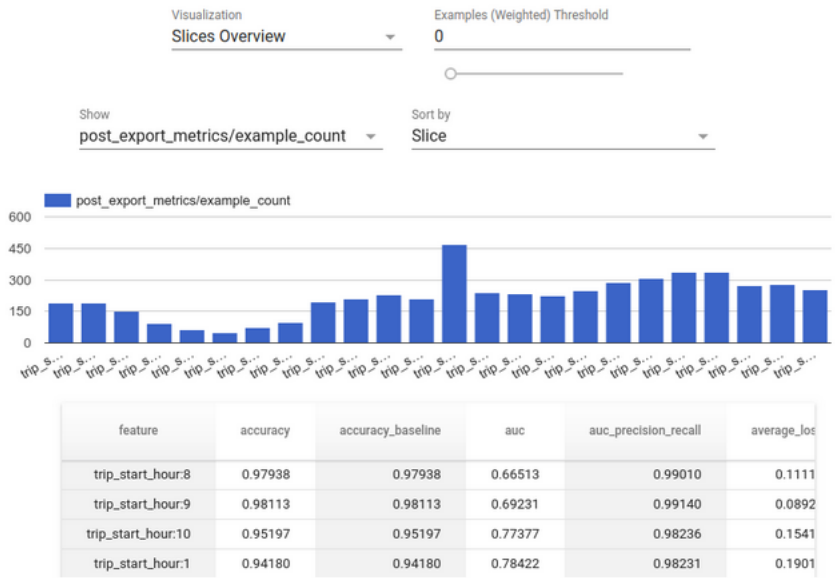
8.5.2. Mallien analysointi käyttäen TensorFlow Model Analysis -kirjaston visualisointityökalua

TensorFlow Model Analysis -kirjaston visualisointityökalu mahdollistaa mallien syvällisemmän analysoinnin erillisellä evaluointidatalla. Jotta tätä visualisointityökalua voisi käyttää, täytyy mallin koulutuksen aikana muodostaa sitä varten erityinen EvalSavedModel-malli.

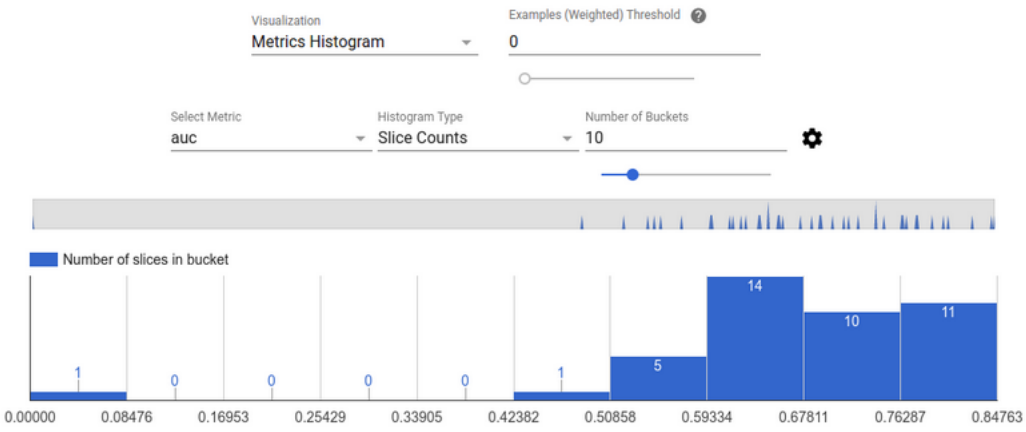
Työkalun avulla on mahdollista analysoida mallin kyvykkyyttä tarkemmalla tasolla, niin sanottuina viipaleina. Viipaleisiin jako -perusteeksi voi valita minkä tahansa datamallin piirteen tai ulostulokentän. Tikettiesimerkin tapauksessa viipaleen voisi muodostaa vaikka tietyn työjonon, viraston tai tuotteen tiketit. Hienojakoisimmillaan tarkastelua olisi mahdollista tehdä vaikka joka luokan arvolle erikseen, mutta mitä tarkemmalle tasolle analyysin haluaa viedä, sitä kauemmin evaluaation ajaminen kestää.

Työkalussa on kaksi näyttöä: viipaleiden yleiskatsaus (Slices Overview) ja metriikkahistogrammi (Metrics Histogram). Esimerkit näistä löytyvät kuvista 8.5. ja 8.6.

Viipaleiden yleiskatsaus -näkymässä valitun mittarin arvo näytetään jokaiselle viipaleelle erikseen. Mittariksi voi valita esimerkiksi tarkkuuden, näytteiden määrän, F_1-scoren, kustannusfunktion arvon tai sekaannusmatriisin. Viipaleiden visualisointia on mahdollista lajitella eri tavoin esimerkiksi nimen mukaiseen aakkosjärjestykseen tai jonkin muun mittarin mukaiseen järjestykseen. Näytön yläosan liukukytkimellä on mahdollista suodattaa näkymästä pois sellaiset luokat, joihin kuuluu kynnysarvoa vähemmän näytteitä.



Kuva 8.5: Esimerkki viipalointimetriikkatyökalusta, viipaleiden yleiskatsaus (Slices Overview) -näkymä. Kuva tutoriaalista [84]



Kuva 8.6: Esimerkki viipalointimetriikkatyökalusta, metriikkahistogrammi (Metrics Histogram) -näkymä. Kuva tutoriaalista [72]

Metriikkahistogramminäkymässä viipaleet jaetaan ryhmiin valitun mittarin arvojen perusteella. Kuvassa mittariksi on valittu AUC, joka voi saada arvoja nollan ja yhden välillä. Kussakin

histogrammin pylväässä näkyväksi arvoksi voidaan valita joko kyseiseen mittarin arvoalueeseen kuuluvien viipaleiden (Slice Counts) tai näytteiden (Example Counts) määrä.

RITM/INC-ennustemallille voidaan ajaa myös erityisiä binäärisen luokittelutehtävän kuvaajia kuten ROC-käyriä ja Precision-Recall-käyriä. Moniluokkaisista luokittelutehtävistä on mahdollista ajaa vastaavia visualisointeja muodostamalla jokaiselle luokalle ”yksi vastaan kaikki muut” -käyriä.

Ennusteen kynnysarvon valinnan vaikutusta on mahdollista analysoida kuvaajasta, jossa mallin tarkkuus, Precision, Recall ja F_1-score on piirretty suhteessa kynnysarvoon. Tällaisesta kuvaajasta on helppo nähdä, miten eri mittarit käyttäytyvät, kun kynnysarvon arvoa muutetaan. Todennäköisesti kynnysarvo kannattaa valita niin, että F_1-score maksimoituu.

8.6. Käyttöönotto

Projekti ei ole vielä edennyt käyttöönottovaiheeseen. Käyttöönottopäätöksen tulee tekemään hankkeen ohjausryhmä. Koneoppimista hyödyntävän käyttötapauksen käyttöönottopäätös tehdään samoin perustein kuin minkä tahansa IT-sovelluksen käyttöönottopäätös mutta pienellä lisällä. Sen lisäksi, että käyttötapauksen toiminnallisuus hyväksytään perinteisen hyväksymistestin tulosten perusteella, kehitetyn koneoppimismallin hyväksyntä tehdään evaluointivaiheen analyysin tulosten perusteella.

Valmiiksi koulutettu koneoppimismalli on kuin mikä tahansa ohjelmistokoodi ja se voidaan viedä tuotantoon ihan normaalein käytännöin.

8.7. Monitorointi ja ylläpito

Toiminnanohjausjärjestelmään tehdään useita kertoja vuodessa muutoksia, jotka saattavat rapauttaa mallin ennustekyvykkyyden niiltä osin. Mikäli tekoälyä halutaan jatkossa käyttää luotettavasti, tulisi näihin muutoksiin pystyä varautumaan. Tämä vaatii sekä liiketoiminnan ja koneoppimismallien kehittäjien huomattavasti läheisempää yhteistyötä että teknisesti luovia ratkaisuja, miten muutokset saadaan huomioitua malleissa.

Koska hankkeen fokus on ollut tekoälyratkaisun kehittämisessä, ratkaisun ylläpito ja monitorointi jatkuvana palveluna on jäänyt toistaiseksi hyvin vähälle huomiolle. Olisi tärkeää, että jatkossa voitaisiin verrata koneoppimismallin ennustetta toiminnanohjausjärjestelmään päätyneeseen mahdollisesti käsin korjattuun tietoon. Tällä hetkellä ennusteet jäävät vain lokitietoihin, josta niitä on vaikea hyödyntää.

9. Yhteenveto ja pohdinnat jatkokehittämisestä

Edellä olemme tutustuneet niihin koneoppimiseen liittyviin teorioihin ja käytäntöihin, joiden olen katsonut olevan oleellisia menestyksellisten koneoppimista hyödyntävien IT-projektien läpiviennissä valtionhallinnossa Valtorin tekoälyalustaa hyödyntäen.

Olemme perehtyneet älykkään data-analyysiin prosessimalliin, luonnollisen kielen käsittelyyn, koneoppimiseen ja neuroverkkoihin. Näiden teoriapainotteisten lukujen jälkeen tutustuimme konkreettisen esimerkin avulla tekstidokumenttien luokitteluun käytettävien ennustemallien kehittämiseen tutkivana prosessina. Valtorin tekoälyalustan olen esitellyt sillä tarkkuudella, mikä on julkisessa dokumentissa mahdollista. Lopuksi tutustuimme esimerkkiprojektin avulla koneoppimista hyödyntävän käyttötapauksen rakentamiseen Valtorin hallinnoiman tekoälyalustan päälle.

Valtorissa ollaan ottamassa vasta ensimmäisiä askelia tekoälyn hyödyntämisessä. Kaikilla meillä on siis paljon opittavaa. Erilaiset haasteet ovat useimmiten olennainen osa jokaista IT-projektia. Näin ollen on selvää, että myös ensimmäisen koneoppimista hyödyntävän käyttötapauksen rakentamisessa on tullut monta mutkaa matkaan. Tekoälyratkaisussa hyödynnetään uusinta teknologiaa, joka kehittyy vauhdilla. Tämän vuoksi pari vuotta sitten ei ole voitu tietää, mikä tänään on mahdollista. Osa uusista ohjelmistokirjastojen julkaisuista on tuonut mukanaan hyviä parannuksia, mutta jotkin toivotut ominaisuudet ovat yhä raakileita.

Teknisesti suurimman haasteen on aiheuttanut tekoälyratkaisun rakentaminen konesaliympäristöön. TensorFlow Extended on kehitetty pilviympäristössä ja vaikka tutoriaalien mukaan sen ajaminen Kubeflowssa olisi pitänyt olla yksinkertaista, tämä on aiheuttanut suuria hankaluuksia. Käytännössä kun pipelinen yksi kohta on saatu toimimaan, ongelmat ovat jatkuneet seuraavan vaiheen kanssa. Sanonta: ”Kun nokka irtoaa, niin pyrstö tarttuu” kuvaa erinomaisesti kehittämisprojektin etenemistä. Ongelmia on ollut ratkomassa asiantuntevat ihmiset, mutta haasteita on ollut niin valtava määrä, että hankkeen aikataulu on viivästynyt kohta vuodella.

Teknisten haasteiden suuri määrä on johtanut siihen, että muiden tärkeiden asioiden ja prosessin vaiheiden käsittely on jäänyt liian vähälle huomiolle. Esimerkiksi käyttötapaukset on määritelty ja toteutettu ennen kunnollista data-analyysiä, dokumentointi on vielä puutteellista ja jatkuvan palvelun käytännöistä on sovittu vasta ylätasolla.

Käyttötapauksia on jouduttu karsimaan, jotta tuotantoon päästäisiin edes riisutuilla versioilla. Tähän on vaikuttanut paitsi tekniset haasteet, myös lähtötilanteen osaamistaso. Hankkeen resursointi, aikataulutus ja odotustaso on ollut osin epärealistista. Ensimmäisen vaiheen karsittu ratkaisu osaa

hyödyntää vain taulukkomuotoista dataa ja neuroverkkoja. Muun muassa kuvan- ja puheentunnistusta hyödyntävien käyttötapausten kehittäminen jää seuraavien projektien tehtäväksi.

Hankkeeseen on lähdetty kyseenalaistamatta olettamaa, että koneoppimismallien avulla on mahdollista saavuttaa asetetut liiketoimintatavoitteet. Tätä olettamusta ei ole etukäteen lähdetty analysoimaan datan perusteella oikeaksi tai vääräksi. Koska evaluointivaihe ei teknisesti toimi, hankkeella ei ole vielä tässäkään vaiheessa tutkittua tietoa siitä, onko olettamus validi.

Koneoppimismallien kyvykkyys voi jäädä riittämättömäksi joko siksi, että tehtävä on yksinkertaisesti liian haastava tai siksi, että data on huonolaatuista. Koska koneoppimismallit oppivat datan perusteella, huonolaatuisella datalla saa kehitettyä vain huonolaatuisia malleja. Tämän vuoksi datan laatuun tulisi jatkossa erityisesti panostaa.

Kyseessä on siis ollut erittäin vaativa hanke. Pohdin vielä seuraavissa aliluvuissa muutamia kehittämiskohteita hieman tarkemmin.

9.1. Ympäristöt

Valtorin tekoälyratkaisussa kehitys- ja testiympäristö sijaitsevat pilvipalvelussa, ja hyväksymistesti- ja tuotantoympäristö Valtorin hallinnoimassa konesalissa virtuaalipalvelinratkaisuna. Tavallisten käyttötapausten kehitysputki etenee ympäristöissä suoraviivaisesti eli järjestyksessä kehitysympäristö, testiympäristö, hyväksymistestiympäristö ja tuotantoympäristö. Tämä sama järjestys ei kuitenkaan ole optimaalinen dataperusteisten koneoppimismallien kehittämisessä.

On erittäin tärkeää, että ennen isompien investointipäätösten tekemistä, ensin kokeillaan, onko käytettävissä olevalla datalla ja koneoppimismalleilla mahdollista saavuttaa riittävän luotettava ennustetarkkuus, että toivotut liiketoimintahyötytavoitteet ovat realistisia. Mallin kokeilu vaatii usein suuren tuotantodatamäärän hyödyntämistä. Tämän vuoksi jo mallin ensimmäisen kokeilun on yleensä tapahduttava tietoturva- ja tietosuojasysistä omassa konesalissa.

Neuroverkkojen opettaminen vaatii usein huomattavaa laskentakapasiteettia. Korkean kapasiteetin tarve on kuitenkin vain ajoittaista, joten sen hankkiminen omaan konesaliin ei todennäköisesti ole taloudellisesti kannattavaa. Isot pilvipalvelutoimittajat myös investoivat valtavasti koneoppimiseen liittyvien toiminnallisuuksien kehittämiseen omilla alustoillaan. Näitä edistyksellisiä palveluja ei aina ole mahdollista käyttää omilla palvelimilla. Tämän vuoksi olisi hyvä miettiä vaihtoehtoisia tapoja hyödyntää pilvipalveluja ja erityisesti pilvilaskentaa tietoturvallisesti.

Yksi vaihtoehto voisi olla, että ainoastaan datan esikäsittely numeeriseen muotoon tehtäisiin konesalissa. Tämän jälkeen neuroverkkojen opettamisen vaatima laskenta voisi tapahtua numeromuotoisella datalla pilvessä. Näin pilvessä käsiteltävää dataa olisi ulkopuolisten hyvin hankala tulkita.

Koneoppimismallien kehittämiseen liittyvää prosessia eri ympäristöissä ei ole vielä kuvattu. Ainakin projektivaiheessa malleja kehitetään hyväksymistestiympäristössä. Vielä on epäselvää, tapahtuuko mallien jatkuva kouluttaminen tuotantovaiheessa hyväksymistesti- vai tuotantoympäristössä. Tällä on kuitenkin ratkaiseva merkitys siihen, kuinka paljon kapasiteettia näissä ympäristöissä tarvitaan. Tällä hetkellä ympäristöt on rakennettu kapasiteetiltaan identtisiksi.

9.2. Ylläpito ja monitorointi

Valtorin ja Palkeiden yhteisessä hankkeessa on keskitytty tekoälyalustan rakentamiseen sekä viiden ensimmäisen käyttötapauksen saamiseksi tuotantoon. Koska hankkeen fokus on ollut tekoälyratkaisun kehittämisessä, ratkaisun ylläpito ja monitorointi jatkuvana palveluna on jäänyt toistaiseksi hyvin vähälle huomiolle. Perinteisten käyttötapausten ylläpito on molemmille organisaatioille osa niiden vakiintunutta toimintaa. Sen sijaan koneoppimismallien ylläpito ja monitorointi on vielä uutta.

Koneoppimismallit vaativat ihan eri tason seurantaa tuotannossa kuin perinteisesti koodatut järjestelmät. Mallien ennustekyvykkyyttä tulisi seurata jatkuvasti. Samoin tulisi määritellä prosessit, miten malleja koulutetaan jatkossa. Liiketoimintaprosesseissa pikkuhiljaa tapahtuvaan siirtymään olisi hyvä varautua säännöllisin väliajoin (tai jatkuvasti) tapahtuvalla mallien uudelleen- tai jatkokouluttamisella. Sopiva frekvenssi löytynee kokemuksen kautta. Tämän lisäksi monitoroinnissa esiin nousevan yllättävän ennustekyvykkyyden laskun tulisi käynnistää mallin tarkemman analysointiprosessin jatkotoimenpiteineen.

Liiketoimintaprosessien hallittujen muutosten huomioiminen koneoppimismallissa on ihan oma prosessinsa. Ensinnäkin pitää sopia, miten mallin ylläpidosta vastaava saa ajoissa tiedon tulevista muutoksista. Koska koneoppimismalli oppii pääsääntöisesti ainoastaan datasta, tulee olla tapa muodostaa laadukasta opetusdataa jo ennen kuin sitä syntyy muutoksen myötä tuotannossa. Kaiken kaikkiaan koko pitkä prosessi koulutusdatan hallinnoinnista ja mallien kehittämisestä, mallien evaluointiin, integraatiotestaukseen, hyväksymistestaukseen ja jatkuvaan kouluttamiseen tulee olla määritelty rooleineen, tehtävineen ja vastuineen.

9.3. Osaaminen

Kaikkein tärkeimpänä menestystekijänä tulevien koneoppimismalleja hyödyntävien projektien kannalta näen osaamisen kehittämisen. Syvemmälle osaamiselle olisi tarvetta joka vaiheessa, kuten esimerkiksi hankkeen rahoittamisesta päätettäessä, hankkeen suunnittelussa ja ohjaamisessa, hankinnan vaatimusten määrittelyssä, toimittajan ohjaamisessa, ketterässä kehittämisessä sekä jatkuvan palvelun suunnittelussa.

Ehkä kaikkein tärkeintä olisi antaa jokaiselle projektiin, sen ohjausryhmään ja sidosryhmiin kuuluvalle perehdytys älykkään data-analyysin iteratiivisesta prosessista. Muun muassa AutoML:n ympärillä käyvä hypetys on voinut saada aikaan mielikuvia, että prosessissa olisi kyse vain teknisten työkalujen sokeasta käytöstä. Käytännössä asiantuntijoiden merkitys prosessissa on aivan olennainen. Näissäkin projekteissa kehittämistä tulisi siis tehdä ketterästi, teknisten ja liiketoiminnan asiantuntijoiden kiinteässä yhteistyössä.

Erityisesti päättäjillä tulisi olla selkeä ymmärrys siitä, että ensin tulee tutkia aidolla datalla, onko liiketoiminnan tavoitteet mahdollista saavuttaa ja vasta tämän analyysin perusteella tehdä lopullinen investointipäätös varsinaisesta toteutusprojektista. Aitoon dataan perustuvan älykkään data-analyysin tuloksena projektin tavoitteita ja kustannus-hyöty-analyysiä on mahdollista tarkistaa perustuen tutkittuun tietoon.

9.4. Ehdotus Valtionvarainministeriön tekoälyä hyödyntäviä hankkeita koskevien erityisrahoitushakujen kehittämiseksi

Tällä hetkellä on menossa Valtiovarainministeriön erityisrahoitushaku nousevia teknologioita hyödyntäviin hankkeisiin, jonka tavoitteena on avustaa virastoja tehostamaan ja automatisoimaan prosessejaan. Rahaa on jaossa lähes kuusi miljoonaa. ”Rahoituksen myöntämisessä arvioidaan seuraavia vaatimuksia, joiden kaikkien täyttyminen on rahoituksen saannin edellytys:

- Hankkeessa hyödynnetään robotiikkaa, tekoälyä, data-analytiikka tai muita nousevia teknologioita.
- Hanke on toimeenpantavissa heti, kun rahoitus on myönnetty.
- Investoinnin takaisinmaksuaika on enintään kolme vuotta rahoituksen myöntämisestä.
- Tavoitteena olevan taloudellisen hyödyn on oltava vähintään kaksinkertainen investointiin verrattuna kuuden vuoden laskentajaksolla rahoituksen myöntämisestä.” [85]

Koska koneoppimismalleja hyödyntävien projektien takaisinmaksuaika riippuu täysin koneoppimismallien kyvykkyydestä, realistisen takaisinmaksuajan laskeminen on mahdollista vasta älykkään data-analyysin evaluointitulosten perusteella. Kaikki sitä ennen tehdyt laskelmat ovat enemmän tai vähemmän toiveita. Mikäli projektit olisivat lähdössä liikkeelle tyhjästä, niin itse myöntäisin rahoitusta annettujen vaatimusten perusteella ainoastaan ohjelmistorobotiikkaa hyödyntäviin hankkeisiin, koska niihin liittyvä teknologia, prosessit ja kokemuksen tuoma osaaminen on huomattavasti kypsemmällä tasolla.

Ehdottaisinkin, että tekoälyä hyödyntäviä hankkeita koskevat rahoitushaut järjestettäisiin jatkossa kaksivaiheisina. Ensimmäisessä vaiheessa myönnettäisiin rahoitus ainoastaan älykkään data-analyysin tekemiseen. Myös ensimmäisen vaiheen hakua varten edellytettäisiin koko projektin ja jatkuvan palvelun ensimmäisiä vuosia koskeva kustannus-hyöty-analyysi, mutta tässä vaiheessa tunnistettaisiin avoimesti arvioihin liittyvät epävarmuustekijät. Rahoitushaun toisessa vaiheessa tehtäisiin päätös hankkeen jatkorahoituksesta älykkään data-analyysin tuloksena tarkennettujen tavoitteiden ja kustannus-hyöty-analyysin perusteella.

Näkisin, että Valtorin hallinnoima tekoälyalusta valmiiksi kilpailutettuine asiantuntijapalveluineen mahdollistaisi valmistuttuaan näiden rahoitushakujen ensimmäisen vaiheen älykkäiden data-analyysien toteuttamisen valtionhallinnossa ketterästi, kustannustehokkaasti ja tietoturvallisesti.

Lähdeluettelo

[1] Tietoa Valtorista. https://valtori.fi/tietoa-valtorista. [Viitattu 6.7.2020]

[2] Tietopyyntö – Tekoälyn hyödyntäminen asiakaspalvelun parantamiseksi Valtorissa ja Palkeissa. 10.9.2018. https://valtori.fi/-/tietopyynto-tekoalyn-hyodyntaminen-asiakaspalvelun-parantamiseksi-valtorissa-ja-palkeissa. [Viitattu 6.7.2020]

[3] The elements of AI-verkkokurssi. https://www.elementsofai.com/fi [Viitattu 14.7.2020]

[4] Michael R. Berthold, Christian Borgelt, Frank Höppner, Frank Klawonn: Guide to Intelligent Data Analysis: How to Intelligently Make Sense of Real Data. Springer, London (2010). s. 6-12

[5] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, Rüdiger Wirth: Cross Industry Standard Process for Data Mining 1.0, Step-by-step Data Mining Guide. CRISP-DM consortium (2000) *CRISP-DM 1.0 Step-by-step data mining guides*.

[6] Kai Hakala, Jenna Kanerva: Johdatus kieliteknologiaan -kurssin luentomuistiinpanot, Turun yliopisto (2019)

[7] Marita Risku: Erikoistyö - Tekstidokumenttien luokittelu. https://github.com/maritari/text-classification/blob/master/Tekstidokumenttien%20luokittelu.ipynb. [Viitattu 9.7.2020]

[8] Turku NLP group: Parser demo. http://bionlp-www.utu.fi/parser_demo [Viitattu 14.7.2020]

[9] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean: Efficient Estimation of Word Representations in Vector Space. (2013) Workshop Proceedings of the International Conference on Learning Representations (ICLR) 2013.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2019) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)

[11] fastText - Library for efficient text classification and representation learning. https://fasttext.cc/. [Viitattu 9.7.2020]

[12] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov: Enriching Word Vectors with Subword Information. (2017) Transactions of the Association for Computational Linguistics, Volume 5

[13] Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov: Bag of Tricks for Efficient Text Classification. (2017) Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers

[14] Jeffrey Pennington, Richard Socher, Christopher D. Manning: GloVe: Global Vectors for Word Representation (2014) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)

[15] Dan Jurafsky, James H. Martin: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (3rd ed. draft). Stanford University (2019) https://web.stanford.edu/~jurafsky/slp3/. [viitattu 10.7.2020]

[16] Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, Adam Kalai: Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. (2016) NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems

[17] BERT. https://github.com/google-research/bert [Viitattu 13.7.2020]

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention Is All You Need. (2017) Advances in Neural Information Processing Systems 30 (NIPS 2017)

[19] Jay Alammar: Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention). https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/ [Viitattu 13.7.2020]

[20] Jay Alammar: The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). http://jalammar.github.io/illustrated-bert/ [Viitattu 13.7.2020]

[21] BERT/multilingual. https://github.com/google-research/bert/blob/master/multilingual.md [Viitattu 14.7,2020]

[22] Telmo Pires, Eva Schlinger, Dan Garrette: How multilingual is Multilingual BERT? (2019) Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics

[23] Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, Sampo Pyysalo: Multilingual is not enough: BERT for Finnish. arXiv:1912.07076v1 (2019)

[24] FinBERT. BERT model trained from scratch on Finnish. http://turkunlp.org/FinBERT/ [Viitattu 13.7.2020]

[25] Kielipankki. https://www.kielipankki.fi/. [Viitattu 9.7.2020]

[26] Lahjoita puhetta -kampanjasivu. https://yle.fi/aihe/lahjoita-puhetta. [Viitattu 9.7.2020]

[27] Lahjoita puhetta -palvelu. https://lahjoitapuhetta.fi/. [Viitattu 9.7.2020]

[28] Jouni Luoma, Miika Oinonen, Maria Pyykönen, Veronika Laippala, Sampo Pyysalo: A Broad-coverage Corpus for Finnish Named Entity Recognition. (2020) Proceedings of The 12th Language Resources and Evaluation Conference

[29] Turku NLP group: Turku neural parser pipeline. https://turkunlp.org/Turku-neural-parser-pipeline/ [Viitattu 14.7.2020]

[30] Turku NLP group: Word embedding demo. http://bionlp-www.utu.fi/wv_demo/ [Viitattu 14.7.2020]

[31] Annif. https://annif.org/ [Viitattu 16.7.2020]

[32] FISKMÖ-projekti. https://blogs.helsinki.fi/fiskmo-project/?lang=fi. [Viitattu 14.7.2020]

[33] Stephen Marsland: Machine Learning: An Algorithmic Perspective, CRC Press (2009), s 6-7, 51

[34] Jukka Heikkonen: Machine Learning and Pattern recognition -kurssin luentomuistiinpanot, Turun yliopisto (2019)

[35] Antti Airola: Data Analysis and Knowledge Discovery -kurssin luentomuistiinpanot, Turun yliopisto (2018)

[36] Corinna Cortes, Vladimir Vapnik: Support-Vector Networks. Machine learning 20.3 (1995), s. 273-297.

[37] Christopher J.C. Burges: A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery 2 (1998), s. 121-167

[38] Teuvo Kohonen: The self-organizing map. Proceedings of the IEEE 78.9 (1990). s. 1464-1480.

[39] Arthur E. Hoerl: Application of ridge analysis to regression problems. Chemical Engineering Progress 58, (1962), s. 54-59.

[40] Robert Tibshirani: Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society: Series B (Methodological) 58.1 (1996), s. 267-288

[41] Hui Zou and Trevor Hastie: Regularization and Variable Selection via the Elastic Net. Journal of the Royal Statistical Society. Series B (Statistical Methodology) 67.2 (2005), s. 301-320

[42] Leo Breiman: Bagging Predictors. Machine Learning 24 (1996), s. 123-140

[43] Robert E. Schapire: The Boosting Approach to Machine Learning: An Overview. Nonlinear Estimation and Classification, Springer (2003).

[44] Yann LeCun, Yoshua Bengio, Geoffrey Hinton: Deep learning. Nature 521(7553), (2015), s. 436-444

[45] Ian Goodfellow, Yoshua Bengio, Aaron Courville: Deep Learning. MIT Press (2016) http://www.deeplearningbook.org/

[46] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks. Proceedings of NIPS (2012).

[47] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton. Speechrecognition with deep recurrent neural networks. Proceedings of ICASSP (2013)

[48] Timo Knuutila: Deep learning -kurssin luentomuistiinpanot, Turun yliopisto (2019)

[49] François Chollet: Deep Learning with Python. Manning Publications (2018).

[50] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, Yann LeCun: What is the best multi-stage architecture for object recognition? 2009 IEEE 12th International Conference on Computer Vision (2009), s. 2146–2153.

[51] Vinod Nair, Geoffrey E. Hinton: Rectified Linear Units Improve Restricted Boltzmann Machines. Proceedings of the 27th International Conference on International Conference on Machine Learning (2010), s. 807–814

[52] Xavier Glorot, Antoine Bordes, Yoshua Bengio: Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, volume 15 (2011), s. 315-323

[53] Moshe Leshno, Vladimir Ya.Lin, AllanPinkus, ShimonSchocken: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks Volume 6, Issue 6 (1993), s.861-867

[54] Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner: Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11) (1998) s. 2278–2324

[55] Yoon Kim: Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2014)

[56] Sepp Hochreiter, Jürgen Schmidhuber: Long Short-term Memory. Neural Computation 9(8) (1997), s. 1735-1780

[57] Felix A. Gers, Jürgen Schmidhuber, Fred Cummins: Learning to Forget: Continual Prediction with LSTM. Neural Computation. 12 (10) (2000), s. 2451–2471.

[58] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. The Journal of Machine Learning Research 15(56) (2014), s. 1929−1958

[59] Sergey Ioffe, Christian Szegedy: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning, PMLR 37 (2015) s. 448-456

[60] Augustin Cauchy: Méthode générale pour la résolution des systèmes d'équations simultanées. C. R. Acad. Sci. Paris (1847) s. 536-538

[61] Sinno Jialin Pan, Qiang Yang: A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering. Volume: 22, Issue: 10, (2010)

[62] The Jupyter Notebook. https://jupyter-notebook.readthedocs.io/en/stable/ [Viitattu 23.7.2020]

[63] Scikit-learn. https://scikit-learn.org/stable/ [Viitattu 23.7.2020]

[64] TensorFlow Extended (TFX) is an end-to-end platform for deploying production ML pipelines. https://www.tensorflow.org/tfx [Viitattu 21.7.2020]

[65] TensorFlow Extended (TFX): Real World Machine Learning in Production. https://blog.tensorflow.org/2019/06/tensorflow-extended-tfx-real-world_26.html [Viitattu 21.7.2020]

[66] Kubeflow. https://www.kubeflow.org/ [Viitattu 23.7.2020]

[67] TensorFlow. https://www.tensorflow.org/ [Viitattu 21.7.2020]

[68] Keras. https://keras.io/ [Viitattu 21.7.2020]

[69] ML Metadata. https://www.tensorflow.org/tfx/guide/mlmd [Viitattu 23.7.2020]

[70] Keras Tuner documentation. https://keras-team.github.io/keras-tuner/ [Viitattu 5.10.2020]

[71] TensorBoard: TensorFlow's visualization toolkit. https://www.tensorflow.org/tensorboard [Viitattu 21.9.2020]

[72] Improving Model Quality With TensorFlow Model Analysis. https://www.tensorflow.org/tfx/guide/tfma [Viitattu 21.9.2020]

[73] Tensorflow Model Analysis Metrics. https://www.tensorflow.org/tfx/model_analysis/metrics [Viitattu 23.7.2020]

[74] TensorFlow Model Analysis. https://www.tensorflow.org/tfx/tutorials/model_analysis/tfma_basic [Viitattu 23.7.2020]

[75] TensorFlow Data Validation: Checking and analyzing your data. https://www.tensorflow.org/tfx/guide/tfdv [Viitattu 23.7.2020]

[76] H2O. https://www.h2o.ai/ [Viitattu 23.7.2020]

[77] OpenCV. https://opencv.org/ [Viitattu 23.7.2020]

[78] Scikit-image. https://scikit-image.org/ [Viitattu 23.7.2020]

[79] Tesseract OCR. https://tesseract-ocr.github.io/ [Viitattu 23.7.2020]

[80] Valtori ja Palkeet: Tarjouspyyntö 655/02.05/2018: Tekoälyratkaisu ja tekoälytoteutukset sekä niihin liittyvät palvelut. Liite 1 Hankinnan kohteen kuvaus 27.3.2019.

[81] Valtori ja Palkeet: Tarjouspyyntö 655/02.05/2018: Tekoälyratkaisu ja tekoälytoteutukset sekä niihin liittyvät palvelut. Liite 1.8.1 KTV1: Käyttötapaukset Valtorin prosessissa ’tikettien luokittelu ja ohjaaminen oikeaan työjonoon TOP:ssa’

[82] Using TensorBoard in Notebooks. https://www.tensorflow.org/tensorboard/tensorboard_in_notebooks [Viitattu 21.9.2020]

[83] Displaying image data in TensorBoard. https://www.tensorflow.org/tensorboard/image_summaries [Viitattu 21.9.2020]

[84] Getting Started with TensorFlow Model Analysis. https://www.tensorflow.org/tfx/model_analysis/get_started [Viitattu 21.9.2020]

[85] Erityisrahoitushaku nousevia teknologioita hyödyntäviin hankkeisiin 30.10.2020 saakka. https://vm.fi/robohaku2020 [Viitattu 24.9.2020]

Tekstidokumenttien luokittelu

Tässä työkirjassa esitellään luonnollisen kielen prosessointiin (NLP) perustuvaa tekstidokumenttien luokittelua. Olen kirjoittanut työkirjan perehdytysmateriaaliksi työyhteisöni jäsenille, jotka ovat kiinnostuneita koneoppimisesta ja luonnollisen kielen käsittelystä, mutta joilla ei ole niistä aiempaa kokemusta. Tavoitteenani on auttaa heitä saavuttamaan perustason ymmärrys, mistä asiassa on kyse ja samalla mahdollisesti kannustaa kokeilemaan myös itse. Työkirjaa voi halutessaan huoletta lukea hyppäämällä koodiosuuksien yli. Koodin ymmärtäminen ei ole välttämätöntä tekstidokumenttien luokittelun perusteiden ymmärtämisen kannalta.

Luonnollisella kielellä tarkoitetaan kieltä, jonka alkuperää ei tiedetä (esim. suomi, ruotsi, englanti). Vastaavasti esimerkiksi ohjelmointikielet ovat keinotekoisia kieliä.

Luokittelussa käytetään ohjattua koneoppimista. Ohjatussa oppimisessa tarvitaan aina mahdollisimman hyvin todellisuutta vastaava, riittävän suuri koulutusaineisto, jossa jokaiselle koulutusaineiston esimerkille on talletettu myös sen oikean luokan arvo. Koulutuksen aikana koneoppimisalgoritmi optimoi ennustemallin sovittamalla sen koulutusaineistoon. Tavoitteena on, että tuloksena saatava malli osaisi mahdollisimman luotettavasti ennustaa oikean luokan sellaisille dokumenteille, joita se ei ole opetusvaiheessa nähnyt.

Työkirjan alkuosassa hyödynnetään tavallisella kannettavalla Windows-koneella (ei grafiikkasuoritinta, GPU) helposti käyttöönotettavia luonnollisen kielen prosessointiin kehitettyjä Python-koneoppimiskirjastoja. Nämä kirjastot on helppo ladata omalle tietokoneelle esimerkiksi osana open-source Anaconda-jakelua (https://www.anaconda.com/distribution/ (https://www.anaconda.com/distribution/)).

Myös yksinkertaisia neuroverkkoja on mahdollista opettaa tavallisella kotikoneella. Suurten datamäärien ja syvempien neuroverkkojen kouluttaminen vaatii kuitenkin muistia ja laskentakapasiteettia peruskotikonetta enemmän. Työkirjan lopussa hyödynnetäänkin neuroverkkojen kouluttamiseen CSC:n yliopistoille tarjoamaa Puhti-supertietokonetta (https://www.csc.fi/-/supertietokone-puhti-on-avattu-tutkijoiden-kayttoon (https://www.csc.fi/-/supertietokone-puhti-on-avattu-tutkijoiden-kayttoon)).

Työkirja on tehty käyttäen interaktiivista Jupyter Notebookia (https://jupyter.org/ (https://jupyter.org/)), joka on selaimella käytettävä ohjelmointityökalu. Myös Jupyter Notebook asentuu automaattisesti osana Anaconda-jakelua. Työkirjaa voi joko lukea valmiina dokumenttina tai sen koodisoluja voi (muokata ja) ajaa itse, jolloin uudet ajotulokset korvaavat vanhat.

1. Käytetyt kirjastot

Kaikki työkirjassa käytetyt Python-kirjastot on kerätty tähän työkirjan alkuun, jotta ne olisi kätevästi ladattavissa keskusmuistiin, heti kun työkirja avataan.

Tällä hetkellä Pythonin uusin versio on 3.8.0, joka on julkaistu 14.10.2019 (https://www.python.org/downloads/release/python-380/ (https://www.python.org/downloads/release/python-380/)). Kuitenkin monipuolinen luonnollisen kielen käsittelyyn kehitetty NLTK-kirjasto toimii korkeintaan Pythonin versiolla 3.7 (https://www.nltk.org/install.html (https://www.nltk.org/install.html)) ja neuroverkkojen opettamiseen kehitetty Keras on yhteensopiva vain Pythonin versioiden 2.7-3.6 kanssa (https://keras.io/ (https://keras.io/)). Siksi käytän työkirjan alkuosassa Pythonin versiota 3.7 ja loppuosassa versiota 3.6. Käytännössä minulla on siis Anacondan avulla asennettuna kaksi ympäristöä, joista valitsen sopivan jo ennen työkirjan (Jupyter Notebook) avaamista. Kun haluan vaihtaa toiseen ympäristöön, suljen työkirjan, valitsen Anacondassa uuden ympäristön ja avaan siitä ympäristöstä työkirjan uudelleen. Tämän jälkeen ajan työkirjan alusta vain siinä ympäristössä käyttämäni kirjastot.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

In [2]:
```
# käytän Python 3.7.0 kanssa
import nltk
# nltk.download('punkt') täytyy ladata vain kertaalleen
```

In [3]:
```
# käytän Python 3.6.8 kanssa, koska yhteentoimivuutta ei luvata uudempien versioide
n kanssa.
from keras import models
from keras import layers
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from keras import optimizers
from gensim.models import KeyedVectors
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
#import gensim
```

```
Using TensorFlow backend.
```

Jotta en pidempien taukojen jälkeen joutuisi aloittamaan aina koko työkirjan ajamista alusta, olen lisännyt tähän alkuun myös tärkeiden myöhemmässä vaiheessa laskettavien vektoreiden lataamisen muistista.

In [12]:
```
X_train = np.load('X_train.npy')
y_train = np.load('y_train.npy')
X_dev = np.load('X_dev.npy')
y_dev = np.load('y_dev.npy')
X_test = np.load('X_test.npy')
y_test = np.load('y_test.npy')

labels_train = np.load('labels_train.npy')
labels_dev = np.load('labels_dev.npy')
labels_test = np.load('labels_test.npy')

Xpad_train = np.load('Xpad_train.npy')
Xpad_dev = np.load('Xpad_dev.npy')
Xpad_test = np.load('Xpad_test.npy')

embmatrix = np.load('embmatrix.npy')
embmatrix_all = np.load('embmatrix_all.npy')
```

2. Luokitteluun käytettävä data

Demodatana käytetään Kielipankin CC-BY-NC 4.0 lisenssillä jakamaa Ylilauta-korpusta (http://urn.fi/urn:nbn:fi:lb-2016101210 (http://urn.fi/urn:nbn:fi:lb-2016101210)), jota Turun yliopiston kieli- ja puheteknologian apulaisprofessori Sampo Pyysalo on edelleen muokannut yksinkertaiseen (aihe, teksti) TSV-muotoon ja tuottanut siitä menetelmien vertailuun tasapainotetun kymmenen luokan datasetin. Käytetty aineisto on saatavilla täältä: https://github.com/spyysalo/ylilauta-corpus (https://github.com/spyysalo/ylilauta-corpus).

Ylilauta on anonyymi keskustelufoorumi, joka ei vaadi rekisteröitymistä tai nimimerkin käyttöä. Foorumi löytyy osoitteesta https://ylilauta.org/ (https://ylilauta.org/). Ylilauta-korpukseen on tallennettu kyseisen keskustelufoorumin keskustelupalstoja vuosilta 2014-2016. Tässä työssä käytettävään suppeampaan aineistoon on otettu sama määrä viestejä kymmenestä yleisimmin esiintyvästä aiheluokasta. Nämä aiheluokat ovat:

1. Ajoneuvot
2. Hikky (Hikikomero, masentuneiden ja sosiaalisesti syrjäytyneiden vertaistukiryhmä)
3. Kuntosali
4. Muoti
5. Pelit
6. Penkkiurheilu
7. Politiikka
8. Seksuaalisuus
9. Sota
10. Televisio

Aineisto on jaettu valmiiksi kolmeen osaan:

- Koulutusdata erilaisten mallien kouluttamiseen: ylilauta-train.txt (10 000 kpl/aihe, yht 100.000 kpl)
- Validointidata mallien parametrien optimoimiseen ja keskinäiseen vertailuun, jotta voidaan valita näistä paras: ylilauta-dev.txt (1 000 kpl/ aihe, yhteensä 10 000 kpl)
- Testidata, jota käytetään parhaimmaksi valitun mallin luotettavuuden arviointiin: ylilauta-test.txt (1 000 kpl/ aihe, yhteensä 10 000 kpl)

Koska käytämme validointiaineistoa hyperparametrien optimoimiseen ja parhaan mallin valitsemiseen, on syytä testata parhaan mallin luotettavuus aina täysin erillisellä testidatalla.

Luetaan ensin kaikki kolme aineistoa datakehikoihin (Panda's dataframe: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html)). Data tallentuu alkuperäisessä muodossaan sarakkeeseen 'original'. Tämän jälkeen erotetaan alkuperäisestä tekstistä erikseen aiheet (sarakkeeseen 'label') ja varsinaiset dokumenttien sisältötekstit (sarake 'text'). Varmistetaan myös viestien lukumäärät aiheittain.

2.1 Koulutusdata

In [4]:
```
# luetaan data alkuperäisessä muodossaan ensimmäiseen sarakkeeseen (sarake numero
0):
df_train = pd.read_csv('ylilauta-train.txt', sep='\n', header = None)
# erotetaan aihe eli luokka toiseen sarakkeeseen (sarake numero 1):
df_train[1] = df_train[0].apply(lambda x: x.split()[0].split('_')[4].strip())
# erotetaan sisältöteksti kolmanteen sarakkeeseen (sarake numero 2):
df_train[2] = df_train[0].str.split(n=1).str[1]
# annetaan sarakkeille nimet:
df_train.columns = ['original','label','text']
# tulostetaan 10 ensimmäistä riviä:
print(df_train.head(10))
# tulostetaan datakehikon muoto (100000 riviä, 3 saraketta):
df_train.shape
```

```
                                            original       label  \
0  __label__ajoneuvot Kattokaas bemarijonnet tää ...   ajoneuvot
1  __label__pelit http://www.gamespot.com/article...       pelit
2  __label__sota Miehittäjiä vastaan taistelevien...        sota
3  __label__ajoneuvot Sen jälkeen kun ST1 osti Sh...   ajoneuvot
4  __label__muoti Toisiksi alin vasemmalta näyttä...       muoti
5  __label__politiikka Väität , että perinteisest...  politiikka
6  __label__sota varmaa toi heppatyttö kersantti ...        sota
7  __label__kuntosali Tässä langassa kuvailemme t...   kuntosali
8  __label__politiikka Lisään vain , että mitään ...  politiikka
9  __label__sota En tiedä miten edes olisi mahdol...        sota

                                                text
0  Kattokaas bemarijonnet tää , ei taida teidän 3...
1  http://www.gamespot.com/articles/e3-2014-why-x...
2  Miehittäjiä vastaan taistelevien tahojen tavoi...
3  Sen jälkeen kun ST1 osti Shellin Suomen ja Ruo...
4  Toisiksi alin vasemmalta näyttää hyvältä . Ei ...
5  Väität , että perinteisestä vasemmistosta on v...
6  varmaa toi heppatyttö kersantti k ja kapteeni ...
7  Tässä langassa kuvailemme treenaamista haikuil...
8  Lisään vain , että mitään tilastoja ilokaasuke...
9  En tiedä miten edes olisi mahdollista pistää e...
```

Out[4]:
```
(100000, 3)
```

In [5]:
```
df_cat = df_train.astype('category')
df_cat['label']
print('Viestien lukumäärät aiheittain:')
print(df_train["label"].value_counts())
```

```
Viestien lukumäärät aiheittain:
seksuaalisuus    10000
ajoneuvot        10000
hikky            10000
pelit            10000
politiikka       10000
televisio        10000
sota             10000
penkkiurheilu    10000
kuntosali        10000
muoti            10000
Name: label, dtype: int64
```

2.2 Validointidata

In [6]:
```
df_dev = pd.read_csv('ylilauta-dev.txt', sep='\n', header = None)
df_dev[1] = df_dev[0].apply(lambda x: x.split()[0].split('_')[4].strip())
df_dev[2] = df_dev[0].str.split(n=1).str[1]
df_dev.columns = ['original','label','text']
print(df_dev.head(10))
df_dev.shape
```

```
                                            original          label  \
0  __label__ajoneuvot Polttoöljyn tankkaaminen ja...      ajoneuvot
1  __label__kuntosali Voi mahoton ! Täällähän on ...      kuntosali
2  __label__penkkiurheilu Mutta tää peli on kyllä...  penkkiurheilu
3  __label__kuntosali Ei tosta kyllä olympia-taso...      kuntosali
4  __label__sota Jep , tota ei meenaa pikkasenkaa...           sota
5  __label__pelit 3-4 päivää oli jonotusta ja lag...          pelit
6  __label__muoti Tuo fraasi on jo yleinen vitsi ...          muoti
7  __label__politiikka >Oletko sinä erityisen arv...     politiikka
8  __label__televisio Seuraako muut nyymit tätä s...      televisio
9  __label__sota Jahas , niin kai sitten . Onko k...           sota

                                                text
0  Polttoöljyn tankkaaminen ja jopa sillä ajamine...
1  Voi mahoton ! Täällähän on jälleen vanha kunno...
2  Mutta tää peli on kyllä väsynyt paska , eilen ...
3  Ei tosta kyllä olympia-tasoon ole , vaikka hyv...
4  Jep , tota ei meenaa pikkasenkaan pulskempi ka...
5  3-4 päivää oli jonotusta ja lagia , sitten lop...
6  Tuo fraasi on jo yleinen vitsi vuonna 2014. ht...
7  >Oletko sinä erityisen arvokas kenellekään muu...
8  Seuraako muut nyymit tätä sarjaa ? Hyvää viihd...
9  Jahas , niin kai sitten . Onko kuninkaallisess...
```

Out[6]:
```
(10000, 3)
```

In [7]:
```
df_cat = df_dev.astype('category')
df_cat['label']
print('Viestien lukumäärät aiheittain:')
print(df_dev["label"].value_counts())
```

```
Viestien lukumäärät aiheittain:
kuntosali        1000
politiikka       1000
hikky            1000
penkkiurheilu    1000
televisio        1000
sota             1000
muoti            1000
seksuaalisuus    1000
ajoneuvot        1000
pelit            1000
Name: label, dtype: int64
```

2.3 Testidata

In [8]:
```
df_test = pd.read_csv('ylilauta-test.txt', sep='\n', header = None)
df_test[1] = df_test[0].apply(lambda x: x.split()[0].split('_')[4].strip())
df_test[2] = df_test[0].str.split(n=1).str[1]
df_test.columns = ['original','label','text']
print(df_test.head(10))
df_test.shape
```

```
                                            original          label  \
0  __label__politiikka Kyllä kai tyypillinen kapi...     politiikka
1  __label__sota En suosittele sinne menemistä as...           sota
2  __label__hikky Tiedän yhden jampan joka on mui...          hikky
3  __label__penkkiurheilu No vaa kuka molaroi gam...  penkkiurheilu
4  __label__sota Tässä pitää tietää , että nuo lä...           sota
5  __label__penkkiurheilu Voidaan siirtyä neutraa...  penkkiurheilu
6  __label__seksuaalisuus >Exällä oli liian iso m...  seksuaalisuus
7  __label__politiikka >ok puukottaa toi tehtaanj...     politiikka
8  __label__politiikka Juutalaistyylinen tapa jyr...     politiikka
9  __label__muoti luultavasti babby ' s first haj...          muoti

                                                text
0  Kyllä kai tyypillinen kapitalisti ( yrittäjä )...
1  En suosittele sinne menemistä asenteella mitäh...
2  Tiedän yhden jampan joka on muinoin käyttänyt ...
3  No vaa kuka molaroi game sevenin ? Se nimittäi...
4  Tässä pitää tietää , että nuo lähetit valitaan...
5  Voidaan siirtyä neutraaliin oc-lankaan jos tää...
6  >Exällä oli liian iso muna , oisikohan se vaik...
7  >ok puukottaa toi tehtaanjohtaja koska se on t...
8  Juutalaistyylinen tapa jyrätä toiinajattelu al...
9  luultavasti babby ' s first hajuvesi eikä viel...
```

Out[8]: (10000, 3)

In [9]:
```
df_cat = df_test.astype('category')
df_cat['label']
print('Viestien lukumäärät aiheittain:')
print(df_test["label"].value_counts())
```

```
Viestien lukumäärät aiheittain:
kuntosali        1000
politiikka       1000
hikky            1000
penkkiurheilu    1000
televisio        1000
sota             1000
muoti            1000
seksuaalisuus    1000
ajoneuvot        1000
pelit            1000
Name: label, dtype: int64
```

3 Datasettien tokenisointi ja vektorisointi

Koneoppimismenetelmät eivät osaa hyödyntää tekstimuotoista dataa sellaisenaan. Jotta luonnollista kieltä voitaisiin analysoida ja käsitellä koneoppimismenetelmin, siitä on ensin johdettava numeerisia vektorimuotoisia piirteitä.

3.1 Tokenisointi NLTK-kirjaston avulla

Ensimmäinen vaihe on tekstin tokenisointi, eli tekstin jakaminen sanoihin. Tokenisointiin voi käyttää esim. Natural Language Toolkit (NLTK)-kirjastosta löytyvää word_tokenize metodia. NLTK-kirjaston käyttöä on opetettu kattavasti kirjassa Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit, jonka ovat kirjoittaneet Steven Bird, Ewan Klein, and Edward Loper. Kirja on luettavissa kokonaisuudessaan osoitteessa https://www.nltk.org/book/ (https://www.nltk.org/book/).

In [10]:
```
df_train['tokenized'] = df_train['text'].apply(nltk.word_tokenize)
```

In [11]:
```
print('Ensimmäinen viesti ennen tokenisointia:\n', df_train['text'][0], '\n')
print('ja tokenisoinnin jälkeen:\n', df_train['tokenized'][0])
```

```
Ensimmäinen viesti ennen tokenisointia:
 Kattokaas bemarijonnet tää , ei taida teidän 316 tarjota ihan näin tasasta kyyt
iä . :D

ja tokenisoinnin jälkeen:
 ['Kattokaas', 'bemarijonnet', 'tää', ',', 'ei', 'taida', 'teidän', '316', 'tarj
ota', 'ihan', 'näin', 'tasasta', 'kyytiä', '.', ':', 'D']
```

Jotta esim. Kissa ja kissa laskettaisiin jatkossa samaksi sanaksi, muutetaan kaikki isot kirjaimet pieniksi.

In [12]:
```
df_train['tokenized']=df_train['tokenized'].apply(lambda x: [w.lower() for w in x])
print('Kaikki pienin kirjaimin:\n', df_train['tokenized'][0])
```

```
Kaikki pienin kirjaimin:
 ['kattokaas', 'bemarijonnet', 'tää', ',', 'ei', 'taida', 'teidän', '316', 'tarj
ota', 'ihan', 'näin', 'tasasta', 'kyytiä', '.', ':', 'd']
```

3.2 Tokenisointi ja vektorisointi Scikit-learn-kirjaston avulla

Myös Scikit-learnin kirjastosta löytyy käteviä luonnollisen kielen käsittelyyn ja koneoppimiseen kehitettyjä metodeja. Hyvä tutoriaali aiheesta löytyy täältä: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text). Jatketaan piirrevektorien generointia kyseisen tutoriaalin avulla.

Yksinkertaisin tapa muodostaa teksteistä piirrevektoreita on ns. bag-of-words-menetelmä. Siinä jokaiselle opetusdatasetissä esiintyvälle eri sanalle annetaan ensin oma indeksi. Tämän jälkeen jokaiselle dokumentille muodostetaan vektori laskemalla kunkin sanan esiintymismäärä kyseisessä dokumentissa ja tallentamalla tämä luku vektoriin aina kyseistä sanaa vastaavan indeksin kohdalle. Bag-of-words -menetelmä on siis kiinnostunut vain sanojen lukumääristä, se ei ota huomioon sanajärjestystä. Tehdään vektorisointi käyttämällä CountVectorizer-metodia. CountVectorizer tekee samalla myös tekstin esiprosessoinnin ja tokenisoinnin.

Mutta ihan ensin muutetaan datakehikon data laskentaan paremmin soveltuviksi numpy-vektoreiksi ja ladataan ne muistiin, jottei niitä tarvitse laskea aina uudelleen, kun työkirja tauon jälkeen avataan uudelleen.

In [13]:
```
X_train = np.array(df_train['text'])
y_train = np.array(df_train['label'])

X_dev = np.array(df_dev['text'])
y_dev = np.array(df_dev['label'])

X_test = np.array(df_test['text'])
y_test = np.array(df_test['label'])

np.save('X_train.npy', X_train)
np.save('y_train.npy', y_train)
np.save('X_dev.npy', X_dev)
np.save('y_dev.npy', y_dev)
np.save('X_test.npy', X_test)
np.save('y_test.npy', y_test)
```

In [14]:
```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
print(X_train_counts.shape)
X_train_counts[0]
```

```
(100000, 399520)
```

Out[14]:
```
<1x399520 sparse matrix of type '<class 'numpy.int64'>'
        with 12 stored elements in Compressed Sparse Row format>
```

Tuloksena saatiin siis 100000x399520 -kokoinen matriisi, jossa 100000 on viestien määrä ja 399520 on erilaisten sanojen lukumäärä opetusdatassa. Metodi löysi esim. ensimmäisestä viestistä 12 eri tokenia. Näin ollen 399520 pituisessa vektorissa on 12 kohdassa luku ja kaikki muut kohdat saavat arvon 0. Tilan säästämiseksi Scikit-learn käyttääkin ns. harvoja matriiseja, jotka tallentavat vain nollasta poikkeavat tiedot.

Selvitetään, mitkä ovat opetusaineiston eniten käytetyt sanat. Kunkin sanan kokonaismäärä saadaan laskemalla dokumenttikohtaiset sanamäärät yhteen. Listauksen tekemiseen otin vinkkiä täältä: https://medium.com/@cristhianboujon/how-to-list-the-most-common-words-from-text-corpus-using-scikit-learn-dad4d0cab41d (https://medium.com/@cristhianboujon/how-to-list-the-most-common-words-from-text-corpus-using-scikit-learn-dad4d0cab41d)

In [15]:
```
sum_X_train_counts = X_train_counts.sum(axis=0)
```

In [16]:
```
words_sum = [(word, sum_X_train_counts[0, idx]) for word, idx in count_vect.fit(X_t
rain).vocabulary_.items()]
words_sum =sorted(words_sum, key = lambda x: x[1], reverse=True)
```

Listataan 30 yleisintä sanaa ja niiden esiintymismäärät koulutusaineistossa.

In [17]:
```
words_sum[:30]
```

Out[17]:
```
[('ja', 112066),
 ('on', 90328),
 ('ei', 65747),
 ('että', 47182),
 ('se', 32955),
 ('niin', 29982),
 ('mutta', 29158),
 ('kun', 28159),
 ('jos', 26662),
 ('ole', 26618),
 ('en', 18875),
 ('tai', 18179),
 ('kuin', 17325),
 ('ihan', 15535),
 ('nyt', 15224),
 ('oli', 14662),
 ('sen', 14560),
 ('vain', 13234),
 ('mitä', 13158),
 ('sitten', 13090),
 ('voi', 12813),
 ('kyllä', 12730),
 ('mitään', 12341),
 ('sitä', 12136),
 ('koska', 11921),
 ('vaan', 11406),
 ('joka', 11194),
 ('ovat', 11151),
 ('olla', 11036),
 ('vaikka', 10384)]
```

NLTK:sta on mahdollista ladata muulla aineistolla valmiiksi opetetut pysäytyssanat (Stop words) eli lista sanoista, joita ei huomioida piirrevektoreissa. NLTK:n pysäytyssanoja on yhteensä 235. Nämä pysäytyssanat ovat niin yleisiä sanoja, ettei niistä ole hyötyä luokittelun kannalta.

In [4]:
```
# nltk.download('stopwords')
stop_w = nltk.corpus.stopwords.words('finnish')
np.save('stop_w', stop_w)
print(stop_w)
len(stop_w)
```

```
['olla', 'olen', 'olet', 'on', 'olemme', 'olette', 'ovat', 'ole', 'oli', 'olisi
', 'olisit', 'olisin', 'olisimme', 'olisitte', 'olisivat', 'olit', 'olin', 'olim
me', 'olitte', 'olivat', 'ollut', 'olleet', 'en', 'et', 'ei', 'emme', 'ette', 'e
ivät', 'minä', 'minun', 'minut', 'minua', 'minussa', 'minusta', 'minuun', 'minul
la', 'minulta', 'minulle', 'sinä', 'sinun', 'sinut', 'sinua', 'sinussa', 'sinust
a', 'sinuun', 'sinulla', 'sinulta', 'sinulle', 'hän', 'hänen', 'hänet', 'häntä',
'hänessä', 'hänestä', 'häneen', 'hänellä', 'häneltä', 'hänelle', 'me', 'meidän',
'meidät', 'meitä', 'meissä', 'meistä', 'meihin', 'meillä', 'meiltä', 'meille', '
te', 'teidän', 'teidät', 'teitä', 'teissä', 'teistä', 'teihin', 'teillä', 'teilt
ä', 'teille', 'he', 'heidän', 'heidät', 'heitä', 'heissä', 'heistä', 'heihin', '
heillä', 'heiltä', 'heille', 'tämä', 'tämän', 'tätä', 'tässä', 'tästä', 'tähän',
'tallä', 'tältä', 'tälle', 'tänä', 'täksi', 'tuo', 'tuon', 'tuotä', 'tuossa', 't
uosta', 'tuohon', 'tuolla', 'tuolta', 'tuolle', 'tuona', 'tuoksi', 'se', 'sen',
'sitä', 'siinä', 'siitä', 'siihen', 'sillä', 'siltä', 'sille', 'sinä', 'siksi',
'nämä', 'näiden', 'näitä', 'näissä', 'näistä', 'näihin', 'näillä', 'näiltä', 'nä
ille', 'näinä', 'näiksi', 'nuo', 'noiden', 'noita', 'noissa', 'noista', 'noihin
', 'noilla', 'noilta', 'noille', 'noina', 'noiksi', 'ne', 'niiden', 'niitä', 'ni
issä', 'niistä', 'niihin', 'niillä', 'niiltä', 'niille', 'niinä', 'niiksi', 'kuk
a', 'kenen', 'kenet', 'ketä', 'kenessä', 'kenestä', 'keneen', 'kenellä', 'kenelt
ä', 'kenelle', 'kenenä', 'keneksi', 'ketkä', 'keiden', 'ketkä', 'keitä', 'keissä
', 'keistä', 'keihin', 'keillä', 'keiltä', 'keille', 'keinä', 'keiksi', 'mikä',
'minkä', 'minkä', 'mitä', 'missä', 'mistä', 'mihin', 'millä', 'miltä', 'mille',
'minä', 'miksi', 'mitkä', 'joka', 'jonka', 'jota', 'jossa', 'josta', 'johon', 'j
olla', 'jolta', 'jolle', 'jona', 'joksi', 'jotka', 'joiden', 'joita', 'joissa',
'joista', 'joihin', 'joilla', 'joilta', 'joille', 'joina', 'joiksi', 'että', 'ja
', 'jos', 'koska', 'kuin', 'mutta', 'niin', 'sekä', 'sillä', 'tai', 'vaan', 'vai
', 'vaikka', 'kanssa', 'mukaan', 'noin', 'poikki', 'yli', 'kun', 'niin', 'nyt',
'itse']
```

Out[4]: 235

Verrataan NLTK:n pysäytyssanalistaa oman koulutusaineistomme 235 yleisimpään sanaan. Listataan samat sanat sekä ne sanat NLTK:n pysäytyssanalistalta, joita ei löydy koulutusaineiston 235 yleisimmästä sanasta.

In [19]:
```
top235= [i[0] for i in words_sum[:235]]
same_stop_word= [w for w in stop_w if w in top235]
print('Samat sanat:\n',len(same_stop_word))
print(same_stop_word)

not_stop_word = [w for w in stop_w if w not in top235]
print('\nLoput pysäytyssanat:\n',len(not_stop_word))
print(not_stop_word)
```

```
Samat sanat:
 72
['olla', 'olen', 'olet', 'on', 'ovat', 'ole', 'oli', 'olisi', 'ollut', 'en', 'et
', 'ei', 'eivät', 'minä', 'sinä', 'hän', 'he', 'tämä', 'tämän', 'tätä', 'tässä',
'tästä', 'tähän', 'tuo', 'tuon', 'tuossa', 'se', 'sen', 'sitä', 'siinä', 'siitä
', 'siihen', 'sillä', 'sinä', 'nämä', 'näitä', 'nuo', 'ne', 'niiden', 'niitä', '
mikä', 'mitä', 'missä', 'mistä', 'minä', 'miksi', 'joka', 'jonka', 'jota', 'joss
a', 'jotka', 'että', 'ja', 'jos', 'koska', 'kuin', 'mutta', 'niin', 'sekä', 'sil
lä', 'tai', 'vaan', 'vai', 'vaikka', 'kanssa', 'mukaan', 'noin', 'yli', 'kun', '
niin', 'nyt', 'itse']

Loput pysäytyssanat:
 163
['olemme', 'olette', 'olisit', 'olisin', 'olisimme', 'olisitte', 'olisivat', 'ol
it', 'olin', 'olimme', 'olitte', 'olivat', 'olleet', 'emme', 'ette', 'minun', 'm
inut', 'minua', 'minussa', 'minusta', 'minuun', 'minulla', 'minulta', 'minulle',
'sinun', 'sinut', 'sinua', 'sinussa', 'sinusta', 'sinuun', 'sinulla', 'sinulta',
'sinulle', 'hänen', 'hänet', 'häntä', 'hänessä', 'hänestä', 'häneen', 'hänellä',
'häneltä', 'hänelle', 'me', 'meidän', 'meidät', 'meitä', 'meissä', 'meistä', 'me
ihin', 'meillä', 'meiltä', 'meille', 'te', 'teidän', 'teidät', 'teitä', 'teissä
', 'teistä', 'teihin', 'teillä', 'teiltä', 'teille', 'heidän', 'heidät', 'heitä
', 'heissä', 'heistä', 'heihin', 'heillä', 'heiltä', 'heille', 'tallä', 'tältä',
'tälle', 'tänä', 'täksi', 'tuotä', 'tuosta', 'tuohon', 'tuolla', 'tuolta', 'tuol
le', 'tuona', 'tuoksi', 'siltä', 'sille', 'siksi', 'näiden', 'näissä', 'näistä',
'näihin', 'näillä', 'näiltä', 'näille', 'näinä', 'näiksi', 'noiden', 'noita', 'n
oissa', 'noista', 'noihin', 'noilla', 'noilta', 'noille', 'noina', 'noiksi', 'ni
issä', 'niistä', 'niihin', 'niillä', 'niiltä', 'niille', 'niinä', 'niiksi', 'kuk
a', 'kenen', 'kenet', 'ketä', 'kenessä', 'kenestä', 'keneen', 'kenellä', 'kenelt
ä', 'kenelle', 'kenenä', 'keneksi', 'ketkä', 'keiden', 'ketkä', 'keitä', 'keissä
', 'keistä', 'keihin', 'keillä', 'keiltä', 'keille', 'keinä', 'keiksi', 'minkä',
'minkä', 'mihin', 'millä', 'miltä', 'mille', 'mitkä', 'josta', 'johon', 'jolla',
'jolta', 'jolle', 'jona', 'joksi', 'joiden', 'joita', 'joissa', 'joista', 'joihi
n', 'joilla', 'joilta', 'joille', 'joina', 'joiksi', 'poikki']
```

Tähän asti olemme laskeneet viestien sanamäärävektoriin mukaan kaikki sanat, jolloin niistä on tullut valtavan pitkiä (tässä tapauksessa lähes 400 000). Varsinkin neuroverkkopohjaisten menetelmien kouluttaminen vaatii niin paljon laskentaa, ettei tavallisen kotikoneen muisti ja laskentateho useinkaan riitä näin pitkien vektoreiden käsittelyyn. Onneksi CountVectorizer-metodissa on mahdollista rajata parametreilla sanaston kokoa.

Sellaisista sanoista, joita löytyy joka dokumentista, ei juurikaan ole hyötyä luokittelun kannalta. Max_df-parametrilla on mahdollista poistaa kaikki sanat, joita löytyy esim. yli 70%:ssa dokumenteista. Vastaavasti min_df-parametrilla voidaan rajata pois sellaiset sanat, joita esiintyy vain hyvin harvoissa dokumenteissa.

Kokeilin Naive Bayes -menetelmää käyttäen erilaisia parametrikombinaatioita ja huomasin, että NLTK-kirjaston pysäytyssanojen poisto rajasi sanaston kokoa max_df:ää (70%) enemmän ja antoi paremman ennustetarkkuuden. Parhaan ennustetarkkuuden (Naive Bayes) sain, kun rajasin lisäksi pois sanat, joita esiintyy alle 0.003%:ssa dokumentteja. Sanaston kooksi jää näillä parametreilla 98225 sanaa. Muodostetaan count-vektorit näillä parametreilla kaikille dataseteille.

In [20]:
```
count_vect_s003 = CountVectorizer(stop_words=stop_w, min_df=0.00003)
count_vect_s003.fit(X_train)

X_train_counts_stop_w003 = count_vect_s003.transform(X_train)
X_dev_counts_stop_w003 = count_vect_s003.transform(X_dev)
X_test_counts_stop_w003 = count_vect_s003.transform(X_test)

print(X_train_counts_stop_w003.shape)
```

```
(100000, 98225)
```

Tehdään myös esimerkin vuoksi sellaiset versiot, joihin on otettu mukaan myös kahden ja kolmen peräkkäisen sanan yhdistelmät (2- ja 3-grammit).

In [21]:
```
count_vect_s0033 = CountVectorizer(ngram_range=(1, 3), stop_words=stop_w, min_df=0.
00003)
count_vect_s0033.fit(X_train)

X_train_counts_stop_w0033 = count_vect_s0033.transform(X_train)
X_dev_counts_stop_w0033 = count_vect_s0033.transform(X_dev)
X_test_counts_stop_w0033 = count_vect_s0033.transform(X_test)

print(X_train_counts_stop_w0033.shape)
```

```
(100000, 200895)
```

Sanojen lukumääristä muodostetut piirrevektorit eivät välttämättä sellaisenaan ole riittävän kuvaavia. Yleensä dokumentit ovat eri pituisia, jolloin sanojen keskimääräiset lukumäärät voivat vaihdella paljonkin, vaikka dokumentit edustaisivat samaa aihepiiriä. Siksi kunkin sanan lukumäärä dokumentissa kannattaa jakaa dokumentin kaikkien sanojen lukumäärällä. Näin laskettua suuretta kutsutaan termifrekvenssiksi (tf).

Dokumenttifrekvenssi (df) puolestaan ilmaisee niiden dokumenttien lukumäärän, joissa kyseinen termi esiintyy. Eli jos sanan dokumenttifrekvenssi on pieni, sana esiintyy yleensä dokumentissa joko monta kertaa tai sitten ei lainkaan. Jos taas dokumenttifrekvenssi on suuri, kyseinen sana esiintyy tasaisesti kaikissa dokumenteissa. Tällaisen sanan käyttö piirrevektorissa ei juurikaan tuo luokittelulle hyödyllistä lisäinformaatiota.

Kun termifrekvenssi jaetaan dokumenttifrekvenssillä (kerrotaan käänteisellä dokumenttifrekvenssillä), saadaan tf_idf. Sekä tf että tf_idf voidaan laskea Scikit-learnin TfidfTransformer-metodin avulla.

In [22]:
```
# kun muodostetaan tf, annetaan use_idf-parametrille arvo False:
tf_transformer = TfidfTransformer(use_idf=False)
X_train_tf = tf_transformer.fit_transform(X_train_counts_stop_w003)
X_train_tf.shape
```

Out[22]:
```
(100000, 98225)
```

Käytetään aikaisemmin parametreilla rajattuja sanamäärävektoreita ja muodostetaan näitä vastaavat tf_idf-piirrematriisit koulutus-, evaluointi ja testidatalle.

In [24]:
```
tfidf_transformer003 = TfidfTransformer()
tfidf_transformer003.fit(X_train_counts_stop_w003)

X_train_tfidf003 = tfidf_transformer003.transform(X_train_counts_stop_w003)
X_dev_tfidf003 = tfidf_transformer003.transform(X_dev_counts_stop_w003)
X_test_tfidf003 = tfidf_transformer003.transform(X_test_counts_stop_w003)
```

In [26]:
```
tfidf_transformer0033 = TfidfTransformer()
tfidf_transformer0033.fit(X_train_counts_stop_w0033)

X_train_tfidf0033 = tfidf_transformer0033.transform(X_train_counts_stop_w0033)
X_dev_tfidf0033 = tfidf_transformer0033.transform(X_dev_counts_stop_w0033)
X_test_tfidf0033 = tfidf_transformer0033.transform(X_test_counts_stop_w0033)
```

4. Viestien luokittelu tekstisisällön perusteella

Tutkitaan, kuinka hyvin viestien aiheluokan pystyy ennustamaan tekstisisällön perusteella. Luokittelua on mahdollista tehdä erilaisilla koneoppimisalgoritmeilla, jotka käyttävät niitä varten tekstisisällöstä johdettuja algoritmille sopivia numeerisia vektorimuotoisia piirteitä.

4.1 Naive Bayes

Aloitetaan yksinkertaisella todennäköisyyksiin perustuvalla Naive Bayes -luokittelumenetelmällä. Bayesilaisen päätösteorian mukainen luokittelija suosittelee päätöksiä, jotka minimoivat odotetun kokonaisriskin. Tässä tapauksessa riskinä on tehdä virhe luokittelussa. Naive Bayes -menetelmässä tehdään laskentaa merkittävästi yksinkertaistava oletus, että datasta johdetut piirteet olisivat keskenään riippumattomia. Vaikka tämä oletus yksinkertaistaa mallia liikaakin, Naive Bayesilla saadaan usein yllättävän hyviä tuloksia edistyneempiin menetelmiin nähden erittäin vähäisellä laskentakapasiteetilla.

Käytetään Scikit-learnin Pipeline-luokkaa, joka helpottaa menetelmän validointia eri hyperparametreilla. Kokeillaan ensin mallia, jossa pysäytyssanoja ei ole poistettu.

In [29]:
```
text_NB = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('classifier', MultinomialNB()),
 ])
```

In [30]:
```
text_NB.fit(X_train, y_train)
```

Out[30]:
```
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error
='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip...f=False, use_idf=True)), ('classifier', MultinomialNB(alpha=1.0,
class_prior=None, fit_prior=True))])
```

In [31]:
```
predicted = text_NB.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[31]:
```
0.7055
```

Näillä parametreilla päästiin siis 70,6%:n tarkkuuteen. Kokeillaan seuraavaksi muuten samaa mallia, mutta poistetaan NLTK:n pysäytyssanat.

In [32]:
```
text_NB_s = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w)),
    ('tfidf', TfidfTransformer()),
    ('classifier', MultinomialNB()),
 ])
```

In [33]:
```
text_NB_s.fit(X_train, y_train)
```

Out[33]:
```
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error
='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None,
        stop_words=['olla', 'o...f=False, use_idf=True)), ('classifier', Multino
mialNB(alpha=1.0, class_prior=None, fit_prior=True))])
```

In [34]:
```
predicted = text_NB_s.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[34]:
```
0.7207
```

Pysäytyssanat poistamalla tarkkuus parani 72,1%:iin. Kokeilemalla erilaisia hyperparametrikombinaatioita saavutin parhaan tarkkuuden (73,1%) poistamalla pysäytyssanat ja min_df-arvolla 0.00003.

In [5]:
```
text_NB_s = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w, min_df=0.00003)),
    ('tfidf', TfidfTransformer()),
    ('classifier', MultinomialNB()),
 ])

text_NB_s.fit(X_train, y_train)

predicted = text_NB_s.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[5]:
```
0.7306
```

Tutkitaan seuraavaksi scikit-learnin confusion-matrixia apua käyttäen, miten eri luokat erottuvat toisistaan. Matriisin rivit edustavat oikeita luokkia ja sarakkeet ennusteita. Diagonaalilla näkyy siis kyseisen luokan oikein luokiteltujen viestien lukumäärät. Esim. ensimmäisellä rivillä näkyy niiden dokumenttien luokitteluennusteet, joiden todellinen luokka on yksi eli 'Ajoneuvot'. Näistä 1000:sta luokkaan yksi kuuluvasta viestistä 772 on luokiteltu oikein, neljä on ennustettu virheellisesti luokkaan kaksi 'Hikky', jne.

In [36]:
```
print(confusion_matrix(predicted, y_dev))
```

```
[[772   4  13  11  17   9  11  11  24   5]
 [ 42 741  69  66  72  49 107 112  70  55]
 [ 31  20 766  41  35  44  14  49  23  31]
 [ 29  13  30 753  19  23   7  33  23  15]
 [ 19  11   8  20 690  62  11  12  25  33]
 [  7   6  13   7  36 685  10   8  19   8]
 [ 29  56  13  23  22  31 792  13 151  64]
 [ 35 119  57  46  27  27  22 745  26  29]
 [ 16   9  12  10  25  20   7  13 617  15]
 [ 20  21  19  23  57  50  19   4  22 745]]
```

Ongelmia näkyy olevan joka luokan kanssa.

Tähän asti olemme käyttäneet vain yksittäisiin sanoihin perustuvaa vektorisointia. Kokeillaan, paraneeko ennustettavuus, jos piirrevektoriin otetaan mukaan myös kaikki kahden tai kolmen peräkkäisen sanan yhdistelmät eli 2- ja 3-grammit.

In [37]:
```
text_NB_s = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 2), stop_words=stop_w, min_df=0.0000
3)),
    ('tfidf', TfidfTransformer()),
    ('classifier', MultinomialNB()),
 ])

text_NB_s.fit(X_train, y_train)

predicted = text_NB_s.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[37]:
```
0.718
```

In [38]:
```
text_NB_s = Pipeline([
    ('vect', CountVectorizer(ngram_range=(1, 3), stop_words=stop_w, min_df=0.0000
3)),
    ('tfidf', TfidfTransformer()),
    ('classifier', MultinomialNB()),
 ])

text_NB_s.fit(X_train, y_train)

predicted = text_NB_s.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[38]:
```
0.7173
```

Tulos ei parantunut käyttämällä n-grammeja. Kokeilin myös ilman pysäytyssanojen ja harvinaisten sanojen poistoa, mutta tulos oli silloin vielä huonompi.

Paras validointitulos käyttäen Naive Bayes -menetelmää jäi 73.1%:in tarkkuuteen. Ajan kyseisen koulutusajon uudelleen ja arvioin sen jälkeen sen ennustekyvyn luotettavuuden testiaineistolla.

In [7]:
```
predicted = text_NB_s.predict(X_test)
np.mean(predicted == y_test)
```

Out[7]:
```
0.7154
```

Testiaineistolla Naive Bayes -menetelmän tarkkuudeksi saatiin 71,6%

4.2 Tukivektorikone (support vector machine)

Tutkitaan, minkälaiseen tarkkuuteen pääsemme käyttämällä tukivektorikone-menetelmää. Tukivektorikone on lineaarinen luokittelija, joka pyrkii erottamaan kaksi luokkaa toisistaan sovittamalla niiden väliin päätöshypertason ja tämän tason kanssa yhdensuuntaiset toisistaan mahdollisimman etäälle mutta yhtä kauas päätöstasosta sijoittuvat marginaalihypertasot. Parhaassa tapauksessa marginaalitasot pystytään muodostamaan niin, ettei niiden väliin jää yhtään datapistettä. Silloin marginaalitasojen paikan määräävät ne datapisteet, jotka ovat lähimpänä päätöstasoa. Näitä datapisteitä kutsutaan tukivektoreiksi. Usein eri luokkia ei pystytä erottamaan toisistaan tasolla täydellisesti toisistaan. Silloin käytetään ns. joustavan marginaalin luokitinta, mutta silloinkin tukivektoreiksi kutsutaan niitä datapisteitä, jotka määrittävät marginaalitasojen paikan.

Käytetään tähän https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html (https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html) -tutorialin mukaisesti Scikit-learnin SGDClassifier-metodia.

Kokeillaan ensin ottamalla kaikki sanat huomioon:

In [39]:
```
text_svm = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='hinge', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=5, tol=None)),
])

text_svm.fit(X_train, y_train)
predicted = text_svm.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[39]: 0.7141

Seuraavaksi kokeillaan poistamalla pysäytyssanat ja min_df:n arvolla 0.0003:

In [40]:
```
text_svm = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w, min_df=0.00003)),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='hinge', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=5, tol=None)),
])

text_svm.fit(X_train, y_train)
predicted = text_svm.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[40]: 0.7136

Tulos huononi hieman tällä rajauksella, joten kokeillaan vielä poistamalla ainoastaan pysäytyssanat:

In [41]:
```
text_svm = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w)),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='hinge', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=5, tol=None)),
])

text_svm.fit(X_train, y_train)
predicted = text_svm.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[41]: 0.7186

Paras tulos saatiin poistamalla pysäytyssanat. Jäimme silti jälkeen Naive Bayes -menetelmällä saavutetusta tarkkuudesta. SGDClassifier-metodin dokumentaation https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html) mukaan parhaisiin tuloksiin pääsemiseksi datan olisi hyvä olla normalisoitua (datan keskiarvo on 0 ja varianssi 1). Tähän päästäisiin z-standardisoimalla piirrematriisi, mutta koska harva (sparse) piirrematriisimme muuttuisi silloin tiheäksi, se ei enää mahtuisi tavallisen koneen muistiin.

Yritetään vielä parantaa tulosta kokeilemalla eri hyperparametrien arvoja. Tutoriaalissa on opastettu myös GridSearchCV-metodin käyttö, jolla voidaan kätevästi käydä läpi eri parametrikombinaatiot. Kokeillaan svm-menetelmää käyttäen sanoja sekä sanoja ja 2-grammeja sekä rangaistus (penalty) parametria arvoilla 0,01/ 0,001. GridSearchCV käyttää cross validointia löytääkseen parhaan parametrikombinaation. Tämän vuoksi se vie erittäin paljon laskenta-aikaa. Siksi "leikkaa - liimaa - muuta hyperparametrien arvoja" -menetelmä on tavallisella koneella huomattavasti nopeampi tapa kokeilla eri kombinaatioita.

In [42]:
```
parameters = {
    'vect__ngram_range': [(1, 1), (1, 2)],
    'classifier__alpha': (1e-2, 1e-3),
}
```

In [43]:
```
gs = GridSearchCV(text_svm, parameters, cv=5, iid=False, n_jobs=-1).fit(X_train, y_
train)
```

In [44]:
```
print(gs.best_score_)
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, gs.best_params_[param_name]))
```

```
0.7295299999999999
classifier__alpha: 0.001
vect__ngram_range: (1, 1)
```

Paras tulos saatiin ilman 2-grammeja ja rangaistusparametrin arvolla 0,001 eli samoilla arvoilla kuin olimme yllä jo kokeilleet. Cross validoinnin vuoksi tarkkuus näyttää paremmalta kuin yllä (ylempi laskettu erillisellä validointidatalla).

Kokeillaan vielä alkuperäistä mallia useammalla (30) iteraatiolla.

In [45]:
```
text_svm = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w)),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='hinge', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=30, tol=None)),
])

text_svm.fit(X_train, y_train)
predicted = text_svm.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[45]:
```
0.7189
```

Iteraatioiden määrän lisääminen ei käytännössä parantanut tulosta. Lineaarisissa malleissa on ongelmallista, jos piirteiden määrä on isompi kuin datapisteiden. Kokeillaan siksi vielä piirteiden rajoittamista 100 000:een.

In [46]:
```
text_svm = Pipeline([
    ('vect', CountVectorizer(max_features=100000, ngram_range=(1,1), stop_words=sto
p_w)),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='hinge', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=5, tol=None)),
])

text_svm.fit(X_train, y_train)
predicted = text_svm.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[46]: 0.7136

Tämäkään ei valitettavasti parantanut tulosta. Paras tarkkuus käyttäen tukivektorikonetta CGDClassifier-luokan avulla jäi 71,9%:iin.

On yllättävää, ettei tukivektorikoneella päästy Naive Bayesin tarkkuuteen. Kokeillaan vielä parantaa tulosta käyttäen toista Scikit-lernin tukivektorikone-luokkaa LinearSVC. Edellä pelkkien pysäytyssanojen poisto antoi parhaan tuloksen. Kun kokeilin kaikki samat kombinaatiot kuin yllä, sain parhaan validointituloksen käyttäen koko sanastoa ja ilman n-grammeja.

In [17]:
```
text_svm = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('classifier', LinearSVC(penalty='l2', loss='squared_hinge',
                             dual=True, tol=0.0001, C=1.0,
                             multi_class='ovr', fit_intercept=True,
                             intercept_scaling=1, class_weight=None,
                             verbose=0, random_state=None, max_iter=30)),
])

text_svm.fit(X_train, y_train)
predicted = text_svm.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[17]: 0.7367

Tällä tukivektorikoneella saavutettu tarkkuus 73,7% on parempi kuin Naive Bayesin 73.1%. Kokeillaan vielä parantaa tulosta iteroimalla säätelyparametrin C arvoja (0.9, 1.1, 0.8, 0.7, 0.6, 0.5, 0.55, 0.65, 0.59, 0,61).

In [8]:
```
text_svm = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('classifier', LinearSVC(penalty='l2', loss='squared_hinge',
                             dual=True, tol=0.0001, C=0.6,
                             multi_class='ovr', fit_intercept=True,
                             intercept_scaling=1, class_weight=None,
                             verbose=0, random_state=None, max_iter=30)),
])

text_svm.fit(X_train, y_train)
predicted = text_svm.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[8]: 0.7404

Paras tulos saavutettiin C:n arvolla 0.6, jolloin tarkkuudeksi saatiin 74,0%. Tämä on tähän mennessä paras validointidatalla saavutettu tulos. Katsotaan vielä, minkälainen tarkkuus saadaan testiaineistolla.

In [9]:
```
predicted = text_svm.predict(X_test)
np.mean(predicted == y_test)
```

Out[9]: 0.7197

Testiaineiston perusteella tarkkuus on 72.0%

4.3 Muita lineaarisia malleja

Dokumenttien vektorisoinnissa syntyy todella pitkiä vektoreita. Tämä rajaa merkittävästi, mitä koneoppimismenetelmiä voidaan tavallisella kotikoneella käyttää. Kokeilin kahta hyvin yleisesti käytettyä koneoppimismenetelmää (K nearest neighbors sekä Regularized linear model with ridge regression), mutta molemmat kaatuivat. Nearest neighbors ilmoitti kyseessä olevan muistin loppumisen, kun taas Ridge-menetelmän käyttö kaatoi Kernelin.

SGDClassifier-metodia voidaan käyttää myös muiden kuin tukivektorikoneen käyttöön. Demotaan siis vielä muutamaa niistä:

1. ‘log’: logistic regression
2. ‘perceptron’: perceptron algorithm
3. ‘squared_hinge’: like hinge (SVM) but is quadratically penalized

In [47]:
```
text_log = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w)),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='log', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=5, tol=None)),
])

text_log.fit(X_train, y_train)
predicted = text_log.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[47]: 0.5671

In [48]:
```
text_perceptron = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w)),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='perceptron', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=5, tol=None)),
])

text_perceptron.fit(X_train, y_train)
predicted = text_perceptron.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[48]: 0.677

In [49]:
```
text_squared_hinge = Pipeline([
    ('vect', CountVectorizer(stop_words=stop_w)),
    ('tfidf', TfidfTransformer()),
    ('classifier', SGDClassifier(loss='squared_hinge', penalty='l2',
                                 alpha=1e-3, random_state=42,
                                 max_iter=5, tol=None)),
])

text_squared_hinge.fit(X_train, y_train)
predicted = text_squared_hinge.predict(X_dev)
np.mean(predicted == y_dev)
```

Out[49]:
```
0.645
```

Koska alustavat tulokset eivät ole tämän lupaavampia, ei näitä menetelmiä lähdetä tutkimaan tarkemmin. Kokeillaan sen sijaan seuraavaksi neuroverkkopohjaista lähestymistä.

4.4 Neuroverkko

Neuroverkkopohjaiset menetelmät ovat epälineaarisia luokittelijoita. Nyt siis eri luokat pyritään erottamaan toisistaan epälineaarisella "rajalla". Epälineaarisuuden vuoksi malliin tarvitaan mukaan epälineaarisia soluja sisältäviä kerroksia. Mikä tahansa funktio on mahdollista approksimoida monen paikallisen funktion summana. Teoriassa jo kaksi piilotettua tasoa sisältävällä neuroverkolla, joissa on riittävä määrä epälineaarisia soluja, voidaan kuvata mitkä tahansa monimutkaiset päätösalueet. Neuroverkoilla on siis teoriassa rajaton selityskapasiteetti. Haasteena onkin, että ne voivat liiankin helposti "oppia ulkoa" opetusdatan ja ylisovittua siihen oppimalla sellaisiakin piirteitä opetusdatasta, jotka eivät ole yleistettävissä uuteen dataan. Tällainen ylisovittunut malli ei siis toimi luotettavasti esim. validointidatalla. Tämän vuoksi kun neuroverkkoja opetetaan, seurataankin jatkuvasti validointidatalla saatavien tulosten kehittymistä ja lopetetaan opettaminen, kun validointidatalla lasketut tulokset alkavat huonontua. Lisäksi ylisovittumista pyritään ennaltaehkäisemään esim. dropout-kerroksella, joka esitellään työkirjan myöhemmässä vaiheessa.

Aloitetaan vektorisoimalla aiheluokat "one-hot"-vektoreiksi, joissa dokumentin oikean aiheluokan kohdalle tulee arvo 1 ja muiden aiheluokkien kohdalle arvo 0. Tallennetaan nämä muistiin myöhempää käyttöä varten

In [24]:
```
label_encoder=LabelEncoder()
label_encoder.fit(y_train)

print(label_encoder.classes_)
```

```
['ajoneuvot' 'hikky' 'kuntosali' 'muoti' 'pelit' 'penkkiurheilu'
 'politiikka' 'seksuaalisuus' 'sota' 'televisio']
```

In [25]:
```
y_train_int=label_encoder.transform(y_train)
labels_train = to_categorical(y_train_int)

y_dev_int=label_encoder.transform(y_dev)
labels_dev = to_categorical(y_dev_int)

y_test_int=label_encoder.transform(y_test)
labels_test = to_categorical(y_test_int)
```

In [26]:
```
np.save('labels_train.npy', labels_train)
np.save('labels_dev.npy', labels_dev)
np.save('labels_test.npy', labels_test)
```

Nyt esim. koulutusdatan ensimmäisen viestin luokkavektori näyttää tältä:

In [10]:
```
labels_train[0]
```

Out[10]:
```
array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

Jotta erilaisten neuroverkkojen kokeilu olisi helpompaa, tehdään funktio neuroverkon määrittelemiseksi. Koneeni tehorajoitteiden vuoksi kokeillaan neuroverkkoja, joissa on vain yksi tai kaksi piilotettua tasoa.

In [12]:
```
def create_model1(features, nodes1):
    model = models.Sequential()
    model.add(layers.Dense(nodes1, activation='relu', input_shape=(features,)))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

In [13]:
```
def create_model2(features, nodes1, nodes2):
    model = models.Sequential()
    model.add(layers.Dense(nodes1, activation='relu', input_shape=(features,)))
    model.add(layers.Dense(nodes2, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Käytetään ensin syötteenä sanojen lukumääriä. Jos kaikki sanat otettaisiin mukaan, vektoreista tulisi 399520:n pituisia. Käyttämäni koneen teho ei riitä niin suuren tietomäärän työstämiseen. Kokeillaan siis tavallista sanamäärävektoria, mutta rajoitetaan piirteiden määrä 100000 sanaan.

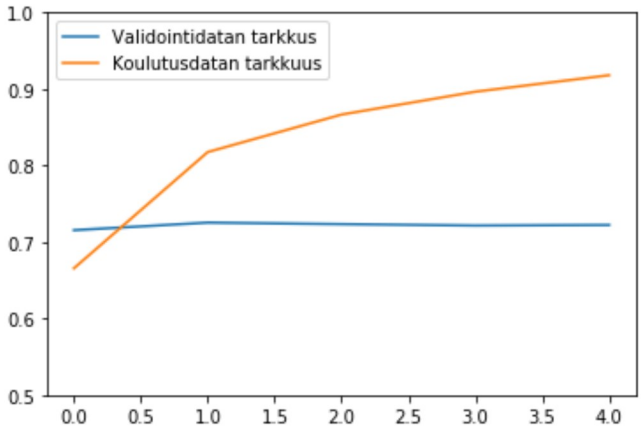
In [11]:
```
count_vect = CountVectorizer(max_features=100000)
count_vect.fit(X_train)

X_train_counts = count_vect.transform(X_train)
X_dev_counts = count_vect.transform(X_dev)
X_test_counts = count_vect.transform(X_test)
```

Kokeillaan neuroverkkomallia, jossa on yksi piilotettu taso, jossa on 128 solmua. Käytetään Early stopping -menetelmää, jossa joka optimointikierroksen jälkeen mallin virhemäärä lasketaan validointidatan avulla. Jos virhe ei pienene esim. kolmen viimeisen kierroksen aikana (parametri: patience), koulutusprosessi pysäytetään. Restore_best_weights-ohjausparametrin arvolla True opetuksen tuloksena saatu malli käyttää koulutuksen aikana validoituja parhaita parametriarvoja.

In [11]:
```
features = X_train_counts.shape[1]
nodes1 = 128
model = create_model1(features, nodes1)

es = EarlyStopping(monitor='val_loss', min_delta=0, patience=3, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 100, batch_size=150,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[11]:
```
0.7254
```

In [12]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Tarkkuudeksi saatiin validointidatalla 72,6%, joka on huonompi kuin tukivektorikoneella tai Naive Bayesilla saatu tarkkuus. Kuviosta huomataan, ettei validointidatan tarkkuus parane toisen kierroksen jälkeen (x-akselin indeksi 0 vastaa ensimmäistä validointitulosta, joka lasketaan ensimmäisen kierroksen jälkeen).

Näinkin yksinkertaisen neuroverkon koulutus vei koneellani noin 40 minuuttia. Neuroverkoilla olisi huikea selitysvoima, mutta niiden opettamiseen vaaditaan myös huikean paljon laskentatehoa. Tavallisella kotikoneella erilaisten yksikertaistenkin mallien koulutus ja niiden hyperparametrien virittäminen on helposti tuntien tai jopa päivien urakka.

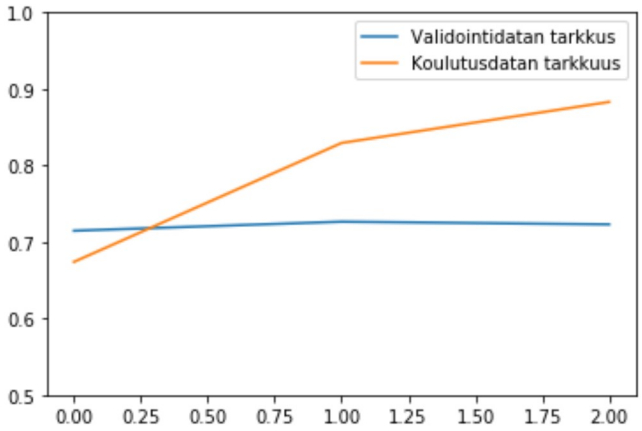
Koska nämä mallit ovat niin yksinkertaisia ja dataa on niinkin paljon (100 000 viestiä), neuroverkko tuntuu oppivan riittävästi jo ensimmäisillä kierroksilla. Ajan säästämiseksi tiputankin EarlyStoppingin patience-parametrin arvoon 1.

Kokeillaan lisätä solujen määrää 300:aan.

In [13]:
```
nodes1 = 300
model = create_model1(features, nodes1)

es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 100, batch_size=150,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[13]:
```
0.7265
```

In [14]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



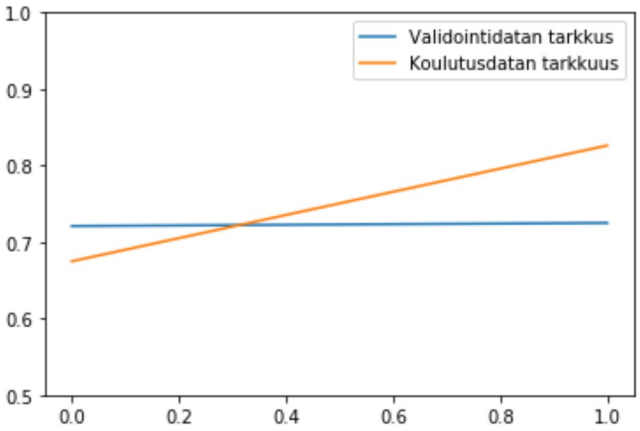
Tarkkuus parani vain hyvin vähän (nyt 72,7%, 124:llä solulla 72,6%). Kokeillaan seuraavaksi samaa, mutta muutetaan piirrevektoria niin, että jokainen vektorin luku voi saada vain arvon 0 tai 1. Eli mikäli kyseinen sana esiintyy vähintään kerran tekstissä, sanaa vastaava vektorin luku saa arvon 1.

In [15]:
```
count_vect = CountVectorizer(max_features=100000, binary=True)
count_vect.fit(X_train)

X_train_counts = count_vect.transform(X_train)
X_dev_counts = count_vect.transform(X_dev)
X_test_counts = count_vect.transform(X_test)
```

In [16]:
```
features = X_train_counts.shape[1]
nodes1 = 300
model = create_model1(features, nodes1)

es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 10, batch_size=100,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[16]:
```
0.7209
```

In [17]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Tulos huononi hiukan. Puhdistetaan välillä muistia.

In [12]:
```
# Muistin puhdistus: tuhoaa nykyisen TF-graafin ja luo uuden.
from keras import backend as K
K.clear_session()
```

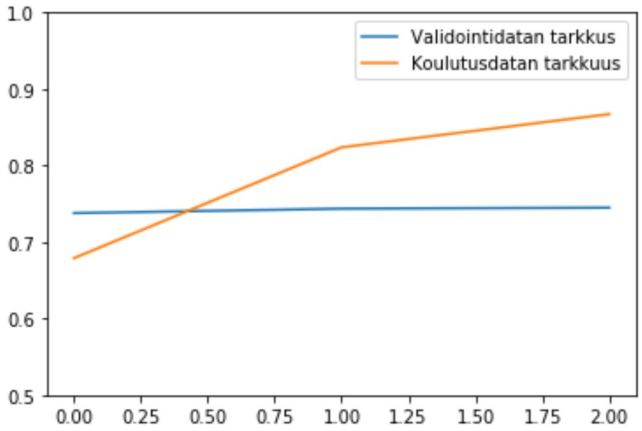
Kokeillaan seuraavaksi tf_idf-piirrematriisia.

In [4]:
```
count_vect = TfidfVectorizer(max_features=100000)
count_vect.fit(X_train)

X_train_counts = count_vect.transform(X_train)
X_dev_counts = count_vect.transform(X_dev)
X_test_counts = count_vect.transform(X_test)
```

In [34]:
```
features = X_train_counts.shape[1]
nodes1 = 300
model = create_model1(features, nodes1)

es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 10, batch_size=100,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[34]:
```
0.7436
```

In [35]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Saatiin tähän mennessä paras tulos 74,4%

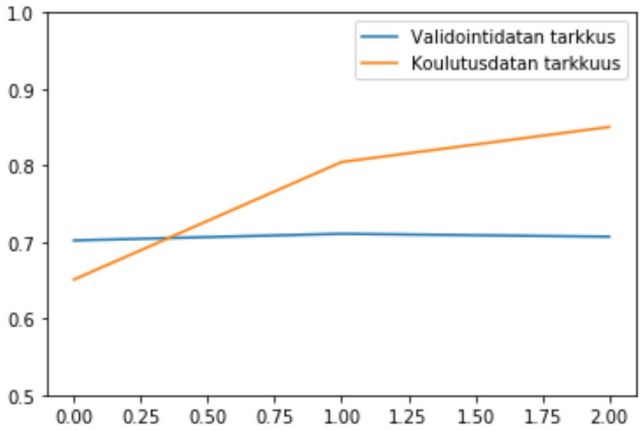
Pienennetään tf_idf-matriisia poistamalla laajasti esiintyvät (max_df=0.7) ja hyvin harvoissa dokumenteissa esiintyvät (min_df=0.0001) sanat. Otetaan mukaan myös 2- ja 3-grammit. Piirteitä on nyt yhteensä 63569:

In [38]:
```
count_vect = TfidfVectorizer(ngram_range=(1, 3), max_df=0.7, min_df=0.0001)
count_vect.fit(X_train)

X_train_counts = count_vect.transform(X_train)
X_dev_counts = count_vect.transform(X_dev)
X_test_counts = count_vect.transform(X_test)
```

In [40]:
```
features = X_train_counts.shape[1]
nodes1 = 300
model = create_model1(features, nodes1)

es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 10, batch_size=100,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[40]:
```
0.7108
```

In [41]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



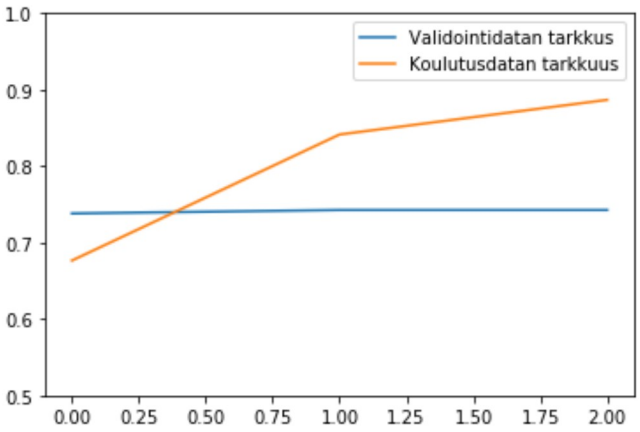
Tulos huononi jälleen. Kokeillaan vielä mallia, jossa on kaksi piilotettua tasoa.

In [21]:
```
count_vect = TfidfVectorizer(max_features=100000)
count_vect.fit(X_train)

X_train_counts = count_vect.transform(X_train)
X_dev_counts = count_vect.transform(X_dev)
X_test_counts = count_vect.transform(X_test)
```

In [44]:
```
features = X_train_counts.shape[1]
nodes1 = 256
nodes2 = 64
model = create_model2(features, nodes1, nodes2)

es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 10, batch_size=100,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[44]:
```
0.7426
```

In [45]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Tulokseksi saatiin 74,3%. Tähän mennessä paras validointitarkkuus 74,4% saatiin tf_idf-vektoreilla ja yhdellä piilotetulla kerroksella, jossa oli 300 solua.

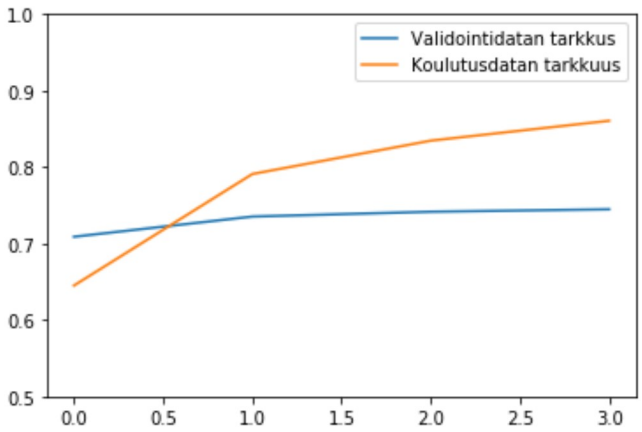
Tähän asti optimointialgoritmina on käytetty rmsprop-algoritmia sen oletusarvoisella oppimisnopeusparametrin arvolla 0.001. Kokeillaan pienentää parametrin arvo puoleen.

In [23]:
```
features = X_train_counts.shape[1]
model = models.Sequential()
model.add(layers.Dense(300, activation='relu', input_shape=(features,)))
model.add(layers.Dense(10, activation='softmax'))
rmsprop = optimizers.RMSprop(lr=0.0005)
model.compile(optimizer= rmsprop,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [24]:
```
es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 10, batch_size=100,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[24]:
```
0.7416
```

In [25]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



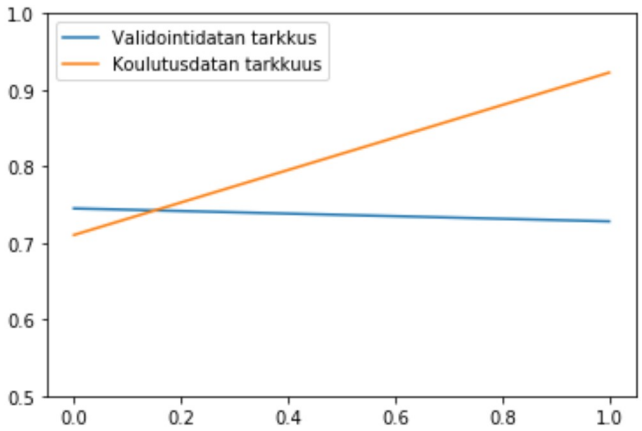
Ainakaan näin pieni muutos oppimisnopeusparametrissa ei parantanut tulosta. Vaihdetaan optimointialgoritmi Adamiksi. RMSProp mukauttaa neuroverkon parametrien oppimisnopeutta keskimääräisen ensimmäisen momentin (keskiarvo) perusteella. Adam ottaa tämän lisäksi huomioon myös gradienttien toiset momentit (varianssi). Käytetään Adamin oletusarvoisia parametriarvoja (learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False).

In [5]:
```
features = X_train_counts.shape[1]
model = models.Sequential()
model.add(layers.Dense(300, activation='relu', input_shape=(features,)))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer= 'adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [6]:
```
es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 10, batch_size=100,
                    validation_data=(X_dev_counts, labels_dev), verbose = 0, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

Out[6]:
```
0.7451
```

In [7]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Saatiin tähän asti paras validointitarkkuus 74,5%, mutta validointitarkkuus laskee jo heti ensimmäisen kierroksen jälkeen. Koitetaan pienentää oppimisnopeutta kymmenesosaan.

Ensemble-menetelmät perustuvat teoriaan, että kun yhdistetään eri luokittelijoita, joista jokainen ennustaa paremmin kuin satunnainen arvaaminen, satunnaiset virheet kumoavat toisensa ja oikeat päätökset vahvistuvat. Yksi tapa luoda erilaisia luokittelijoita, on käyttää luokittelijoiden kouluttamiseen erilaisia otoksia koulutusdatasta. Neuroverkkojen tapauksessa käytännöllinen tapa muodostaa lukuisia erilaisia aliverkkoja on lisätä dropout-kerros, joka maskaa valitun osuuden kerroksen solmuista pois kertomalla ulostuloarvon nollalla. Lisätään myös tällainen dropout-kerros malliin.

Opetusajat alkavat olla jo niin pitkiä, että lisään myös opetuksen edistymisenseurannan näkyviin. Tämä tehdään asettamalla parametri verbose = 1.

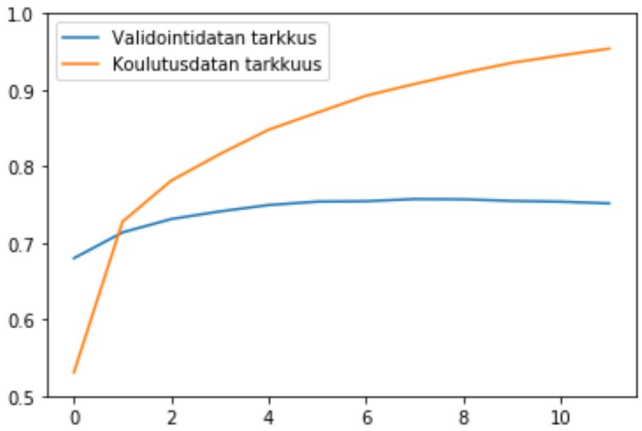
In [13]:
```
features = X_train_counts.shape[1]
model = models.Sequential()
model.add(layers.Dense(300, activation='relu', input_shape=(features,)))
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(10, activation='softmax'))
adam = optimizers.Adam(lr=0.0001)
model.compile(optimizer= adam,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [14]:
```
es = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(X_train_counts, labels_train, epochs= 30, batch_size=100,
                    validation_data=(X_dev_counts, labels_dev), verbose = 1, callba
cks=[es])

_, val_acc = model.evaluate(X_dev_counts, labels_dev, verbose=0)
val_acc
```

```
Train on 100000 samples, validate on 10000 samples
Epoch 1/30
100000/100000 [==============================] - 952s 10ms/step - loss: 2.1428 -
acc: 0.5306 - val_loss: 1.8973 - val_acc: 0.6800
Epoch 2/30
100000/100000 [==============================] - 970s 10ms/step - loss: 1.5832 -
acc: 0.7279 - val_loss: 1.3889 - val_acc: 0.7138
Epoch 3/30
100000/100000 [==============================] - 947s 9ms/step - loss: 1.1194 -
acc: 0.7814 - val_loss: 1.1109 - val_acc: 0.7312
Epoch 4/30
100000/100000 [==============================] - 956s 10ms/step - loss: 0.8527 -
acc: 0.8162 - val_loss: 0.9671 - val_acc: 0.7412
Epoch 5/30
100000/100000 [==============================] - 947s 9ms/step - loss: 0.6842 -
acc: 0.8479 - val_loss: 0.8854 - val_acc: 0.7495
Epoch 6/30
100000/100000 [==============================] - 948s 9ms/step - loss: 0.5675 -
acc: 0.8703 - val_loss: 0.8364 - val_acc: 0.7540
Epoch 7/30
100000/100000 [==============================] - 947s 9ms/step - loss: 0.4763 -
acc: 0.8924 - val_loss: 0.8062 - val_acc: 0.7545
Epoch 8/30
100000/100000 [==============================] - 947s 9ms/step - loss: 0.4037 -
acc: 0.9078 - val_loss: 0.7872 - val_acc: 0.7573
Epoch 9/30
100000/100000 [==============================] - 947s 9ms/step - loss: 0.3438 -
acc: 0.9223 - val_loss: 0.7779 - val_acc: 0.7570
Epoch 10/30
100000/100000 [==============================] - 951s 10ms/step - loss: 0.2926 -
acc: 0.9354 - val_loss: 0.7754 - val_acc: 0.7549
Epoch 11/30
100000/100000 [==============================] - 958s 10ms/step - loss: 0.2507 -
acc: 0.9450 - val_loss: 0.7773 - val_acc: 0.7539
Epoch 12/30
100000/100000 [==============================] - 956s 10ms/step - loss: 0.2150 -
acc: 0.9540 - val_loss: 0.7828 - val_acc: 0.7517
```

Out[14]: 0.7549

In [15]:
```
plt.ylim(0.5,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Saimme parhaan tuloksen 75,5%. Arvioidaan vielä parhaan mallin luotettavuus testidatalla.

In [28]:
```
_, val_acc = model.evaluate(X_test_counts, labels_test, verbose=1)
val_acc
```

```
10000/10000 [==============================] - 30s 3ms/step
```

Out[28]:
```
0.736
```

Testidatalla tarkkuudeksi saatiin 73,6%.

4.5 Neuroverkko käyttäen sanaupotuksia (word embeddings)

Sanaupotukset ovat sanan merkitystä kuvaavia reaalilukuarvoisia vektoreita. Perusideana on, että sanan merkitys voidaan määrittää sen perusteella, minkä muiden sanojen yhteydessä se esiintyy. Näin semanttisesti toisiaan lähellä olevat sanat sijoittuvat myös vektoriavaruudessa lähelle toisiaan.

Sanaupotuksia on mahdollista opettaa valmiilla neuroverkkomalleilla omasta aineistosta. Tällaisia työkaluja ovat ainakin Googlen kehittämä word2vec, Facebookin kehittämä fastText ja Stanford Universityn kehittämä GloVe. Tämä vaatii kuitenkin suuren datamäärän sekä runsaasti laskentakapasiteettia. Siksi usein käytetäänkin valmiiksi opetettuja (jollakin kattavalla aineistolla) sanaupotuksia.

Suomenkielessä sanoilla on useita taivutusmuotoja ja sanajohdoksia. FastText sopii hyvin suomenkielisten sanojen vektoriesitysten muodostukseen, koska se hyödyntää mallissaan myös sanojen alimerkkijonoja.

Osoitteesta https://fasttext.cc/docs/en/crawl-vectors.html (https://fasttext.cc/docs/en/crawl-vectors.html) voi ladata fastTextin valmiit sanaupotusvektorit 157 kielelle. Olen tallentanut sieltä koneelleni suomenkielisten sanojen vektoriesitykset tekstimuodossa. Ladataan tämä sanaupotusmalliksi.

In [7]:
```
embmodel=KeyedVectors.load_word2vec_format("cc.fi.300.vec.gz", binary=False)
```

In [8]:
```
embmodel.vectors.shape
```

Out[8]:
```
(2000000, 300)
```

Mallissa on yhteensä kaksi miljoonaa sanaa. Jokainen matriisin rivi edustaa yhtä sanaa. Sarakkeiden lukumäärästä nähdään, että sanat esitetään 300-dimensioisen avaruuden vektoreina.

Tutkitaan hieman mallin ominaisuuksia. Sanojen läheisyys lasketaan käyttäen kosini samankaltaisuutta. Suure saa suurimman arvonsa 1, kun vektoreiden välinen kulma on 0, ja pienimmän arvon 0, kun vektorit ovat ortogonaalisia suhteessa toisiinsa.

In [9]:
```
print("Lähimpänä sanaa 'peli' löytyvät sanat:")
print(embmodel.most_similar("peli",topn=10))
print()

print("Sanojen 'jalkapallo' ja 'jääkiekko' samanlaisuus:")
print(embmodel.similarity("jalkapallo", "jääkiekko"))
print("Sanojen 'jalkapallo' ja 'jää' samanlaisuus:")
print(embmodel.similarity("jalkapallo", "jää"))
```

```
Lähimpänä sanaa 'peli' löytyvät sanat:
[('15-peli', 0.7574991583824158), ('ohipeli', 0.7506250143051147), ('DS-peli',
0.7480830550193787), ('peli.', 0.7412402629852295), ('peruspeli', 0.738192915916
4429), ('PSP-peli', 0.7369946241378784), ('GP-peli', 0.7369784116744995), ('perh
epeli', 0.7339655160903931), ('puzzlepeli', 0.7338589429855347), ('puzzle-peli',
0.7330989241600037)]

Sanojen 'jalkapallo' ja 'jääkiekko' samanlaisuus:
0.71136767
Sanojen 'jalkapallo' ja 'jää' samanlaisuus:
0.1774352
```

Jotta nämä valmiit sanaupotusvektorit pystyttäisiin yhdistämään alkuperäisten tekstidokumenttiemme sanoihin, meidän pitää vektorisoida alkuperäiset tekstidokumentit uudella tavalla. Ensin koko korpuksen sanasto indeksoidaan. Sitten jokaisesta dokumentista muodostetaan lista kokonaislukuja niin, että jokainen dokumentin sana korvataan kyseistä sanaa vastaavalla indeksillä. Tämä voidaan tehdä helposti käyttämällä Kerasin Tokenizer-apuohjelmaluokkaa. Tietokoneeni rajallisen muistikapasiteetin vuoksi otetaan malliin mukaan vain 100 000 yleisintä sanaa.

In [29]:
```
tokenizer = Tokenizer(num_words=100000)
tokenizer.fit_on_texts(X_train)

word_index = tokenizer.word_index

Xseq_train = tokenizer.texts_to_sequences(X_train)
Xseq_dev = tokenizer.texts_to_sequences(X_dev)
Xseq_test = tokenizer.texts_to_sequences(X_test)
```

In [30]:
```
print(X_train[1000])
print(Xseq_train[1000])
```

```
Autohifikritiikki perustuu siihen , että äänentoiston asentajat eivät ymmärrä pu
htaan äänen päälle , vaan rakentavat epätasapainoisia järjestelmiä . Ylilyödyt b
assot ovat yleisimpiä ongelmia ja vain harva tekee äänentoiston kunnolla . Toine
n tekijä on siinä , että monessa autossa perussetti on tarpeeksi pätevä . Kun au
to pitää kovaa meteliä , niin laadukkaaseen äänentoistoon turha panostaa ellei p
anosta kunnon äänieristykseen .
[1070, 87, 4, 37809, 47099, 73, 354, 10786, 7593, 244, 26, 13196, 14669, 78199,
28, 31656, 521, 1, 18, 1907, 182, 37809, 492, 205, 2151, 2, 61, 4, 2911, 1420, 3
7810, 2, 331, 3808, 8, 269, 65, 935, 11681, 6, 47100, 519, 2779, 448, 7871, 381]
```

Yllä nähdään, miten ensimmäinen viesti on muutettu niitä vastaavien sanaindeksien listaksi.

Koska dokumenttien sisältämä sanamäärä vaihtelee, myös niistä muodostettujen sanaindeksilistojen pituus vaihtelee. Tokenizer aloittaa sanaston indeksoinnin luvusta 1. Indeksi 0 ei siis vastaa mitään sanaa, joten sitä voidaan käyttää "listan täytteenä" (padding), kun kaikista dokumenteista halutaan muodostaa keskenään saman pituisia listoja.

Muodostetaan nyt kaikista dokumenteista (viesteistä) saman pituiset sanaindeksilistat lisäämällä loppuun vaadittava määrä indeksejä 0. tehdään tämä käyttäen Kerasin pad_sequences-luokkaa.

In [31]:
```
Xpad_train = pad_sequences(Xseq_train, padding='post')
print(Xpad_train[0])
max_words = len(Xpad_train[0])
print(max_words)
```

```
[62914 77475   148 ...     0     0     0]
1307
```

Tehdään validointi- ja testiaineistojen vektoreista saman pituisia kuin koulutusaineiston pisin viesti (1307 sanaa). Tallennetaan vektorit myöhempää käyttöä varten.

In [32]:
```
Xpad_dev = pad_sequences(Xseq_dev, maxlen=max_words, padding='post')
Xpad_test = pad_sequences(Xseq_test, maxlen=max_words, padding='post')
print(len(Xpad_dev[0]))
```

```
1307
```

In [33]:
```
np.save('Xpad_train.npy', Xpad_train)
np.save('Xpad_dev.npy', Xpad_dev)
np.save('Xpad_test.npy', Xpad_test)
```

Seuraavaksi muodostetaan matriisi, joka yhdistää viesteistä muodostetun sanaston sanat upotussanavektoreihin. Eli matriisin rivillä 1 on viesteistä muodostetun sanaston indeksiä 1 vastaavan sanan 'ja' 300-dimensioinen upotusvektori. Jos sanalle ei löydy upotusvektoria, sitä vastaavan vektorin kaikki luvut jäävät matriisissa arvoon 0.

In [15]:
```
embmatrix =  np.zeros((100001, 300))
count = 0
for word, i in word_index.items():
    if i < 100001:
        if word in embmodel.vocab:
            embmatrix[i] = embmodel.get_vector(word)
            count += 1
```

In [20]:
```
np.save('embmatrix.npy', embmatrix)
```

In [16]:
```
embmatrix[1]
```

Out[16]:
```
array([ 6.85999990e-02,  7.13000000e-02, -3.77999991e-02, -1.22199997e-01,
       -4.28000018e-02, -1.84699997e-01,  2.77999993e-02, -4.28999998e-02,
        4.50500011e-01, -8.00000038e-03,  4.74999994e-02,  6.40000030e-02,
        6.49999976e-02, -9.53999981e-02, -2.07000002e-02,  1.76999997e-02,
        7.64999986e-02,  2.43999995e-02, -2.25000009e-02,  5.75999990e-02,
        2.14300007e-01,  3.68000008e-02, -7.81000033e-02, -3.77999991e-02,
        6.43000007e-02, -7.10000005e-03,  6.58000037e-02, -8.10000002e-02,
       -1.70800000e-01, -3.06000002e-02, -1.43700004e-01, -7.53000006e-02,
       -4.17999998e-02,  5.57000004e-02, -1.84200004e-01,  6.17000014e-02,
       -8.16000029e-02, -4.49999981e-03, -5.40999994e-02,  1.02000004e-02,
       -3.99999990e-04,  1.41700000e-01, -1.70000002e-03, -3.05000003e-02,
        3.84999998e-02, -5.71999997e-02, -1.47000002e-02, -5.90000022e-03,
       -4.34999987e-02, -3.75000015e-02,  2.19200000e-01,  1.53000001e-02,
        1.64999999e-02,  3.46999988e-02,  1.30000000e-03,  2.38000005e-02,
        1.43999998e-02, -1.33999996e-02,  2.92000007e-02,  2.17000004e-02,
       -5.20000001e-03,  2.19000001e-02, -1.48800001e-01, -6.27999976e-02,
        4.45999987e-02, -7.11999983e-02, -9.60000046e-03, -3.94000001e-02,
        4.10000002e-03,  1.99999995e-04, -1.87500000e-01, -5.60999997e-02,
       -2.00000009e-03, -4.52999994e-02,  5.33999987e-02, -7.49999983e-03,
       -6.36999980e-02,  9.49999969e-03, -4.69000004e-02, -8.34999979e-02,
       -7.55999982e-02, -3.83000001e-02, -1.42399997e-01, -1.57800004e-01,
       -5.37000000e-02,  2.74000000e-02,  4.93000001e-02,  5.53000011e-02,
        1.16999997e-02,  2.41999999e-02,  7.73999989e-02, -9.80000012e-03,
       -2.37700000e-01,  1.09999999e-03, -3.33000012e-02,  2.26000007e-02,
       -6.01999983e-02,  4.28000018e-02, -9.89999995e-02, -2.00000009e-03,
       -4.03000005e-02,  9.91000012e-02, -5.70000010e-03, -1.54999997e-02,
        3.29000019e-02,  6.08999990e-02, -2.95000002e-02,  6.80000009e-03,
       -5.73000014e-02,  6.94999993e-02, -5.31000011e-02,  2.00999994e-02,
        2.99999993e-02,  8.51000026e-02,  1.81000009e-02,  9.51000005e-02,
        5.81000000e-02,  2.98999995e-02, -1.18799999e-01,  8.57999995e-02,
       -3.86999995e-02,  7.59000033e-02, -2.76999995e-02,  1.03299998e-01,
       -9.39999986e-03,  1.35000004e-02,  7.72000030e-02, -1.92000002e-01,
        3.57999988e-02,  1.30999997e-01,  2.33999994e-02,  1.15699999e-01,
       -9.54999998e-02,  1.56999994e-02, -3.42000015e-02,  2.74000000e-02,
        5.95000014e-02, -1.26000002e-01, -4.60000010e-03,  6.43000007e-02,
       -6.81999996e-02,  3.09999995e-02,  8.29999987e-03, -1.00800000e-01,
        2.95000002e-02,  6.00000005e-03, -4.03000005e-02,  1.36000002e-02,
        3.24999988e-02,  1.00500003e-01,  4.76999991e-02, -8.00000038e-03,
        1.97999999e-02,  1.47000002e-02,  3.22000012e-02,  1.24000004e-02,
       -2.62899995e-01, -4.49000001e-02, -2.09999993e-03, -2.80000009e-02,
       -1.11400001e-01,  2.87699997e-01,  3.70000005e-02,  1.16400003e-01,
        7.79999979e-03, -7.72000030e-02,  3.62000018e-02, -9.42000002e-02,
       -6.75000027e-02, -1.46300003e-01, -8.99999961e-03, -1.65900007e-01,
       -3.99000011e-02,  6.69999979e-03, -1.25699997e-01,  7.10000023e-02,
       -3.64999995e-02, -5.07999994e-02, -4.12999988e-02,  5.90000022e-03,
       -6.37999997e-02, -6.39999984e-03,  7.72000030e-02, -2.49999994e-03,
        1.15099996e-01, -2.60000005e-02, -3.09999995e-02,  8.99999985e-04,
       -3.88999991e-02,  9.04999971e-02, -3.20999995e-02,  4.52999994e-02,
       -1.25200003e-01, -1.73099995e-01,  3.80000006e-03, -3.07000000e-02,
       -7.05000013e-02,  4.98999991e-02, -5.37000000e-02,  3.42999995e-02,
        2.67699987e-01,  6.30000001e-03,  6.10999987e-02, -5.22000007e-02,
        2.32999995e-02, -4.21999991e-02,  6.99999975e-04,  3.26999985e-02,
        6.62999973e-02, -2.30000000e-02,  6.27999976e-02,  1.09999999e-03,
       -8.25999975e-02, -1.09999999e-02, -1.55999996e-02,  6.52000010e-02,
        1.89600006e-01, -2.66999993e-02, -9.82000008e-02, -7.88000003e-02,
        5.86000010e-02, -5.77000007e-02, -1.99999996e-02, -9.09999982e-02,
        2.34999992e-02,  5.42999990e-02,  1.23300001e-01, -1.21999998e-02,
       -8.89999978e-03,  5.60999997e-02,  9.08999965e-02,  4.36000004e-02,
        4.58999984e-02, -2.37000007e-02,  2.84000002e-02, -8.99000019e-02,
       -7.59000033e-02, -6.00000005e-03, -3.07999998e-02, -3.64999995e-02,
        1.24000004e-02,  3.20000015e-02,  2.51000002e-02, -8.50000046e-03,
       -5.90999983e-02,  5.18000014e-02, -5.46000004e-02, -3.20999995e-02,
       -5.79999993e-03, -2.18000002e-02,  6.53000027e-02, -3.66999991e-02,
       -2.30999999e-02, -8.15000013e-02,  2.80000009e-02, -1.83500007e-01,
```

In [19]:
```
print('Upotusvektori löytyi ',count, ' sanalle.')
```

```
Upotusvektori löytyi  86570  sanalle.
```

Nyt olemme valmiita muodostamaan upotusvektoreita hyödyntävän neuroverkon. Embedding-kerroksen 2D-tulosta ei pysty suoraan yhdistämään Dense-kerrokseen, joka käsittelee 1D-syötteitä. Tämän vuoksi Enbedded-kerroksen tulos täytyy muuntaa 1D:ksi joko litistämällä tulos Flatten-kerroksella tai käyttämällä GlobalMaxPool1D- tai GlobalAveragePool1D-kerroksia (palauttavat litistettävän dimension kaikkien piirteiden maksimin tai keskiarvon).

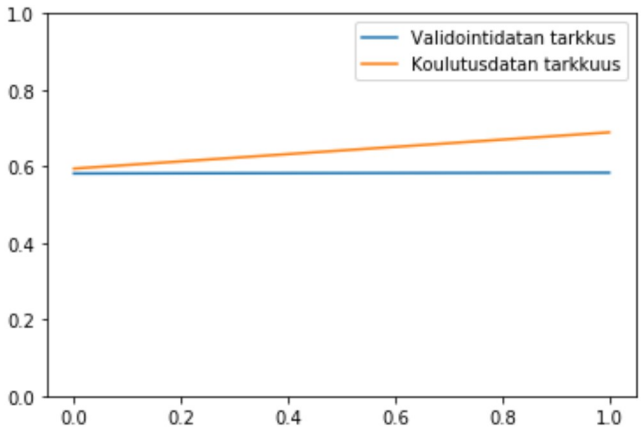
In [9]:
```
model = models.Sequential()
model.add(layers.Embedding(input_dim=100001,
                           output_dim=300,
                           weights = [embmatrix],
                           input_length = 1307,
                           trainable=False ))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 1307, 300)         30000300  
_________________________________________________________________
flatten_1 (Flatten)          (None, 392100)            0         
_________________________________________________________________
dense_3 (Dense)              (None, 64)                25094464  
_________________________________________________________________
dense_4 (Dense)              (None, 10)                650       
=================================================================
Total params: 55,095,414
Trainable params: 25,095,114
Non-trainable params: 30,000,300
_________________________________________________________________
```

In [11]:
```
es = EarlyStopping(monitor='val_loss', min_delta=0, patience=1, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(Xpad_train, labels_train, epochs= 10, batch_size=100,
                    validation_data=(Xpad_dev, labels_dev), verbose = 0, callbacks=
[es])

_, val_acc = model.evaluate(Xpad_dev, labels_dev, verbose=0)
val_acc
```

Out[11]:
```
0.5815
```

In [12]:
```
plt.ylim(0,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Tuli harmillisen huono tulos. Lisätään malliin konvoluutiokerros embedding-kerroksen jälkeen ja korvataan Flatten-kerros GlobalMaxPool1D-kerroksella. Lisätään myös dropout-kerros ja kasvatetaan Early stopping-menetelmän patience-parametrin arvo viiteen, jolloin neuroverkon kouluttaminen loppuu vasta, kun validointitulos ei ole parantunut viimeisen viiden kierroksen aikana.

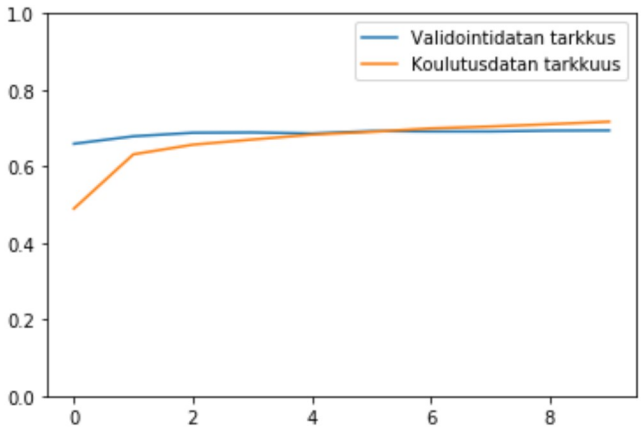
In [17]:
```
model = models.Sequential()
model.add(layers.Embedding(input_dim=100001,
                           output_dim=300,
                           weights = [embmatrix],
                           input_length = 1307,
                           trainable=False ))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 1307, 300)         30000300
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1303, 128)         192128
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_6 (Dense)              (None, 10)                650
=================================================================
Total params: 30,201,334
Trainable params: 201,034
Non-trainable params: 30,000,300
_________________________________________________________________
```

In [18]:
```
es = EarlyStopping(monitor='val_loss', min_delta=0, patience=5, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(Xpad_train, labels_train, epochs= 10, batch_size=100,
                    validation_data=(Xpad_dev, labels_dev), verbose = 0, callbacks=
[es])

_, val_acc = model.evaluate(Xpad_dev, labels_dev, verbose=0)
val_acc
```

Out[18]: 0.6941

In [19]:
```
plt.ylim(0,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkuus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Tämän mallin koulutusajo kestikin kotikoneella jo yli 12 tuntia. Tarkkuudeksi saatiin validointidatalla 69.4%. Pienemmällä oppimisnopeusparametrin arvolla saatettaisiin päästä parempaan tulokseen. Koitetaan siis samaa mallia pienentämällä oppimisnopeus kymmenesosaan (lr=0.0001).

In [4]:
```
model = models.Sequential()
model.add(layers.Embedding(input_dim=100001,
                           output_dim=300,
                           weights = [embmatrix],
                           input_length = 1307,
                           trainable=False ))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
adam = optimizers.Adam(lr=0.0001)
model.compile(optimizer=adam,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

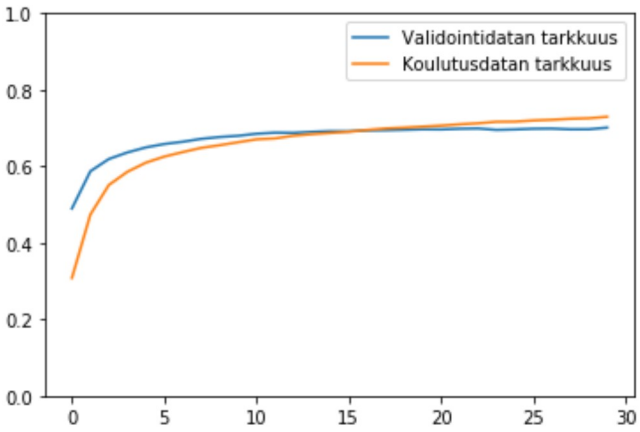
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1307, 300)         30000300
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1303, 128)         192128
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 30,201,334
Trainable params: 201,034
Non-trainable params: 30,000,300
_________________________________________________________________
```

In [6]:
```
es = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(Xpad_train, labels_train, epochs= 30, batch_size=100,
                    validation_data=(Xpad_dev, labels_dev), verbose = 1, callbacks=
[es])

_, val_acc = model.evaluate(Xpad_dev, labels_dev, verbose=0)
val_acc
```

```
Train on 100000 samples, validate on 10000 samples
Epoch 1/30
100000/100000 [==============================] - 4609s 46ms/step - loss: 2.0552
- acc: 0.3080 - val_loss: 1.7739 - val_acc: 0.4897
Epoch 2/30
100000/100000 [==============================] - 4451s 45ms/step - loss: 1.6676
- acc: 0.4739 - val_loss: 1.4046 - val_acc: 0.5871
Epoch 3/30
100000/100000 [==============================] - 4449s 44ms/step - loss: 1.4290
- acc: 0.5512 - val_loss: 1.2471 - val_acc: 0.6191
Epoch 4/30
100000/100000 [==============================] - 4448s 44ms/step - loss: 1.3136
- acc: 0.5855 - val_loss: 1.1690 - val_acc: 0.6362
Epoch 5/30
100000/100000 [==============================] - 4444s 44ms/step - loss: 1.2438
- acc: 0.6094 - val_loss: 1.1196 - val_acc: 0.6494
Epoch 6/30
100000/100000 [==============================] - 4450s 45ms/step - loss: 1.1902
- acc: 0.6253 - val_loss: 1.0861 - val_acc: 0.6583
Epoch 7/30
100000/100000 [==============================] - 4468s 45ms/step - loss: 1.1529
- acc: 0.6372 - val_loss: 1.0626 - val_acc: 0.6642
Epoch 8/30
100000/100000 [==============================] - 4447s 44ms/step - loss: 1.1228
- acc: 0.6482 - val_loss: 1.0422 - val_acc: 0.6718
Epoch 9/30
100000/100000 [==============================] - 4441s 44ms/step - loss: 1.0964
- acc: 0.6552 - val_loss: 1.0265 - val_acc: 0.6766
Epoch 10/30
100000/100000 [==============================] - 4441s 44ms/step - loss: 1.0757
- acc: 0.6630 - val_loss: 1.0153 - val_acc: 0.6798
Epoch 11/30
100000/100000 [==============================] - 4452s 45ms/step - loss: 1.0530
- acc: 0.6706 - val_loss: 1.0053 - val_acc: 0.6852
Epoch 12/30
100000/100000 [==============================] - 4473s 45ms/step - loss: 1.0392
- acc: 0.6730 - val_loss: 0.9969 - val_acc: 0.6880
Epoch 13/30
100000/100000 [==============================] - 4476s 45ms/step - loss: 1.0211
- acc: 0.6799 - val_loss: 0.9899 - val_acc: 0.6875
Epoch 14/30
100000/100000 [==============================] - 4451s 45ms/step - loss: 1.0053
- acc: 0.6841 - val_loss: 0.9839 - val_acc: 0.6901
Epoch 15/30
100000/100000 [==============================] - 4496s 45ms/step - loss: 0.9924
- acc: 0.6873 - val_loss: 0.9779 - val_acc: 0.6919
Epoch 16/30
100000/100000 [==============================] - 4446s 44ms/step - loss: 0.9798
- acc: 0.6904 - val_loss: 0.9742 - val_acc: 0.6916
Epoch 17/30
100000/100000 [==============================] - 4442s 44ms/step - loss: 0.9697
- acc: 0.6951 - val_loss: 0.9700 - val_acc: 0.6939
Epoch 18/30
100000/100000 [==============================] - 4455s 45ms/step - loss: 0.9563
- acc: 0.6990 - val_loss: 0.9663 - val_acc: 0.6943
Epoch 19/30
100000/100000 [==============================] - 4468s 45ms/step - loss: 0.9510
- acc: 0.7012 - val_loss: 0.9643 - val_acc: 0.6955
Epoch 20/30
100000/100000 [==============================] - 4443s 44ms/step - loss: 0.9390
- acc: 0.7038 - val_loss: 0.9606 - val_acc: 0.6968
Epoch 21/30
100000/100000 [==============================] - 4445s 44ms/step - loss: 0.9288
- acc: 0.7064 - val_loss: 0.9587 - val_acc: 0.6965
```

Out[6]: 0.7012

In [7]:
```
plt.ylim(0,1.0)
plt.plot(history.history["val_acc"],label="Validointidatan tarkkuus")
plt.plot(history.history["acc"],label="Koulutusdatan tarkkuus")
plt.legend()
plt.show()
```



Paras validointitulos 70,1% saatiin viimeisellä kierroksella, joten tulos olisi voinut vielä hiukan parantua, mikäli koulutusta olisi jatkettu kauemmin. Mallin koulutus kesti näinkin yli 36 tuntia, joten neuroverkon hienosäätö tavallisella koneella alkaa käydä mahdottomaksi.

In [8]:
```
_, val_acc = model.evaluate(Xpad_test, labels_test, verbose=0)
val_acc
```

Out[8]: 0.6854

Testidatalla tarkkuudeksi saatiin 68,5%. Tulos jäi huonommaksi kuin muilla menetelmillä. Sanaupotusvektoreita oli koneeni muistirajoitteiden vuoksi käytössä vain 86570:lle sanalle, kun koko sanaston koko olisi ollut lähes 400 000. Tämä on todennäköisin syy sille, ettei sanaupotusvektoreita hyödyntämällä päästy parempaan tulokseen. Ilman sanaupotuksia koulutettu neuroverkko koulutettiin 100 000 sanalla ja sillä päästiin jo merkittävästi parempaan tarkkuuteen 73,6%.

4.6 Sanaupotuksia hyödyntävä konvoluutioneuroverkko CSC:n supertietokoneella

Kotikoneen kapasiteettirajoitteiden vuoksi sanaupotuksia hyödyntävien neuroverkkojen luokittelutarkkuus jäi edellä varsin pieneksi. Katsotaan, millaisiin tuloksiin pääsemme CSC:n supertietokoneen Puhtin avulla, kun malliin voidaan ottaa mukaan kaikki koulutusdatan sanat ja hyperparametrien optimointikin on mahdollista järkevässä ajassa.

Muodostetaan uudelleen matriisi, joka yhdistää koulutusaineiston viesteistä muodostetun sanaston sanat upotussanavektoreihin. Tällä kertaa upotussanamatriisiin otetaan mukaan kaikki sanat.

In [5]:
```
embmodel=KeyedVectors.load_word2vec_format("cc.fi.300.vec.gz", binary=False)
```

In [6]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

word_index = tokenizer.word_index

Xseq_train = tokenizer.texts_to_sequences(X_train)
Xseq_dev = tokenizer.texts_to_sequences(X_dev)
Xseq_test = tokenizer.texts_to_sequences(X_test)
```

In [7]:
```
embmatrix_all =  np.zeros((len(word_index) + 1, 300))
count = 0
for word, i in word_index.items():
    if word in embmodel.vocab:
        embmatrix_all[i] = embmodel.get_vector(word)
        count += 1
```

In [10]:
```
np.save('embmatrix_all.npy', embmatrix_all)
```

In [11]:
```
print('Koulutusdatassa on',len(word_index), 'erilaista sanaa. ''Upotusvektori löyty
i', count, 'sanalle eli',
     '%.2f' % (100*count/len(word_index)), 'prosentille.')
```

```
Koulutusdatassa on 399349 erilaista sanaa. Upotusvektori löytyi 212960 sanalle e
li 53.33 prosentille.
```

Tämä on huomattavasti enemmän kuin aikaisemmassa matriisissa, jossa upotusvektorit löytyi vain 86570:lle sanalle.

Jos omat käyttöoikeudet on olemassa, Puhtiin saa SSH-yhteyden Windows-koneella esimerkiksi käyttämällä PuTTYä (https://putty.org/ (https://putty.org/)). Tiedostojen siirtelyyn oman koneen ja Puhtin välillä käytin FileZillaa (https://filezilla-project.org/ (https://filezilla-project.org/)). CSC:n kattavat Linux-ohjeet löytyvät täältä: https://research.csc.fi/csc-guide-linux-basics-for-csc (https://research.csc.fi/csc-guide-linux-basics-for-csc).

Puhti käyttää SLURM-ajonhallintajärjestelmää. Työn saamiseksi ajoon, tarvitaan määrämuotoinen .sh-päätteinen eräajotiedosto, joka lähetetään töiden aikatauluttajalle komennolla sbatch. Eräajotiedoston voi kirjoittaa tavallisella tekstieditorilla (käytin tässä Nanoa).

Eräajotiedoston sisältö:

In [ ]:
```
#!/bin/bash
#SBATCH --job-name=testi
#SBATCH --account=Project_XXX
#SBATCH --partition=gpu
#SBATCH --time=01:00:00
#SBATCH --mem-per-cpu=2G
#SBATCH --gres=gpu:v100:1

module purge
module load tensorflow
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

python CNN.py
```

Ajetaan supertietokoneella seuraavaa koodia (yllä tiedosto CNN.py) vastaava osuus erilaisilla ohjausparametrien arvoilla:

In [ ]:

```
max_words = len(Xpad_train[0])
different_words = embmatrix_all.shape[0]
dim = embmatrix_all.shape[1]

model = models.Sequential()
model.add(layers.Embedding(input_dim=different_words,
                           output_dim=dim,
                           weights = [embmatrix_all],
                           input_length = max_words,
                           trainable=False ))
model.add(layers.Conv1D(filters, kernel_size, activation='relu'))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(dim1, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
adam = optimizers.Adam(lr=lr)
model.compile(optimizer=adam,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

es = EarlyStopping(monitor='val_loss', min_delta=0, patience= patience, verbose=0,
                   mode='auto', baseline=None, restore_best_weights=True)

history = model.fit(Xpad_train, labels_train, epochs= epochs, batch_size= batch_siz
e,
                    validation_data=(Xpad_dev, labels_dev), verbose = 0, callbacks=
[es])

_, val_acc = model.evaluate(Xpad_dev, labels_dev, verbose=0)
print('Tarkkuus validointidatalla', val_acc)

_, val_acc = model.evaluate(Xpad_test, labels_test, verbose=0)
print('Tarkkuus testidatalla', val_acc)
```

- Ajetaan ensin samoilla ohjausparametreilla kuin viimeisin paras tulos: filters = 128, kernel_size = 5, lr = 0.0001, patience = 2, epochs = 100, batch_size = 100, dim1 = 64. Tällä kertaa mallissa on siis mukana kaikki sanat, joille löytyi upotusvektorit.
  - Tulos: tarkkuus validointidatalla 72,4%, tarkkuus testidatalla 71,1%.
- Sama ilman ensimmäistä Dense-kerrosta:
  - Tulos: tarkkuus validointidatalla 72,5%, tarkkuus testidatalla 71,5%.

    Tulos parani, joten jatketaan tämän yksinkertaisemman mallin optimoimista.
- Edellinen, mutta trainable-arvolla 'True' eli nyt muulla aineistolla esikoulutettuja upotussanavektoreita edelleen koulutetaan omalla koulutusaineistollamme:
  - Tulos: tarkkuus validointidatalla 75,9%, tarkkuus testidatalla 74,2%.

    Saimme tähän mennessä parhaan tuloksen. Katsotaan, saadaanko tulosta vielä paremmaksi hyperparametreja (filters, kernel_size ja lr) säätämällä.
- Edellinen, kernel_size = 4 ja 3:
  - Tulos, kernel_size = 4: tarkkuus validointidatalla 75,8%, tarkkuus testidatalla 74,1%.
  - Tulos, kernel_size = 3: tarkkuus validointidatalla 75,1%, tarkkuus testidatalla 73,7%.

    Tulokset huononivat kernel_sizeä pienennettäessä.
- Tuplataan filters = 256, kernel_size = 5, 4 ja 3:
  - Tulos, kernel_size = 5: tarkkuus validointidatalla 76,5%, tarkkuus testidatalla 74,2%.
  - Tulos, kernel_size = 4: tarkkuus validointidatalla 76,2%, tarkkuus testidatalla 74,3%.
  - Tulos, kernel_size = 3: tarkkuus validointidatalla 76,2%, tarkkuus testidatalla 74,4%.

    Isommalla filters-arvolla saatiin parempia tuloksia.
- Edellinen (filters = 256, kernel_size = 5, 4 ja 3), puolitetaan lr = 0,00005:
  - Tulos, kernel_size = 5: tarkkuus validointidatalla 76,3%, tarkkuus testidatalla 74,4%.
  - Tulos, kernel_size = 4: tarkkuus validointidatalla 76,0%, tarkkuus testidatalla 74,2%.
  - Tulos, kernel_size = 3: tarkkuus validointidatalla 75,8%, tarkkuus testidatalla 73,9%.

    Oppimisnopeutta säätelevän lr-parametrin puolittaminen ei parantanut tuloksia.

Jotta lopullisen mallin testidata pysyisi puolueettomana, sen perusteella laskettuja tarkkuusarvoja ei voi käyttää parhaan mallin valitsemiseen. Kun paras malli valitaan validointidatan perusteella, paras malli saatiin koulutettua arvoilla: filters = 256, kernel_size = 5, lr = 0.0001, patience = 2, epochs = 100, batch_size = 100. Tämän mallin tarkkuudeksi testidatalla saatiin 74,2%

Näiden 11 neuroverkon opettaminen supertietokoneella kesti yhteensä muutaman tunnin. Yhdenkään näin ison mallin (sanaupotusvektorit yli 200 000 sanalle) opettaminen ei olisi ollut mahdollista kotikoneellani. Suurimman lähinnä näitä vastaavan neuroverkon opettaminen kesti kotikoneella yli 36 tuntia.

5. Yhteenveto eri luokittelumenetelmistä

Olemme edellä kouluttaneet luokittelijoita erilaisilla menetelmillä:

- Naive Bayes: testiaineistolla tarkkuus 71,6%.
- Tukivektorikone: testiaineistolla tarkkuus 72,0%
- Neuroverkko 100000 sanalla: testiaineistolla tarkkuus 73,6%.
- Konvoluutioneuroverkko esikoulutetuilla sanaupotuksilla, 100000 sanalla: testiaineistolla tarkkuus 68,5%.
- Konvoluutioneuroverkko esikoulutetuilla sanaupotuksilla, kaikilla sanoilla, sanaupotusten edelleen koulutuksella: testiaineistolla tarkkuus 74,2%.

Paras tulos saavutettiin käyttäen konvoluutioneuroverkkoa esikoulutetuilla sanaupotuksilla, jossa neuroverkko vielä edelleen koulutti sanaupotuksia. Tämän mallin kouluttaminen ei ollut mahdollista kotikoneellani, joten siihen käytettiin CSC:n supertietokone Puhtia.

Kiitokset

Olen tehnyt tämän työkirjan erikoistyönä osana Turun yliopiston Tekoälyn maisteriopintoja. Lämpimät kiitokset ohjaajalleni apulaisprofessori Sampo Pyysalolle, joka innosti mm. opettelemaan Puhtin käyttöä!

Espoossa 31.12.2019

Marita Risku, marita.h.risku@utu.fi