Tämä selvitys on osa KoDa – Kokonaisvaltainen datan hallinnointi ja hyödyntäminen (ESR 2017- 2019)[1] -hankkeen toimintaa. KoDa on Karelia ammattikorkeakoulun ja Itä-Suomen yliopiston yhteinen hanke, jonka tavoitteena on pienten ja keskisuurten yritysten kasvun tukeminen. KoDa tarjoaa yrityksille koulutusta datan käytöstä liiketoiminnan kehittämiseen sekä pienyrittäjille soveltuvia työkaluja datan hallinnoinnin ja hyödyntämisen tueksi.

[1] https://www.eura2014.fi/rrtiepa/projekti.php?projektikoodi=S20980

Teksti- ja kuvanäkemykset - sentimentit, persoonallisuuspiirteet, kulutusmieltymykset ja tunteet

Mikko Koponen ja Virpi Hotti

KoDa

Tietojenkäsittelytieteen laitos

Tietojenkäsittelytiede

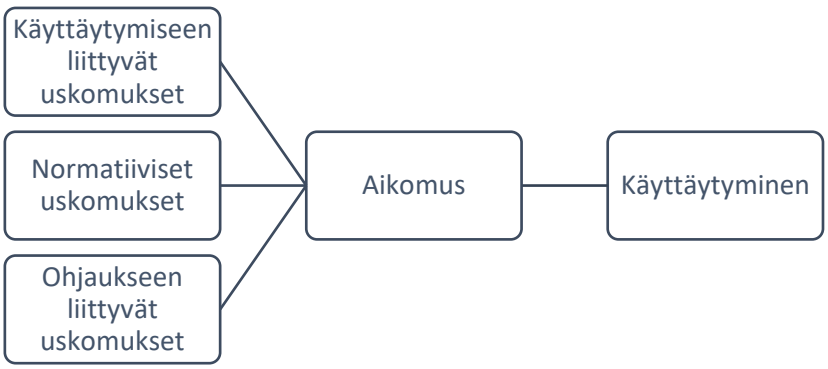
Kesäkuu 2018

Sisällysluettelo

1 Johdanto

Demografisten tietojen (kuten ikä, sukupuoli, rotu, paikka ja työllisyystieto) rinnalle ovat nousseet psykologiset tiedot kuten persoonallisuus (personality), arvot (values), asenteet (attitudes), mielenkiinnon kohteet (interests) ja elämäntyylit (lifestyles). Puhutaan jopa psykologisesta koneesta (psychographic engine) tai hybridiuhkasta[1]. Esimerkiksi Cambridge Analytica luokitteli kunkin yksilön persoonallisuuden niin kutsuttuun "OCEAN"-asteikkoon (Openness, Conscientiousness, Extroversion, Agreeableness, ja Neuroticism) ja sitten muotoiltiin yksilöllisesti kohdennetut viestit, jotka vetosivat jokaisen persoonallisuuteen[2].

Käyttäytymisteoriat selittävät ainakin osittain psykologisen koneen toimivuutta. Meillä on uskomuksia, jotka vaikuttavat aikomuksiimme ja edelleen käyttäytymiseen (Kuva 1). Uskomuksiin vaikuttavat niin sanotut taustatekijät, joissa on psykologisen koneen tarvitsemia asioita kuten persoonallisuuspiirteet. Muita persoonallisuuteen liittyviä asioita ovat arvot, asenteet, tunteet ja älykkyys. Muita taustatekijöitä persoonallisuustekijöiden lisäksi ovat sosiaaliset tekijät (kuten ikä, etnisyys, rotu, sukupuoli, tulot ja uskonto) ja informaatioon liittyvät tekijät (kuten kokemukset ja tietämys).



Kuva 1. Kuva on mukailtu lähteestä Ajzen I (2005) Attitudes, personality and behavior. Open University Press, USA.

[1] https://storage.googleapis.com/upi-live/2018/05/fiia_report55_web_hybrdivaikuttaminen-ja-resilienssi.pdf
[2] https://www.datasciencecentral.com/profiles/blogs/the-real-facebook-controversy

Kun psykologisia tietoja tarkastellaan eri asiayhteyksissä, niin voidaan löytää esimerkiksi seuraavia käyttökohteita[3][4]:

- ”kuvapostituksissa tummempia värejä suosivat kärsivät muita useammin depressiosta ja he postittavat kuvia, joissa on sinertäviä, harmaampia ja tummempia kuvia”
- ”[p]uhegraafianalyysin avulla voidaan puheen kvantifiointia ja graafien muodostamista hyödyntämällä tunnistaa tutkimuksien mukaan 93 % tarkkuudella skitsofreniaan ja maniaan liittyviä piirteitä”
- mielialoja analysoidaan puheen (kuten äänensävy ja korkeus) ja fyysisten tekijöiden (kuten liike, syke, verenpaine ja ihon lämpötila) perusteella

Tässä selvityksessä tarkastellaan viestidataa, jossa on tekstin lisäksi kuvia. Selvityksessä havainnollistetaan, millaisia teksti- ja kuvanäkemyksiä on mahdollista muodostaa. Selvitys auttaa ymmärtämään, millaista taustatietoa teksti- ja kuvanäkemykset tarjoavat ihmisten käyttäytymiselle.

[3] https://www.jyu.fi/it/fi/tutkimus/julkaisut/informaatioteknologia-tiedekunnan-julkaisuja-sarja
[4] https://koulutus.fcg.fi/Portals/2/S03-Neittaanm%C3%A4ki.pdf?ver=2018-05-21-191425-603

2 Datajouko(t)

Futusome-tutkijapalvelusta ladattiin niin sanottu datadumppi, jossa oli 53294 viestiä aiheesta ampumahiihto (Taulukko 1).

Taulukko 1. Datajoukon viestien kirjoittajien lukumäärät kanavittain

Kanava	Kirjoittajien lukumäärä
blog_answer	161
blog_comment	136
blog_post	718
facebook_comment	2918
facebook_event	53
facebook_link	3082
facebook_photo	1356
facebook_post	111
facebook_status	674
facebook_video	299
forum_post	9916
googleplus_post	28
googleplus_post_comment	2
instagram_image	580
instagram_image_comment	102
news_comment	4496
twitter_retweet	4207
twitter_tweet	23935
youtube_video	34
youtube_video_comment	11

Raaka-datajoukkoa käytettiin sellaisenaan sentimenttianalyysissä ja yhteistyötahoetsinnässä.

”Puhdistettu” datajoukko muodostettiin seuraavasti:

- Datasta suodatetaan pois viestit, joiden tyypiksi on määritetty ”retweet”, koska uudelleentweetatut viestit eivät välttämättä kuvaa julkaisijansa persoonallisuutta.
- Lisäksi poistetaan forum_post -tyypisistä keskustelupalstoilta ladatuista viesteistä sellaiset viestit, jotka ovat sisällöltään täsmälleen samanlaisia. Sisällöltään samanlaisia mutta osoitteeltaan eroavia viestejä oli datasetissä huomattava määrä.

3 Sentimenttianalyysi(t)

Luvussa 3.1 käsitellään viestikohtaisten sentimenttien muodostaminen. Luvussa 3.2 käydään läpi menetelmän tarkkuuden testausta kahdella eri datasetillä.Sentimenttien muodostaminen

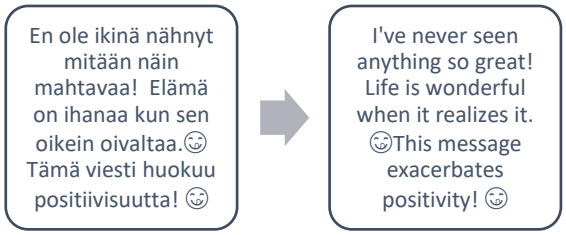
3.1 Sentimenttien muodostaminen

Sentimenttien muodostamisessa käytetty koodi (kolme funktiota, yksi kullekin käytetylle työkalulle) on liitteessä A. Lisäksi havainnollistetaan emojien lisääminen käännettyyn tekstiin (Kuva 2). Sentimenttien muodostamisen päävaiheet ovat seuraavat:

1. Data (viestit, kirjoittajat ja yhteistyökumppanit) ladataan Excel-tiedostosta Python[5] / pandas[6] Dataframeen[7].
2. Viestit käännetään Google Translaten[8] kautta englanniksi googletrans[9]-kirjastolla. Tätä varten viesteistä poistetaan emojit, koska ne aiheuttavat virheitä rajapinnan toiminnassa. Käännetyt viestit tallennetaan Dataframeen.
3. Etsitään emojit kääntämättömistä viesteistä ja pyritään lisäämään ne vastaavaan kohtaan käännettyyn viestiin, koska ne lisäävät[10] käytetyn VADER-sentimenttityökalun tarkkuutta.
4. Käännetyt viestit tokenisoidaan lauseiksi NLTK[11]:n sent_tokenize-funktiolla.
5. Lause-tokenit analysoidaan sentimentin osalta yksitellen Stanford CoreNLP:n sentiment-annotaattorilla[12] sekä NLTK:n VADER[13]-työkalulla.
6. Työkaluilla luodut lausetason sentimentit muunnetaan dokumenttitasolle, eli viestikohtaisiksi sentimenteiksi, ottamalla niistä keskiarvo. Tulokset tallennetaan Dataframeen.
7. Käännetyt viestit tokenisoidaan sanoiksi NTLK:n TweetTokenizer.tokenize() -funktiolla. Tämän jälkeen jokainen viestin sana käydään läpi, pitäen kirjaa viestin kokonaissentimentistä lisäämällä tai vähentämällä kunkin sanan SentiWordNet[14]-sanakirjassa määritetty sentimenttiarvo liukuvasta viestikohtaisesta arvosta. Lopullinen arvo muodostuu liukuvasta arvosta, kun viestin sanat on käyty läpi. Lopullinen arvo tallennetaan Dataframeen.

[5] https://www.python.org/
[6] https://pandas.pydata.org/
[7] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
[8] https://translate.google.fi
[9] http://py-googletrans.readthedocs.io/en/latest/
[10] https://github.com/cjhutto/vaderSentiment - katso ”translating utf8-encoded emojis”
[11] https://www.nltk.org/
[12] https://stanfordnlp.github.io/CoreNLP/sentiment.html
[13] https://github.com/cjhutto/vaderSentiment
[14] http://sentiwordnet.isti.cnr.it/

8. Saatujen työkalukohtaisten sentimenttiarvojen lisäksi lasketaan tuloksista keskiarvo kullekin viestille. Tulos tallennetaan Dataframeen.



Kuva 2. Esimerkki menetelmän mukaisesta viestin kääntämisestä ja emojien lisäämisestä

3.2 Saatujen tulosten testaus ja vertailu

Datan muuntaminen testausta varten on kuvattu luvussa 3.2.1. Toimivuus on testattu (Luku 3.2.2) kahdella datasetillä: Michiganin yliopiston järjestämän sentimenttianalyysi-kilpailun Kaggle-datalla[15], jossa sentimentit on jaoteltu kahteen luokkaan positiivisuuden ja negatiivisuuden perusteella; Stanfordin yliopiston elokuva-arvosteluita sisältävällä datalla[16], joka on luokittelultaan vastaava. Luvussa 3.2.2 on taulukoitu eri datasettien ja menetelmien yhdistelmillä testattaessa saadut tarkkuudet (Accuracy) sekä virhematriisit (Confusion Matrix, Error Matrix[17]).

3.2.1 Datan muuntaminen testausta varten

Työkalulla muodostettu desimaalilukumuotoinen sentimenttidata on muunnettu testausta varten binääriseen muotoon: positiiviset arvot saavat arvon 1, kun taas negatiiviset arvot saavat arvon 0. Tämä muunnos on tehty testausta varten sillä perusteella, että testidatasetit olivat myös binäärisiä. Testauksen yhteydessä kokeiltiin kolmea erilaista tapaa muuntaa data binääriseen muotoon (Kuva 3): seuraavat sentimenttituloksia yhdistelevät menetelmät on testattu kullakin datasetillä erikseen, sekä molemmista dataseteistä yhdistetyllä datasetillä:

1. Määräenemmistöinen äänestys (hard voting)
    a. Vähintään kaksi kolmesta työkalusta äänestää samoin, eli vähintään kaksi positiivista arvoa tarkoittaa positiivista tulosta, ja vastaavasti vähintään kaksi negatiivista arvoa tarkoittaa negatiivista tulosta

[15] https://www.kaggle.com/c/si650winter11
[16] http://ai.stanford.edu/~amaas/data/sentiment/
[17] https://en.wikipedia.org/wiki/Confusion_matrix

2. Keskiarvo (average)
    a. Summataan työkalujen antamat arvot ja jaetaan työkalujen määrällä
3. Painotettu summa (weighted sum)
    a. Kerrotaan kunkin työkalun antama arvo määritetyllä kertoimella, jonka jälkeen summataan saadut arvot. Kertoimet on laskettu siten, että niiden summa on 1, jolloin vaihteluväliksi saadulle painotetulle summalle muodostuu -1 – 1.
    b. Painotukset laadittu kokeellisesti suurimman tarkkuuden saavuttamiseksi. Painotusten määrittämiseen käytettiin satunnaishakua.
    c. Painotukset ovat seuraavat: Vader: 0.45, CoreNLP: 0.35, SentiWordNet: 0.2

Kuva 3. Esimerkki saaduista sentimenttiarvoista ja keskiarvosta

Erilaisilla painotuksilla oli mahdollista päästä suurempaan tarkkuuteen (0.92) Kaggle-datasetin osalta, mutta tämä johti toisessa datasetissä tarkkuuden laskuun. Painotukset on valittu niin, että molempien datasettien osalta tarkkuus olisi mahdollisimman korkea.

3.2.2 Testauksen tulokset

Michiganin yliopiston sentimenttianalyysi-kilpailun Kaggle-data

Testaus suoritettu training.txt -tiedostosta löytyvällä datalla. Tiedostossa on tabulaattorilla eroteltuna riveittäin tekstin polariteetti sekä itse viesti. Data muunnettu ohjelmalle sopivaan Excel-muotoon testausta varten. Eri työkaluilla saadut tarkkuudet löytyvät taulukosta Taulukko 2. Eri yhdistelmämenetelmillä saadut tarkkuudet löytyvät taulukosta Taulukko 3.

Taulukko 2. Yksittäiset työkalut

Vader **Accuracy:** **0.8748193119398671**	**CoreNLP** **Accuracy:** **0.6373229257010696**	**SentiWordNet** **Accuracy:** **0.8119398670135878**

Confusion matrix: **[[2610 365]** **[501 3442]]**	**Confusion matrix:** **[[2793 182]** **[2327 1616]]**	**Confusion matrix:** **[[2478 497]** **[804 3139]]**

Yhdistelmän määritelmät: katso 3.2.1 (Taulukko 3)

Taulukko 3. Yhdistelmä

Äänestys **Accuracy:** **0.8441746169413126** **Confusion matrix:** **[[2790 185]** **[893 3050]]**	**Keskiarvo** **Accuracy:** **0.8866724486845909** **Confusion matrix** **[[2932 43]** **[741 3202]]**	**Painotettu summa** **Accuracy:** **0.8762648164209309** **Confusion matrix:** **[[2782 193]** **[663 3280]]**

Stanford Large Movie Review Dataset

Testaus on suoritettu paketin test-hakemistosta löytyvällä datasetillä, jossa on negatiiviseen ja positiiviseen hakemistoon jaoteltuna 25000 elokuva-arvostelua. Arvostelut olivat alun perin kukin omassa tiedostossaan, jotka muunnettiin ohjelmalle sopivaan Excel-muotoon. Eri työkaluilla saadut tarkkuudet löytyvät taulukosta Taulukko 4. Eri yhdistelmämenetelmillä saadut tarkkuudet löytyvät taulukosta Taulukko 5.

Taulukko 4. Yksittäiset työkalut

Vader **Accuracy: 0.70128** **Confusion matrix:** **[[6760 5740]** **[1728 10772]]**	**CoreNLP** **Accuracy: 0.71856** **Confusion matrix:** **[[12221 279]** **[6757 5743]]**	**SentiWordNet** **Accuracy: 0.62376** **Confusion matrix:** **[[4561 7939]** **[1467 11033]]**

Yhdistelmän määritelmät: katso 3.2.1 (Taulukko 5)

Taulukko 5. Yhdistelmä

Äänestys **Accuracy: 0.72584** **Confusion matrix:** **[[7812 4688]** **[2166 10334]]**	**Keskiarvo** **Accuracy: 0.80348** **Confusion matrix:** **[[10530 1970]** **[2943 9557]]**	**Painotettu summa** **Accuracy: 0.80016** **Confusion matrix:** **[[10274 2226]** **[2770 9730]]**

Alla olevat tulokset on saatu yhdistämällä yllä mainitut kaksi datasettiä ja generoimalla syntyneestä datasetistä työkalulla sentimentit. Eri työkaluilla saadut tarkkuudet löytyvät taulukosta Taulukko 6. Eri yhdistelmämenetelmillä saadut tarkkuudet löytyvät taulukosta Taulukko 7.

Taulukko 6. Yksittäiset työkalut

Vader **Accuracy:** **0.7388934143743342** **Confusion matrix:** **[[9370 6105]** **[2229 14214]]**	**CoreNLP** **Accuracy:** **0.7009524406291121** **Confusion matrix:** **[[15014 461]** **[9084 7359]]**	**SentiWordNet** **Accuracy:** **0.6645466507926562** **Confusion matrix:** **[[7039 8436]** **[2271 14172]]**

Yhdistelmän määritelmät: katso 3.2.1 (Taulukko 5)

Taulukko 7. Yhdistelmä

Äänestys **Accuracy:** **0.7514881884829877** **Confusion matrix:** **[[10602 4873]** **[3059 13384]]**	**Keskiarvo** **Accuracy:** **0.816592518328216** **Confusion matrix:** **[[13315 2160]** **[3694 12749]]**	**Painotettu summa** **Accuracy:** **0.8166551788959208** **Confusion matrix:** **[[13056 2419]** **[3433 13010]]**

Työkalu tuottaa käytetyllä testidatalla lähes poikkeuksetta huomattavasti parempia arvioita, kuin kullakaan yksittäisellä työkalulla yksinään olisi mahdollista tuottaa. Lopullisen yhdistetyn polariteetin muodostamiseen käytetyistä menetelmistä näyttävät painotettu keskiarvo sekä keskiarvo olevan, ainakin valituilla painotuksilla, ennustusvoimaltaan suhteellisen tasaväkiset. Määräenemmistöinen äänestys näyttää suoriutuvan vaihtoehdoista heikoimmin. Tässä valossa on järkevintä käyttää tulosten yhdistämiseen yksinkertaisuuden ja menetelmän yleiskäyttöisyyden nimissä pelkkää keskiarvoa ilman painotuksia. Ohjelma tallentaa Exceliin erillisten työkalujen tuottamien tulosten lisäksi keskiarvon.

4 Yhteistyötahojen etsintä

Sanojen ja muiden ennalta määriteltyjä asioita, kuten yhteistyökumppaneiden nimiä, voidaan hakea viestimassasta (koodi Liitteessä B). Päävaiheet ovat seuraavat:

1. Data (yhteistyökumppanit, viestit) ladataan Excelistä.
2. Luodaan viestidatan Dataframeen sarakkeet jokaiselle yhteistyökumppanille.
3. Alkuperäiset viestit tokenisoidaan sanoiksi Python NLTK:n TweetTokenizerilla.
4. Tokenit stemmataan Pythonin NLTK:n SnowBallStemmerillä. SnowBallStemmeri on tässä tapauksessa käytössä siitä syystä, että se tukee suomenkieltä, ja yhteistyökumppaneiden etsintä tapahtuu sentimenttianalyysistä poiketen kääntämättömästä viestidatasta.
5. Yhteistyökumppanit stemmataan SnowBallStemmerillä. Yhteistyökumppaneiden nimistä on mahdollista sisällyttää useampia variaatioita, ja lisäksi poissulkea samankaltaisia sanoja (kuten Avant -> avanto) virheellisten tulkintojen välttämiseksi.
6. Suoritetaan tokenisoitujen ja stemmattujen yhteistyökumppaneiden nimien haku jokaisesta viestistä. Haku ottaa huomioon moniosaiset yhteistyökumppanin nimet. Moniosaisiin nimiin kelpuutetaan esiintymät, joissa halutun konkordanssietäisyyden päästä ensimmäisestä sanasta löytyy myös toinen sana.
7. Tallennetaan kunkin viestin kohdalle yhteistyökumppanin sarakkeeseen True tai False, riippuen siitä, esiintyykö yhteistyökumppanin nimi viestissä.
8. Tallennetaan lopuksi käännöksillä sekä tuloksilla varustettu Dataframe Excel-tiedostoon. Tiedostoon tallentuvat myös alkuperäisen viestidatan kaikki sarakkeet.

5 Persoonallisuusominaisuudet ja kulutusmieltymykset

IBM Watson ™ Personality Insights[18] -rajapinnan kautta on mahdollista tuottaa analyysejä syötetyn tekstimassan kirjoittajan persoonallisuusominaisuuksista (arvot, persoonallisuuspiirteet ja tarpeet) ja kulutusmieltymyksistä. IBM Watson ™ Personality Insights -rajapinnasta analyysien noutoon ja taulukkomuotoon muuntamiseen käytetty lähdekoodi löytyy liitteestä C. Päävaiheet ovat seuraavat:

1. Data (kirjoittajat, viestit) ladataan excel-tiedostosta.
2. Datasta suodatetaan pois viestit, joiden tyypiksi on määritetty ”retweet”, koska uudelleen tweetatut viestit eivät välttämättä kuvaa julkaisijansa persoonallisuutta. Lisäksi poistetaan forum_post-tyypisistä keskustelupalstoilta ladatuista viesteistä sellaiset viestit, jotka ovat täsmälleen samanlaisia, koska niitä löytyi datasetistä huomattava määrä.
3. Viestit tokenisoidaan NLTK:n TweetTokenizerilla.
4. Kunkin kirjoittajan kokonaisuudessaan kirjoittama sanamäärä lasketaan tokenisoitujen sanojen perusteella koko datasetin laajuisesti. Analyysistä rajataan pois sellaiset kirjoittajat, jotka ovat kirjoittaneet vähemmän kuin 600 sanaa, sillä IBM:n mukaan 600:aa sanaa voidaan pitää sanamäärän raja-arvona, jonka alapuolella analyysi alkaa menettää luotettavuuttaan.[19]
5. Kirjoittajat ja viestit muotoillaan IBM Watson ™ Personality Insights -rajapinnalle sopivaan muotoon, ja lähetetään rajapinnan analysoitavaksi. Kyselyyn lisättiin parametrit kulutustottumusten sekä analyysin skaalaamattomien raaka-arvojen vastaanottamiseksi.
6. Rajapinnan kautta saapuvat tulokset otetaan vastaan ja tallennetaan sanakirja-tietorakenteeseen (Pythonin tietotyyppi dict), josta ne konvertoidaan Dataframeen from_dict-metodilla.
7. JSON-muotoinen data (personality, values, needs, consumption preferences) muunnetaan sarakkeisiin (Kuva 4) seuraavasti:
   - Personality (Luku 5.1)
   - Needs (Luku 5.2)
   - Values (Luku 5.3)
   - Consumption_preferences (Luku 5.4)
8. Persentiilejä, raaka-arvoja ja korrelaatioita (Luku 5.5) tarkastellaan esimerkiksi aiotun käyttäytymisen arvioimiseksi.
9. Tulokset tallennetaan muotoiltuna excel-tiedostoon.

[18] https://www.ibm.com/watson/services/personality-insights/
[19] https://www.ibm.com/watson/developercloud/personality-insights/api/v3/curl.html?curl

word_count	processed_language	'big5_openness'_'percentile'	'big5_openness'_'raw_score'	t_adventurousness'_'percent	t_adventurousness'_'raw_sc
879	en	0,871595481	0,784855656	0,586032371	0,512547931
'facet_intellect'_'percentile'	'facet_intellect'_'raw_score'	'facet_liberalism'_'percentile'	'facet_liberalism'_'raw_score'	s_conscientiousness'_'perce	s_conscientiousness'_'raw_sc
0,990937719	0,716625989	0,949996999	0,59907338	0,294391095	0,602836884
acet_orderliness'_'percentil	acet_orderliness'_'raw_scor	et_self_discipline'_'percent	et_self_discipline'_'raw_sco	cet_self_efficacy'_'percenti	cet_self_efficacy'_'raw_scor
0,590729134	0,501670864	0,752626158	0,603178425	0,65666161	0,767285484
cet_cheerfulness'_'percenti	cet_cheerfulness'_'raw_sco	_excitement_seeking'_'perc	excitement_seeking'_'raw_	acet_friendliness'_'percenti	acet_friendliness'_'raw_scor
0,31903863	0,602352483	0,077293993	0,543247302	0,609102184	0,574380718
acet_cooperation'_'percenti	acet_cooperation'_'raw_scor	'facet_modesty'_'percentile'	'facet_modesty'_'raw_score'	'facet_morality'_'percentile'	'facet_morality'_'raw_score'
0,994631061	0,733087785	0,504118362	0,448920444	0,747049628	0,657199372
'facet_anger'_'percentile'	'facet_anger'_'raw_score'	'facet_anxiety'_'percentile'	'facet_anxiety'_'raw_score'	acet_depression'_'percenti	acet_depression'_'raw_score
0,002721104	0,383709891	0,11992885	0,513956961	0,10621226	0,369460968
need_challenge'_'percentile	need_challenge'_'raw_score	need_closeness'_'percentile	need_closeness'_'raw_score'	need_curiosity'_'percentile'	need_curiosity'_'raw_score'
0,008477242	0,624699848	0,095256947	0,735256362	0,036660577	0,76951744
'need_liberty'_'percentile'	'need_liberty'_'raw_score'	'need_love'_'percentile'	'need_love'_'raw_score'	eed_practicality'_'percenti	need_practicality'_'raw_score
0,002964629	0,633428794	0,081015578	0,700639875	0,004842897	0,67188297
alue_conservation'_'percenti	alue_conservation'_'raw_sco	_openness_to_change'_'perc	openness_to_change'_'raw	value_hedonism'_'percentile	value_hedonism'_'raw_score
0,002577452	0,539436146	0,501494937	0,78441266	0,064081536	0,640975947
ehavior_tuesday'_'percentag	avior_wednesday'_'percent	havior_thursday'_'percenta	behavior_friday'_'percentage	havior_saturday'_'percenta	behavior_0000'_'percentage
0	0	0	0	1	0
behavior_0700'_'percentage	behavior_0800'_'percentage	behavior_0900'_'percentage	behavior_1000'_'percentage	behavior_1100'_'percentage	behavior_1200'_'percentage
0	0	0	0	0	0
behavior_1900'_'percentage	behavior_2000'_'percentage	behavior_2100'_'percentage	behavior_2200'_'percentage	behavior_2300'_'percentage	ve to ownership cost when b
0	0	0	0	0	1
ed by online ads when maki	d by social media when mak	nced by family when making	dulge in spur of the moment	prefer using credit cards for	Likely to eat out frequently
0	0	0	0	1	0
Likely to like horror movies	Likely to like musical movies	ikely to like historical movie	ly to like science-fiction mo	Likely to like war movies	Likely to like drama movies
0	1	1	1	1	0
ly to attend live musical eve	to have experience playing	Likely to like Latin music	Likely to like rock music	Likely to like classical music	Likely to read often
1	1	1	1	1	1

Kuva 4. Esimerkki rajapinnan kautta saadusta tauluksi muotoillusta datasta

5.1 Persoonallisuustyypit

Persoonallisuuspiirteet (personality traits) jaetaan viiteen dimensioon (Avoimuus – Openness, Tietoisuus tai Tunnollisuus – Conscientiousness, Ulospäinsuuntautuneisuus – Extraversion, Neuroottisuus tai Tunteellisuus – Emotional Range, Miellyttävyys – Agreeableness) ja kuhunkin dimensioon kuuluu kuusi fasettia (yhteensä 30 fasettia). Piirteet saadaan IBM Personaliy Insight palvelusta sekä persentiileinä että raaka-arvoiva (raw scores).

Otetaan trait_id ja yhdistetään siihen ensin percentile ja sitten raw_score – näin tehdään sekä big5-alkuisille persoonallisuustyypeille että niihin liittyville facet-alkuisille ‘lapsille’, joiden eteen ei lisätä big5-alkuista etuliitettä, koska fasetit eivät esimerkiksi summaudu big5-tasolla. Sarakemuuttujat ja niiden arvot (pl. kaksoispiste) ovat esimerkiksi seuraavat:

- 'big5_openness'_'percentile': 0.8715954809443731
- 'big5_openness'_'raw_score': 0.7848556557366698
- 'facet_adventurousness'_'percentile': 0.5860323710731868
- 'facet_adventurousness'_'raw_score': 0.5125479310558989
- . . .

5.2 Tarpeet

Tarpeet kuvaavat sitä, mitä ihminen toivoo palvelulta tai tuotteelta.

Otetaan trait_id ja yhdistetään siihen ensin percentile ja sitten raw_score. Sarakemuuttujat ja niiden arvot (pl. kaksoispiste) ovat esimerkiksi seuraavat:

- 'need_challenge'_'percentile': 0.00847724229809843
- 'need_challenge'_'raw_score': 0.6246998481739657
- 'need_closeness'_'percentile': 0.09525694743031426
- 'need_closeness'_'raw_score': 0.7352563618150615
- . . .

5.3 Arvot

Viisi arvoa (tai uskomusta) kuvaavat sitä, mikä on ihmiselle tärkeää.

Otetaan trait_id ja yhdistetään siihen ensin percentile ja sitten raw_score. Sarakemuuttujat ja niiden arvot (pl. kaksoispiste) ovat esimerkiksi seuraavat:

- 'value_conservation'_'percentile': 0.002577452402121816
- 'value_conservation'_'raw_score': 0.5394361457052651
- 'value_openness_to_change'_'percentile': 0.5014949371877822
- 'value_openness_to_change'_'raw_score': 0.7844126600238749
- . . .

5.4 Kulutusmieltymykset

Kulutusmieltymykset (consumption preferences) on johdettu persoonallisuusominaisuuksista.

Otetaan consumption_preference_id-kohtaisesti name-arvo ja yhdistetään siihen score-arvo. Sarakemuuttujat ja niiden arvot (pl. kaksoispiste) ovat esimerkiksi seuraavat:

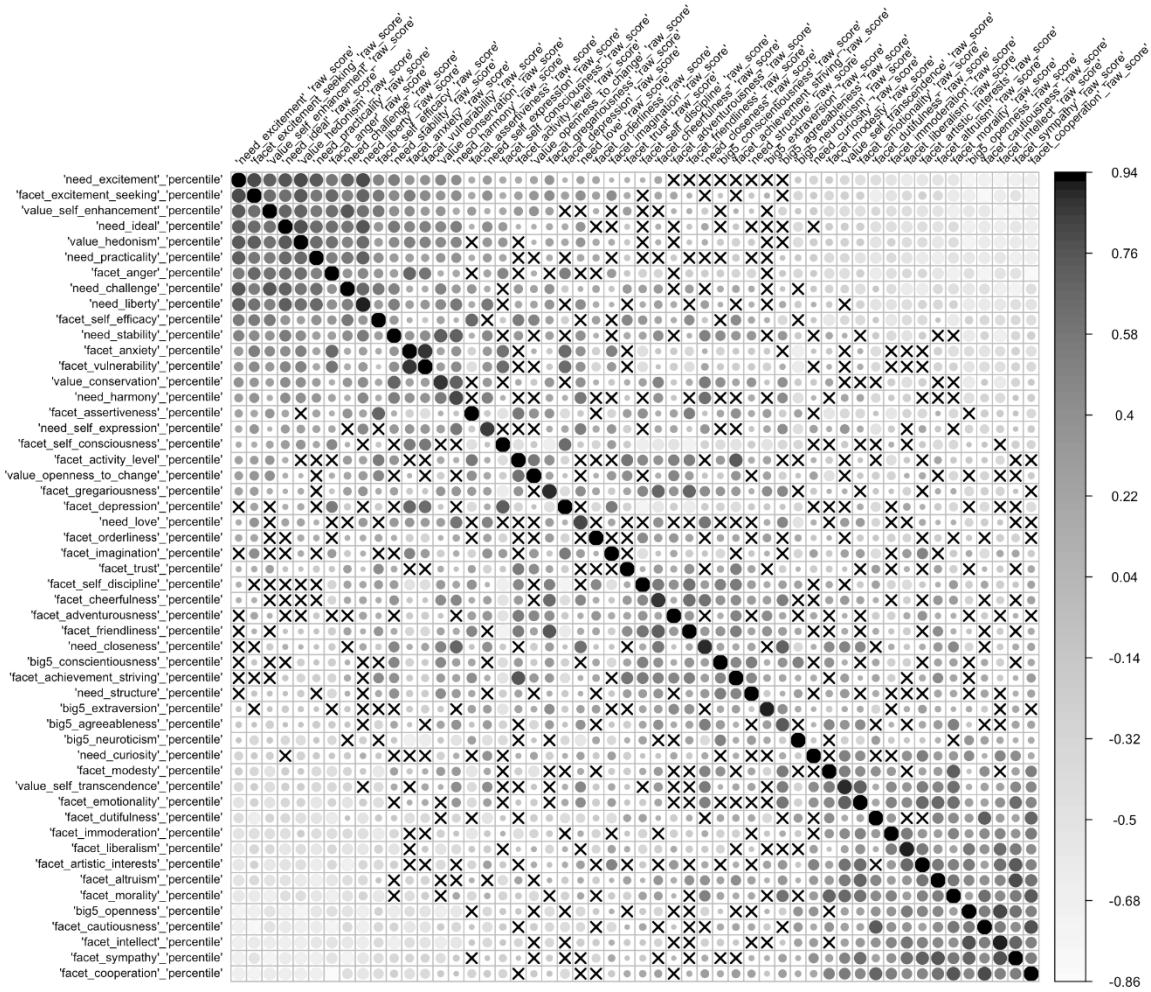
- 'Likely to be sensitive to ownership cost when buying automobiles': 1.0
- 'Likely to prefer safety when buying automobiles': 1.0
- 'Likely to prefer quality when buying clothes': 1.0
- . . .

5.5 Korrelaatiomatriisit

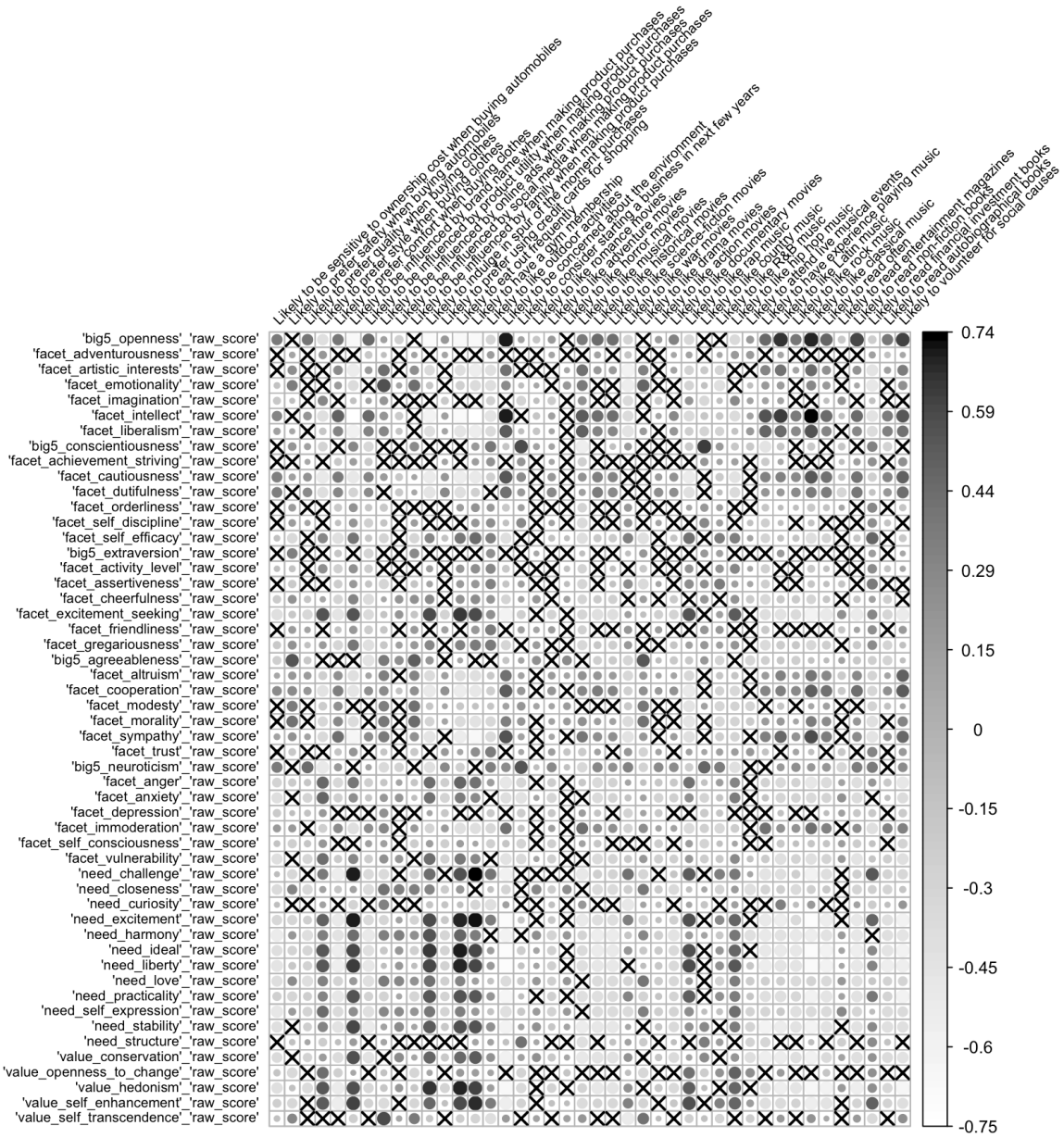
Korrelaatiomatriisien muodostamiseen käytetty koodi on liitteessä D. IBM-rajapinnan kautta saaduista persoonallisuusanalyysin tuloksista laadittiin korrelaatiomatriisit, joista voidaan havaita, että persoonallisuusominaisuudet korreloivat suhteessa toisiinsa (Kuva 5) ja kulutusmieltymyksiin (Kuva 6). Lisäksi kulutusmieltymykset korreloivat toistensa kanssa (Kuva 7). IBM-persoonallisuusdata ladataan Pythoniin excel-tiedostosta ja suoritetaan seuraavat vaiheet:

1. Datan raw_score –tyyppiset (rajapinnan antama raaka-arvo, erotuksen persentiilistä) arvot ajetaan Scikit-learn[20] -kirjaston perprocessing.MinMaxScaler.fit_transform() –funktion läpi, joka normalisoi arvot ja skaalaa ne 0 – 1 välille.
2. Normalisoitu ja skaalattu data tallennetaan excel-tiedostoon.
3. Data ladataan excel-tiedostostosta R[21]:ään
4. Lasketaan halutut korrelaatiot muuttujien välillä Harrell Miscellaneous[22] -kirjaston rcorr-funktiolla, joka laskee korrelaatioille myös p-arvot. Halutut muuttujat kutakin korrelaatiomatriisia varten suodatetaan datajoukosta R-kieleen kuuluvalla matriisien indeksointitoiminnolla.
5. Tuotetaan saadusta korrelaatiomatriiseista kuvat R:n corrplot-kirjastolla[23] halutulla muotoilulla. Tässä tapauksessa oli tarkoituksenmukaista valita väripaletiltaan mustavalkoinen, mahdollisimman helppolukuinen muotoilu.
    a. Korrelaatiomatriiseista tuotettuihin kuviin on merkitty ruksilla korrelaatiot, joiden vahvuus ei ylittänyt tilastollista merkitsevyyttä. Merkitsevyyden raja-arvoksi asetettiin p-arvo 0,05.
    b. Diagonaalin suhteen symmetrisistä matriiseista leikattiin redundantit osiot pois, jolloin matriisi muodostui kolmion muotoiseksi.
    c. Dimensioiltaan symmetrisille korrelaatiomatriiseille suoritettiin lisäksi corrplot-kirjaston mahdollistama pääkomponenttianalyysiä hyödyntävä FPC-järjestys/klusterointi. Suorakulmion mallisille matriiseille tämä ei ollut mahdollista.

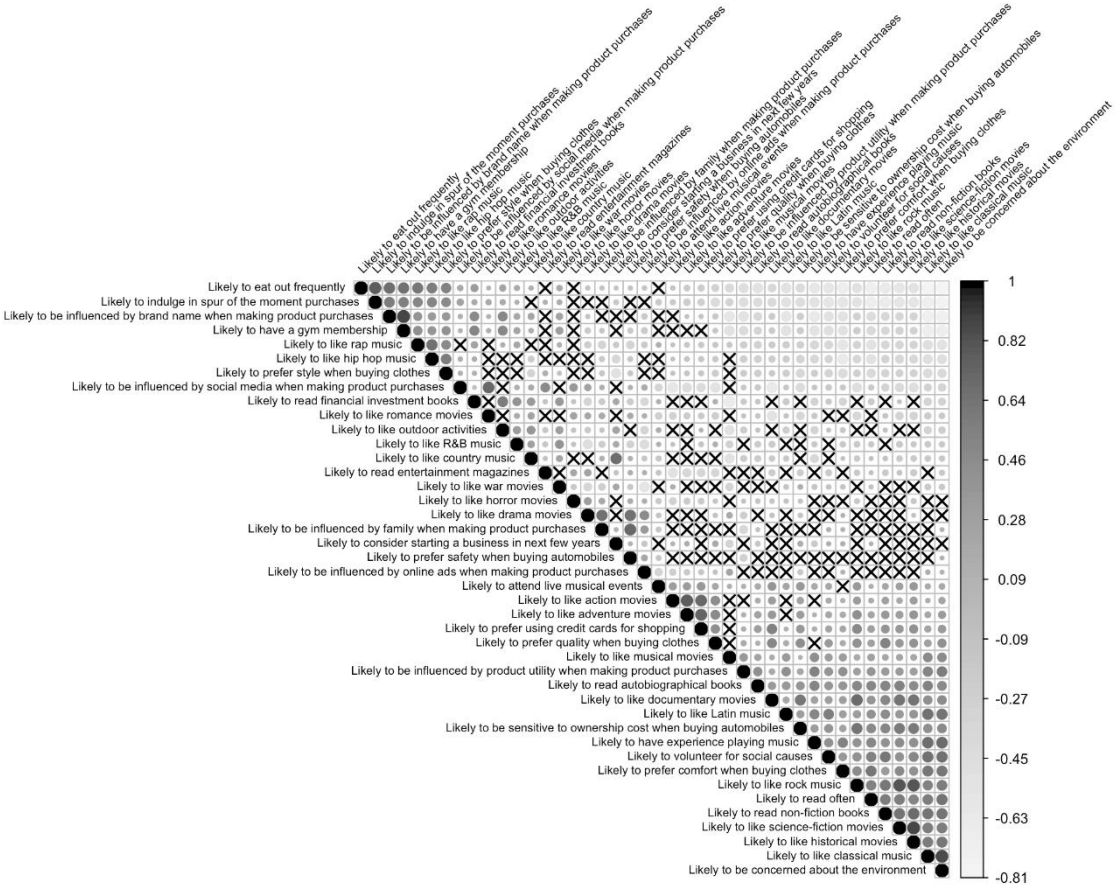
[20] http://scikit-learn.org/stable/index.html
[21] https://www.r-project.org/
[22] https://cran.r-project.org/web/packages/Hmisc/index.html
[23] https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html



Kuva 5. Persoonallisuusominaisuudet (persentiilit ja raaka-arvot)



Kuva 6. Persoonallisuusominaisuudet (raaka-arvot) ja kulutusmieltymykset



Kuva 7. Kulutusmieltymykset

6 Kuvissa esiintyvien henkilöiden tunnetilojen tunnistaminen

Seuraavaksi kuvataan datasetin viestien liitteinä olleiden kuvien lataaminen Facebookista[24] (Luku 6.1), kasvoiksi luokiteltavien piirteiden tunnistaminen ja leikkaaminen kuvista (Luku 6.2) sekä tunnetilojen analysointi leikatuista kasvokuvista (Luku 6.3).

6.1 Kuvien lataaminen

Koodi kuvien lataamiseksi linkeistä löytyy liitteestä E. Facebook-viesteissä on kuvia, joiden lataamisen päävaiheet ovat seuraavat:

1. Ladataan data (viestien osoitteet) excel-tiedostosta.
2. Iteroidaan osoitteet rivi kerrallaan läpi. Kerätään osoitteista erilleen muotoilultaan sellaiset osoitteet, joissa käyttäjän nimi on selkokielisenä ID-numeron sijaan. Pyydetään Facebookin Graph API[25]:n yli käyttäjien ID-numerot.
3. Korvataan osoitteisiin käyttäjien selkokieliset nimet ID-numeroilla. Mikäli osoite ei ole muuten sellaisenaan käyttökelpoinen muotoilunsa tai puutteellisten tietojen vuoksi, pyritään muotoilemaan osoitteet sopivaan muotoon Graph API -kutsuja varten.
4. Kun osoitteet on muotoiltu mahdollisuuksien mukaan Graph API:lle sopivaan muotoon, pyydetään API:n yli latausosoitteet saatavilla oleviin kuviin.
5. Ladataan kuvat http[26]:n yli saaduista latausosoitteista. Kuvat nimetään alkuperäisen viestin osoitteen rivinumeron perusteella.

6.2 Kasvoiksi luokiteltavien piirteiden eristäminen kuvista

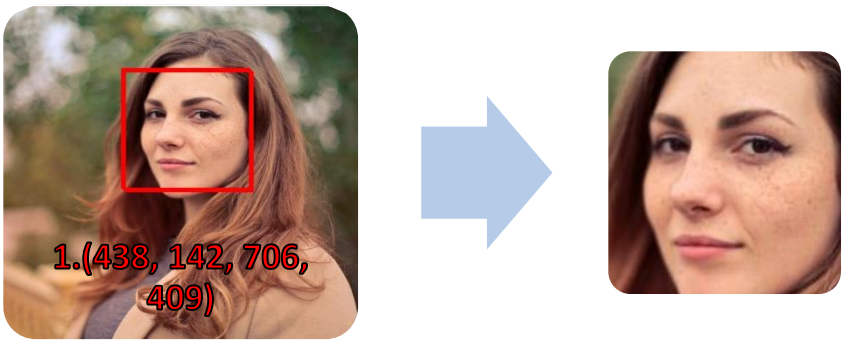
Visuaalinen esitys menetelmästä Kuva 8

Jotta voidaan arvioida kasvojen piirteitä, niin kuvista on leikattava kasvot seuraavasti:

1. Iteroidaan ladatut kuvat Pythonin face_recognition[27] -kirjaston läpi.

[24] https://www.facebook.com/
[25] https://developers.facebook.com/docs/graph-api
[26] https://developer.mozilla.org/en-US/docs/Web/HTTP
[27] https://github.com/ageitgey/face_recognition

2. Face_recognition -kirjasto antaa neljästä arvosta muodostuvan rajaavan alueen kullekin kuvassa havaitulle kasvolle. Yhdessä kuvassa voi olla useita tunnistettuja kasvoja.
3. Face_recognition -kirjaston antamat arvot muunnetaan Pythonin Pillow[28] -kuvankäsittelykirjastolle sopivaan järjestykseen. Esimerkiksi (142, 706, 409, 438) muunnetaan muotoon (438, 142, 706, 409)
4. Alkuperäisistä kuvista leikataan erilleen kasvojen sijaintia kuvaavien arvojen perusteella kullekin kasvolle oma kuvansa, joka tallennetaan erilliseen paikkaan. Leikkeet nimetään siten, että ne ovat yhdistettävissä alkuperäisestä datasetistä löytyviin riveihin.
5. Ohjelmassa on toiminto, jolla voi piirtää tunnistetun kasvoalueen ympärille rajaavan alueen. Rajatuilla alueilla täydennetyt kuvat tallentuvat omaan kansioonsa.



Kuva 8: Esimerkki menetelmän mukaisesta kasvontunnistuksesta

6.3 Tunnetilojen eristäminen kuvista rajatuista kasvoalueista

Kasvokuvista voidaan tunnistaa seuraavat tunnetilat: neutraali, iloinen, yllättynyt, surullinen, vihainen, iljetys, pelko, halveksinta. Seuraavat päävaiheet toteutettiin tunnetilojen tunnistamiseksi:

1. Ladataan ohjelmaan MXNet-kirjastoon[29] valmiiksi koulutettu ONNX[30]-muotoinen malli. Tässä tapauksessa käytettiin mallia FER+ Emotion Recognition[31]

[28] http://python-pillow.org/
[29] https://mxnet.apache.org/
[30] https://github.com/onnx/onnx
[31] https://github.com/onnx/models/tree/master/emotion_ferplus

2. Muunnetaan kappaleessa 6.2 leikattujen kasvoiksi tunnistettujen kuva-alueiden koko 64*64 pikseliin, muunnetaan kuvat mustavalkoisiksi ja invertoidaan värit hyödyntäen Pythonin Pillow-kirjastoa.
3. Suoritetaan kuvien muuntaminen MXNetiin ladatulle ONNX-mallille sopivaan taulukkomuotoon.
4. Ajetaan taulukkomuotoon muutetut kuvat ONNX-mallin lävitse. Malli palauttaa taulukkomuodossa kullekin kuvalle arvot seuraavista tunnetiloista: neutraali, iloinen, yllättynyt, surullinen, vihainen, iljetys, pelko, halveksinta.
5. Muunnetaan saadut arvot todennäköisyyksiksi softmax-funktiolla[32]. Esimerkki muunnoksesta löytyy taulukosta Taulukko 8.
6. Lisätään havaitut tunnetilat excel-tiedostoon seuraavasti: softmax-funktion antaman tunnearvon ollessa > 0,5 kyseisen kasvokuvan katsotaan sisältävän kyseisen tunteen. Kyseistä tunnearvoa merkitsevään sarakkeeseen lisätään kuvan sisältäneelle riville 1. Sarakkeet sisältävät tiedon siitä, kuinka monta esiintymää kustakin tunteesta kyseisessä kuvassa on havaittu.
7. Excel-tiedostoon lisätään myös tieto siitä, kuinka monta kasvoiksi tunnistettua aluetta kussakin kuvassa on havaittu. Tämä tieto lisätään omaan sarakkeeseensa kullekin riville, joita vastaavista kuvista on kyetty löytämään ainakin yhdet kasvot.
8. Excel-tiedoston muutokset tallennetaan.

Taulukko 8: Esimerkki softmax-muunnoksesta

Ennen softmax-funktiota	**Softmax-funktion jälkeen**
3.791335105895996	1.88517235e-01
5.2368292808532715	8.00057509e-01
-0.817960798740387	1.87741068e-03
0.24144500494003296	3.34146497e-03
-2.0410044193267822	5.52583852e-04
-2.404602527618408	3.84140005e-04
-2.8822126388549805	2.38268210e-04
0.16784149408340454	5.03138767e-03

[32] https://en.wikipedia.org/wiki/Softmax_function

LIITE A - Python koodi sentimenttien etsimiseen

```
#messages: A list of messages
def sentiment_sentiwordnet(messages):
    sentiments = []
    tknzr = TweetTokenizer()
    message_count = 0
    for message in messages:
        message_count += 1
        tokenized_message = tknzr.tokenize(str(message))
        tokenized_message = remove_punc_lower(tokenized_message)
        tokenized_message = insert_pos_and_lemma(tokenized_message)
        if len(tokenized_message) <= 0:
            sentiments.append(np.nan)
            continue
        score = 0
        scores = 0
        for i, word_wo_punc in enumerate(tokenized_message):
            ss_set = list(swn.senti_synsets(*word_wo_punc))
            if len(ss_set) < 1:
                continue
            scores += 1
            score += ss_set[0].pos_score() - ss_set[0].neg_score()
        if scores > 0:
            sentiments.append(score / scores)
        else:
            sentiments.append(np.nan)

  return sentiments
#messages_eng: A list of messages
#close_after: If the CoreNLP object should be closed after the function finishes.
#nlp: An object for communicating with CoreNLP
def sentiment_corenlp(messages_eng, close_after = False, nlp = None):
    if nlp == None:
        raise ValueError("No CoreNLP object given. Please use the named parameter 'nlp'")
    sentiment_json = []
    sentiment_float = []
    start = time.time()
    count = 0
    exceptions = 0
    for message in messages_eng:
        try:
            sentiment_json.append(json.loads(nlp.annotate(str(message)))['sentences'])
        except Exception as e:
            print("There was an exception for CoreNLP: ", str(e), " outputting NaN for this particular sentiment")
            sentiment_json.append([{'sentimentValue': np.nan}])
            exceptions += 1
        count += 1
    for message in sentiment_json:
        average = 0
        for sentence in message:
            try:
                sentVal = int(sentence['sentimentValue'])
            except ValueError:
                sentVal = np.nan
            if not np.isnan(sentVal):
                average += int(sentVal)
            else:
                average = np.nan
                break
        if average != np.nan:
            #Sentiment scaled from 0 - 4 to -1 - 1
            #If message empty, append nan
            if len(message) > 0:
                average = ((average / len(message)) - 2) / 2
                sentiment_float.append(average)
            else:
                sentiment_float.append(np.nan)
        else:
            sentiment_float.append(np.nan)
    if close_after:
        nlp.close()
```

```
    return sentiment_float

#messages: A list of messages
def sentiment_vader(messages):
        start = time.time()
        count = 0
        analyzer = SentimentIntensityAnalyzer()
        sentiments = []
        for message in messages:
            message_sentences = sent_tokenize(str(message))
            count += 1
            average = 0
            for sentence in message_sentences:
                average += analyzer.polarity_scores(str(sentence))['compound']
            if len(message_sentences) > 0:
                sentiment = average / len(message_sentences)
            else:
                sentiment = np.nan
            sentiments.append(sentiment)
        return sentiments
```

LIITE B - Python koodi yhteistyötahojen etsinnälle

```
#messagesOrig: Dataframe which contains the messages to be analyzed
#column: Column name for the message column
#cooperators: Dataframe which contains the cooperator names and ignore words
#use_stem: Should the cooperator names be stemmed
#concordance_range = The distance at which the function still considers multi-part
#cooperator names to be a part of the same cooperator name
def detect_cooperator_handles(messagesOrig, column, cooperators, use_stem = False,
                              concordance_range = 2):
    messages = messagesOrig.loc[:,[column]]
    tknzr = TweetTokenizer()
    #Ignore these tokens. Used for URLs mainly.
    ignore_tokens = {'http://','https://','www.'}
    #Ignore tokens that might be confused for a cooperator
    cooperator_ignore_tokens = cooperators['ignore']
    #Have to make a copy, as the dictionary changes during looping
    if use_stem:
        cooperators_intermed = deepest_copy(cooperators['stemmed'])
        cooperators = cooperators['stemmed']
    else:
        cooperators_intermed = deepest_copy(cooperators['nonstemmed'])
        cooperators = cooperators['nonstemmed']
    cooperator_dataframe = pd.DataFrame(index=messages.index, columns=cooperators.keys())
    cooperator_dataframe.fillna(False, inplace = True)
    start = time.time()
    for i, message in messages.iterrows():
        message = tknzr.tokenize(str(message.values[0]))
        tokens = remove_punc_lower(message)
        if use_stem:
            tokens = stem(tokens)
        for y, token in enumerate(tokens):
            concordance_tokens = tokens[y-concordance_range:y+concordance_range+1]
            ignore_token_trigger = False
            #Ignore urls for this purpose - they are most likely not handles
            for ign in ignore_tokens:
                if token.startswith(ign):
                    ignore_token_trigger = True
            if ignore_token_trigger:
                continue
            for cooperator_key,cooperator in cooperators.items():
                contains_words = True
                if cooperator_key in cooperator_ignore_tokens:
                    cooperator_ignore_words = cooperator_ignore_tokens[cooperator_key]
                else:
                    cooperator_ignore_words = []
                for ignore_word in cooperator_ignore_words:
                    if ignore_word in token:
                        contains_words = False
                        break
                if not contains_words:
                    break
                for variation_key, variation in cooperator.items():
                    for key, variation_words in variation.items():
                        #If a cooperator is included in the current token
                        if key in token:
                            contains_words = True
                            #If cooperator has multiple words,
                            #check that all words in cooperator are included in tokens
                            #that are in concordance range
                            if len(variation_words) > 1:
                                contains_words = False
                                for word in variation_words:
                                    if word != key:
                                        for concordance_token in concordance_tokens:
                                            if word in concordance_token:
                                                contains_words = True
                        else:
                            contains_words = False
                    if contains_words:
                        cooperators_intermed[cooperator_key][variation_key][key].append(token)
```

```
                    cooperators_intermed[cooperator_key][variation_key][token]=
cooperators_intermed[cooperator_key][variation_key][key]
                    cooperator_dataframe.loc[i:i, cooperator_key] = True
            i += 1
    cooperators = {}
    if use_stem:
        cooperators['stemmed'] = cooperators_intermed
    else:
        cooperators['nonstemmed'] = cooperators_intermed
    messagesOrig = pd.concat([messagesOrig, cooperator_dataframe], axis = 1)
    return (messagesOrig, cooperators)
```

LIITE C - Python koodi IBM persoonallisuusanalyysin noutamiseen ja taulukoksi muotoiluun

```
#messages: A Dataframe which contains the messages
#name_column: The name of the column which contains the names of the authors
#message_column: The column which contains the messages
#time_column: The column which contains the message timestamps
def get_ibm_personality(messages, name_column, message_column, time_column):
    personality_insights = PersonalityInsightsV3(username= "YOUR_USERNAME_HERE",
                                        password = "YOUR_PASSWORD_HERE", version = "2018-04-12")
    personalities = {}
    tknzr = TweetTokenizer()
    for i, message in messages.iterrows():
        if pd.isnull(message[name_column]):
            continue
        if not message[name_column] in personalities:
            personalities[message[name_column]] = {'contentItems': [], 'valid': False}
        personalities[message[name_column]]['contentItems'].append(
                {'content': str(message[message_column]),
                 'contenttype': 'text/plain',
                 'created': time.mktime(datetime.datetime.strptime(
                        message[time_column], "%Y-%m-%d %H:%M:%S +0000").timetuple()),
                 'id': i,
                 'language': 'en'})
    #600 words are needed for acceptable results https://console.bluemix.net/docs/services/personality-
insights/input.html#sufficient
    #Therefore personalities with less than 600 words not included
    valid_amount = 0
    for key, personality in personalities.items():
        total_words = 0
        for item in personality['contentItems']:
            tokenized_message_intermed = tknzr.tokenize(str(item['content']).lower())
            tokenized_message = []
            for token in tokenized_message_intermed:
                if token not in string.punctuation:
                    tokenized_message.append(token)
            total_words += len(tokenized_message)
        if total_words > 600:
            personalities[key]['valid'] = True
            valid_amount += 1
    personalities_final = {}
    iteration = 0
    for k, personality in personalities.items():
        if personality['valid'] == True:
            iteration += 1
            print("Personalities analyzed: " + str(iteration) + " of " + str(valid_amount))
            try:
                personalities_final[k] = personality_insights.profile({'contentItems': personality['contentItems']}, raw_scores = True,
consumption_preferences = True)
            except Exception as e:
                print('Error for key ' + "" + k + "" + ':', str(e))
                personalities_final[k] = {}
    #This is a bit inefficient - the save format was initially different
    almost_there = pd.DataFrame.from_dict(personalities_final, orient='index')
    almost_there.index.name = name_column
    table = personality_json_to_table(almost_there)
    return table

#A partially formed personality insights table – of type Dataframe
def personality_json_to_table(messages):
    preframe = {}
    preframe["kirjoittaja"] = []
    as_is_columns = {"word_count", "processed_language", "warnings", "word_count_message", "average_sentiment"}
    for row_index, row in messages.iterrows():
        preframe["kirjoittaja"].append(row_index)
        for column_name, column in row.iteritems():
            if column_name in as_is_columns:
                if not column_name in preframe:
                    preframe[column_name] = [column]
                else:
```

```
                preframe[column_name].append(column)
            continue
        column = ast.literal_eval(str(column))
        if column_name == "consumption_preferences":
            for preference_category in column:
                for preference in preference_category["consumption_preferences"]:
                    if not preference["name"] in preframe:
                        preframe[preference["name"]] = [preference["score"]]
                    else:
                        preframe[preference["name"]].append(preference["score"])
        elif column_name == "behavior":
            for behavior in column:
                if not "'" + behavior["trait_id"] + "'_'percentage'" in preframe:
                    preframe["'" + behavior["trait_id"] + "'_'percentage'"] = [behavior["percentage"]]
                else:
                    preframe["'" + behavior["trait_id"] + "'_'percentage'"].append(behavior["percentage"])
        else:
            def insert_traits(preframe, trait):
                if not "'" + trait["trait_id"] + "'_'percentile'" in preframe:
                    preframe["'" + trait["trait_id"] + "'_'percentile'"] = [trait["percentile"]]
                else:
                    preframe["'" + trait["trait_id"] + "'_'percentile'"].append(trait["percentile"])
                if not "'" + trait["trait_id"] + "'_'raw_score'" in preframe:
                    preframe["'" + trait["trait_id"] + "'_'raw_score'"] = [trait["raw_score"]]
                else:
                    preframe["'" + trait["trait_id"] + "'_'raw_score'"].append(trait["raw_score"])
                return preframe
            for trait in column:
                preframe = insert_traits(preframe, trait)
                if column_name == "personality":
                    for child in trait["children"]:
                        preframe = insert_traits(preframe, child)
    frame = pd.DataFrame.from_dict(preframe, orient='columns')
    frame.index = frame["kirjoittaja"].values
    frame.index.name = "kirjoittaja"
    frame.drop(columns=["kirjoittaja"], axis = 1, inplace = True)
    return frame
```

LIITE D – R koodi korrelaatiomatriisien muodostamiseen

```
library(readxl)
library("corrplot")
library("Hmisc")

persentiilit_ja_raw <- read_excel("input/persentiilit_ja_raw.xlsx", sheet="Raw_preferences")
#corr <- round(cor(persentiilit_ja_raw), 2)
corr <- rcorr(as.matrix(persentiilit_ja_raw), type="pearson")
corr_filtered <- corr$r[grepl("Likely to", rownames(corr$r)), grepl( "Likely to", colnames(corr$r))]
p.mat <- corr$P[grepl("Likely to", rownames(corr$P)), grepl( "Likely to", colnames(corr$P))]
corrplot(corr_filtered, p.mat=p.mat, sig.level = .05,  method="circle", col = gray.colors(100, start=1, end=0),
      is.corr = FALSE, tl.col = "black", tl.cex = .65, pch.cex = 1.3, tl.srt = 45, order="FPC", type="upper")
```

LIITE E – Python koodi kuvien lataamiseen latauslinkeistä

```
#messages: A Dataframe which contains picture download links
#link_column: Column name for the column that contains the download links
#path: Where the pictures should be saved
def download_pictures(messages, link_column, path):
    print("Downloading pictures")
    start = time.time()
    last_output = start - 9.5
    failed = 0
    pictures = 0
    existing = 0
    if not os.path.exists(path):
        os.makedirs(path)
    for i, row in messages.iterrows():
        if (time.time() - last_output) > 10:
            last_output = time.time()
            timeleft = (time.time() - start) * (float(len(messages.index)) / (i+0.1)) - (time.time() - start)
            print(str(pictures - failed) + " pictures downloaded, " + str(existing) + " existing files skipped, " + str(failed) + " failed.
Estimated time left: " + str(datetime.timedelta(seconds=int(timeleft))))
        if pd.isnull(row[link_column]) or row[link_column] == "":
            continue
        try:
            if os.path.exists(path + "/" + str(i) + ".jpg"):
                #print(path + "/" + str(i) + ".jpg already exists, skipping...")
                existing += 1
                continue
            url = row[link_column]
            pictures += 1
            with open(path + "/" + str(i) + ".jpg", 'wb') as handle:
                response = requests.get(url, stream=True)
                if not response.ok:
                    #print(response)
                    failed += 1
                    continue
                for block in response.iter_content(1024):
                    if not block:
                        break
                    handle.write(block)
        except Exception as e:
            print("(Row " + str(i) + ") Couldn't download picture from " + str(url))
            print("Error: " + str(e))
            failed += 1
    print("Done downloading pictures. " + str(existing) + " existing pictures skipped. " + str(failed) + " of " + str(pictures) + "
pictures couldn't be downloaded.")
```