521160P Johdatus Tekoälyyn

Harjoitus #4

Klusterointi ja Dimensionalisuuden redusointi

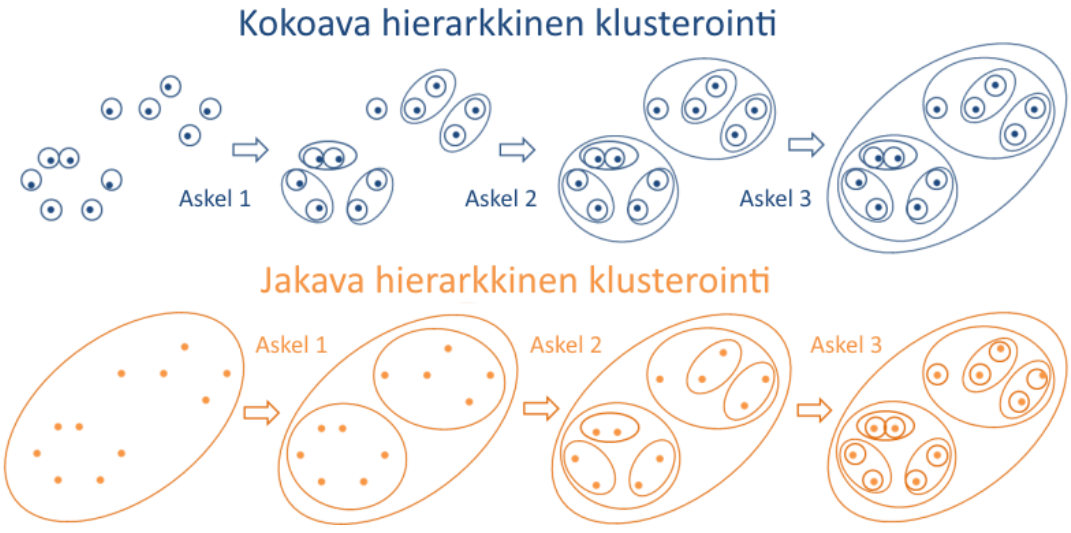
Kevät 2018

Klusterointi

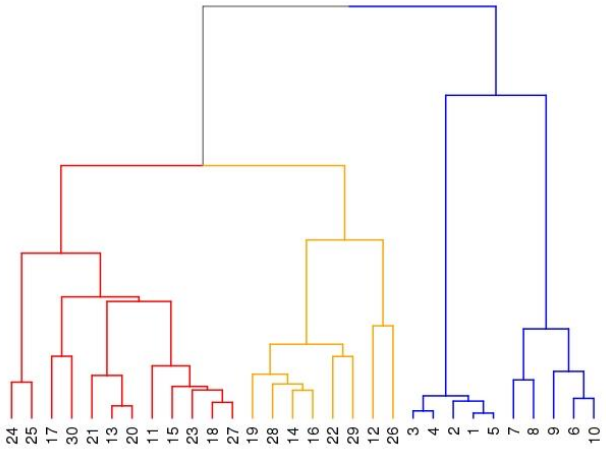
Klusterointi on ohjaamattoman oppimisen tekniikka, jota käytetään datan analysoimiseen ja ryhmittelyyn. Ryhmien löytämiseksi näytteiden väliltä tulee löytää samankaltaisuuksia ja mitata näytteitä esimerkiksi euklidisen etäisyyden tai city block etäisyyden avulla.

Klusteroinnin päämäärä on jakaa data pienempiin ryhmiin, joiden näytteet ovat mahdollisimman samankaltaisia keskenään, mutta eri ryhmien väliset näytteet ovat mahdollisimman erilaisia toisiinsa nähden. Jokaisella eri klusterointiongelmalla ovat omat ominaisuudet, minkä takia ei ole olemassa yleisesti tunnettuja mittoja, jotka toimivat hyvin jokaisessa tapauksessa. Eri klusterointimenetelmät voidaan jakaa karkeasti sen mukaan, mitä ominaisuuksia kyseinen menetelmä hyödyntää.

Hierarkkiset klusterointimenetelmät ovat joko jakavia tai kokoavia. Jakavissa menetelmissä lähdetään liikkeelle joukosta, johon kaikki näytteet kuuluvat ja ne jaetaan samankaltaisuuksien perusteella aina pienempiin ryhmiin askel kerrallaan, kunnes kaikki näytteet on saatu ryhmiteltyä. Kokoavissa menetelmissä lähdetään puolestaan liikkeelle yksittäisistä näytteistä, jotka pyritään yhdistämään aina suurempiin samankaltaisiin kokonaisuuksiin askel askeleelta. Tämän perusteella pystytään piirtämään yksinkertainen puukaavio (dendrogrammi), joka kuvaa hierarkkisesti näytteiden välisiä samankaltaisuuksia. Kuvassa 1 on esitetty havainnollistava kuva hierarkkisten klusterointimenetelmien toiminnasta. Kuvassa 2 on muodostettu dendrogrammi hierarkkisen klusteroinnin lopputuloksesta.



Kuva 1. Havainnollistava kuva jakavan ja kokoavan hierarkkisen klusteroinnin toiminnoista



Kuva 2. Muodostettu dendrogrammi hierarkisen klusteroinnin perusteella

Tiheysperusteisissa klusterointimenetelmissä nimen mukaisesti datanäytteet jaetaan ryhmiin näytteiden tiheystiedon perusteella. Tunnetuin tiheysperusteinen klusterointimenetelmä on nimeltään DBSCAN (Density-based spatial clustering of applications with noise). Algoritmi ottaa parametreina näytteiden välisen minimietäisyyden (epsilon) sekä minimimäärän näytteitä klusterin muodostamiseksi (min_samples). Yhdessä nämä kaksi parametria määräävät tiheyskriteerin, jonka perusteella kyseinen menetelmä yhdistää lähellä toisiaan olevat datanäytteet klusteriksi. Mikäli tiheyden perusteella jokin näytteistä on kaukana mistään ryhmästä (outlier), se jätetään kokonaan luokittelematta. Menetelmän hyviä puolia ovat, että etukäteen ei tarvitse määrittää luokkien lukumäärää, kuten monessa muussa klusterointimenetelmässä. Lisäksi algoritmi pystyy löytämään tiheystiedon perusteella minkä tahansa muotoisia ryhmiä. DBSCAN on kuitenkin erittäin sensitiivinen sen määritettyihin parametreihin ja mikäli tiheyskriteeri on valittu väärin, menetelmä tuottaa ei-toivotun lopputuloksen.

Keskipisteperusteisessa klusterointimenetelmässä tai toisin sanoen K-means menetelmässä mitataan ja verrataan näytteiden samankaltaisuutta klusterikeskipisteihin. Tässä menetelmässä klustereiden lukumäärä tulee tuntea etukäteen. Aluksi k-means algoritmi sijoittaa satunnaisesti klusterikeskipisteet dataan. Tämän jälkeen algoritmi iteroi kahden seuraavan vaiheen välillä:

1. Jokainen näyte datassa on luokiteltu sen mukaan, mikä on näytteen lähin klusterikeskipiste esimerkiksi euklidisen etäisyyden perusteella.

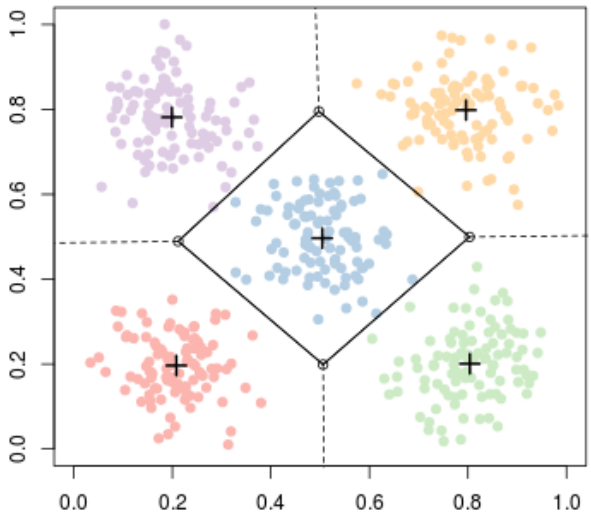
2. Klusterikeskipisteet lasketaan uudestaan ottamalla vaiheen 1 perusteella luokitelluista näytteistä keskiarvot ja asettamalla lasketut keskiarvot uusiksi klusterikeskipisteiksi.

Algoritmi iteroi näiden kahden vaiheen välillä kunnes ennalta määrätty pysähtymiskriteeri toteutuu tai etukäteen annettu iteraatioiden lukumäärä saavutetaan. K-means algoritmilla suppeneminen lopputulokseen on taattu vaikkakin joskus klusteroinnin lopputulos saattaa olla ei-toivotun näköinen. Klusterikeskipisteiden alustaminen on k-means algoritmin toimivuuden kannalta tärkeässä roolissa ja satunnaisen sijoittelun sijaan yksi hyvä strategia on sijoittaa ne yhtä etäälle ja mahdollisimman kauas toisistaan.

Koska klustereiden lukumäärä täytyy tuntea ennen monen klusterointimenetelmän käyttämistä, klustereiden lukumäärän arvioimiseen on kehitetty erilaisia menetelmiä. Yksi menetelmä optimaalisen klustereiden lukumäärän selvittämiseen on silhouette score. Se antaa arvion siitä, kuinka hyvin keskimäärin datanäytteet sopivat niiden omiin klustereihinsa muihin klustereihin verrattuna. Silhouette score saa arvoja

-1 ja 1 väliltä, missä arvo 1 tarkoittaa, että näytteet omissa ryhmissään ovat täysin samanlaisia keskenään ja arvo -1 tarkoittaa, että näytteet omissa ryhmissään ovat täysin erilaisia toisiinsa verrattuna.

Kuvassa 3 on esitetty klusteroinnin lopputulos k-means algoritmilla. Kukin näyte datassa kuuluu lähimpään klusterikeskipisteeseensä. Kuvassa on myös esitetty Voronoi-jaottelu, joka kuvaa klustereiden välisiä luokkarajoja.

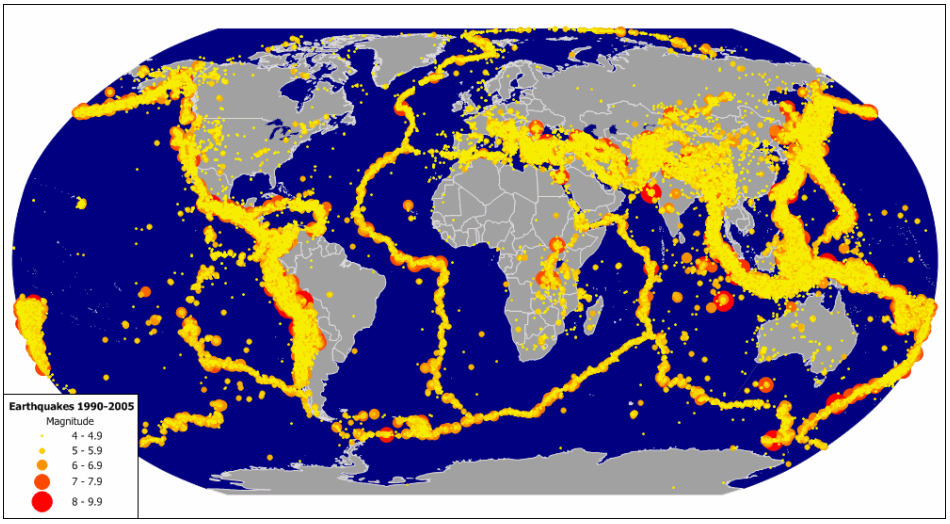


Kuva 3. K-means algoritmi ja Voronoi-jaottelu

Malliperusteisissa menetelmissä data pyritään kuvaamaan mahdollisimman tarkasti jonkin matemaattisen mallin avulla. Matemaattisena mallina voidaan käyttää esimerkiksi tunnettua todennäköisyysjakaumaa tai se voi pohjautua neuraaliverkkoon, josta SOM (engl. self-organizing map) on hyvä esimerkki.

Tehtävä 1

Sinun tehtäväsi on klusteroida dataa vuoden 2017 heinäkuussa maailmalla sattuneista maanjäristyksistä kolmella eri klusterointialgoritmilla: K-means, DBSCAN ja kokoava hierarkkinen menetelmä. Kuvassa 4 on esitetty havainnollistava kuva maailmalla sattuneista maanjäristyksistä vuosina 1990-2005, jota tämän tehtävän datanäytteet mukailevat.



Kuva 4. Vuosina 1990-2005 sattuneet maanjäristykset

Muokkaa tiedostoa nimeltä **earthquake.py**. Datanäytteet sattuneista maanjäristyksistä on annettu normalisoituina (arvot välillä [0,1]) Mercatorin projektion koordinaatteina tiedostossa **normalizedcoordinates.txt**. Tässä tehtävässä main() funktio on valmiiksi annettuna mutta sinun tulee toteuttaa apufunktioiden sisältö.

Aluksi on selvitettävä optimaalisin klustereiden lukumäärä, kun etukäteen on päätetty, että se on jotain 5:n ja 10:n klusterin väliltä. Tämän välivaiheen suorittamiseksi muokkaa funktiota **Silhouette_score(X)**, joka ottaa argumenttina datanäytteet ja palauttaa numeroarvona optimaalisen määrän klustereille.

Silhouette_score(X):

1. Käy läpi esimerkiksi for-silmukassa klustereiden lukumäärät 5-10.

2. Luo joka kerta Kmeans-luokittelija datasetille X komennolla:

```
classifier = KMeans(n_clusters=xxx, n_init=xxx).fit(X)
```

,missä KMeans algoritmin parametri *n_clusters* viittaa klustereiden lukumäärään ja *n_init* luo luokittejan xxx kertaa ja valitsee parhaan lopputuloksen. Anna *n_clusters* arvoksi klustereiden lukumäärä ja *n_init* arvoksi esimerkiksi 100. Huomaa, että .fit(X) lisäyksellä algoritmi opetetaan jo samalla rivillä, missä X viittaa datanäytteisiin.

3. Tämän jälkeen käytetään silhouette scorea arvioimaan klusteroinnin lopputulosta:

```
silhouette = metrics.silhouette_score(X, classifier.labels_, metric='euclidean')
```

4. Kun kaikki vaihtoehdot klustereiden lukumäärästä on käyty läpi, meidän tulee päättää silhouette scoren arvojen perusteella, mikä olisi paras klustereiden lukumääräksi. Tämän voi toteuttaa esimerkiksi päivittämällä jokaisen ”loopin” aikana parhaan silhouette scoren arvoa ja mikäli arvo parani, niin otetaan samalla ylös, mikä oli tällöin klustereiden lukumäärä. Toinen vaihtoehto voisi olla silhouette scoren arvojen päivittäminen listaan ja valitsemalla lopuksi listan maksimiarvon perusteella paras klustereiden lukumäärä.

5. Funktio palauttaa lopuksi optimaalisimman klustereiden lukumäärän.

Kun optimaalisin klustereiden lukumäärä on selvillä, voidaan luoda klusterointialgoritmit KMeans, DBSCAN ja kokoava hierarkkinen klusterointi. Näitä varten on valmiiksi luotu funktiot Kmeans(X, num_clusters), Dbscan(X) ja Agglomerative_clustering(X, num_clusters).

Sekä Kmeans että kokoava hierarkkinen klusterointi tarvitset datanäytteiden lisäksi tiedon klustereiden lukumäärästä, jonka selvitimme aiemmin Silhouette_score(X) funktion avulla. DBSCAN tarvitsee puolestaan informaationaan datanäytteiden tiheyteen liittyviä parametreja (epsilon ja min_samples).

Kmeans(X, num_clusters):

1. Luo K-means luokittelija datasetille X.

```
kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=xxx).fit(X)
```

,missä antamalla parametri *init=’k-means++’* sijoitetaan klusterikeskipisteet aluksi yhtä etäälle toisistaan, *n_clusters* viittaa klustereiden lukumäärään ja *n_init* luo luokittelijan xxx kertaa ja valitsee parhaan lopputuloksen. Anna *n_clusters* arvoksi klustereiden optimaalisin lukumäärä ja *n_init* arvoksi esimerkiksi 100.

2. Luo klusteroitujen datanäytteiden ennustetut ryhmät sekä klusterikeskipisteiden arvot komennoilla:

```
labels = kmeans.labels_
```

```
cluster_centers = kmeans.cluster_centers_
```

3. Funktio palauttaa ryhmiteltyjen datanäytteiden ennustetut ryhmät ja klusterikeskipisteiden arvot (labels, cluster_centers)

Dbscan(X):

1. Luo Dbscan luokittelija datasetille X.

```
dbscan = DBSCAN(eps=xxx, min_samples=xxx).fit(X)
```

,missä parametri *eps* kertoo näytteiden välisen minimietäisyyden, jolla kyseinen näyte ryhmitellään vielä tiettyyn klusteriin ja *min_samples* kertoo, montako näytettä minimissään tulee olla yhdessä klusterissa. Huomaa, että mustat näytteet kuvassa ovat niin sanottuna ulkopuolisia näytteitä (outliers), joita ei ryhmitellä mihinkään ryhmään.

Yritä löytää näille kahdelle parametrille sopivat arvot niin, että klustereita löytyisi datasta noin 10 ja mahdollisimman moni näytteistä ryhmiteltäisiin johonkin klusteriin. Hyvä alkuarvaus tälle datalle on, että epsilonin arvo on 0.01 ja 0.1 välillä ja min_samples arvo on 10-20 näytteen välillä.

2. Luo klusteroitujen datanäytteiden ennustetut ryhmät komennolla:

```
labels = dbscan.labels_
```

3. Funktio palauttaa ryhmiteltyjen datanäytteiden ennustetut ryhmät (labels).

Agglomerative_clustering(X, num_clusters):

1. Luo kokoava hierakkinen klusterointi luokittelija datasetille X.

```
model = AgglomerativeClustering(linkage='ward', n_clusters=num_clusters).fit(X)
```

,missä antamalla parametri *linkage='ward' yhdistetään näytteet minimoiden niiden välinen varianssi* ja *n_clusters* viittaa klustereiden lukumäärään.

2. Luo klusteroitujen datanäytteiden ennustetut ryhmät komennolla:

```
labels = model.labels_
```

3. Funktio palauttaa ryhmiteltyjen datanäytteiden ennustetut ryhmät (labels).

Kirjoita tehtävästä 1 raportti, johon on liitetty kooditiedoston tuottamat kuvaajat. Kommentoi raporttiin lisäksi muutamalla lauseella, että mitä eroavaisuuksia käytetyillä klusterointimenetelmillä on kuvaajien perusteella (KYSYMYS1)? Kerro myös, mikä menetelmistä olisi mielestäsi paras maanjäristysongelman ratkaisuksi, kun halutaan selvittää tarkasti, että missäpäin maapalloa sijaitsevat maanjäristyskeskukset (KYSYMYS2)?

Dimensionalisuuden redusointi

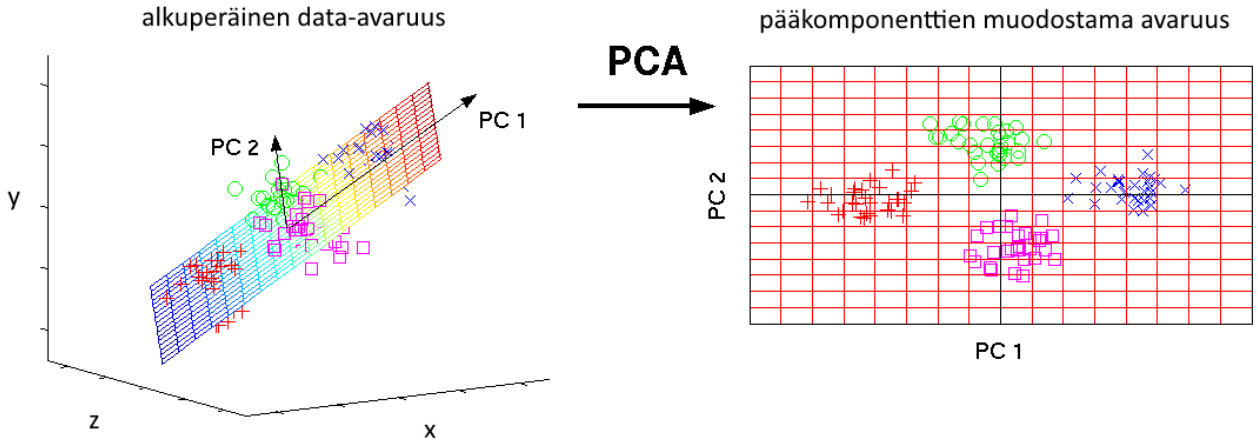
Dimensionaalisuuden redusoinnissa pyritään pienentämään korkea dimensionaalisen datasetin piirteiden lukumäärää informaatiota menettämättä niin, että datan tärkeimmät ominaisuudet voidaan selittää muutaman tärkeimmän piirteen avulla.

Dimensionaalisuuden redusoinnissa on kaksi erilaista lähestymistapaa: piirteiden valitseminen tai piirteiden pakkaaminen. Piirteiden valitsemisessa keskitytään löytämään alkuperäisistä piirteistä koostuva osajoukko, joka mallintaa mahdollisimman hyvin koko datan ominaisuuksia. Piirteiden pakkaamisessa puolestaan muunnetaan korkea dimensionaalinen data pienempi dimensioiseen avaruuteen niin, että menetetään mahdollisimman vähän merkittävää informaatiota. Tässä harjoituksessa käsitellään piirteiden valitsemisen sijaan piirteiden pakkaamista.

Piirteiden pakkaamiseen voidaan käyttää erilaisia ohjaamattoman oppimisen algoritmeja, kuten pääkomponenttianalyysia (PCA, principal component analysis), riippumattomien komponenttienanalyysia (ICA) , Sammon mapping, MDS, t-SNE, Isomap, LLE, jne. Keskitytään tässä harjoituksessa algoritmeihin PCA, MDS ja t-SNE.

Pääkomponenttianalyysissa korkea dimensionaaliseen data-avaruuteen määritellään uudet muuttujat (pääkomponentit), jotka ovat riippumattomia lineaarikombinaatioita kaikista ilmiön kuvaamiseen käytetyistä muuttujista. Näin ollen pääkomponenttianalyysi on lineaarinen dimensionaalisuuden redusointi menetelmä. Siinä pääkomponentit valitaan siten, että ne kuvaavat suurimman osan datan vaihtelusta ja ovat aina kohtisuorassa toisiaan vastaan. Ensimmäisen pääkomponentin tulee selittää mahdollisimman suuren osan datan vaihtelusta, toisen pääkomponentin tulee selittää mahdollisimman suuren osan jäljelle jäävän datan vaihtelusta jne.

Pääkomponenttianalyysi toimii hyvin, mikäli muuttujien välillä on vahvoja korrelaatioita ja datan riippuvuus on lineaarisesti selitettävissä. Muussa tapauksessa kannattaa käyttää jotain epälineaarista dimensionaalisuuden redusointimenetelmää. Kuvassa 5 kolmiulotteiselle data-avaruudelle lasketaan pääkomponentit, jonka jälkeen pääkomponenteista tulee kaksiulotteisen kuvaajan akselit ja dimensioita saadaan pudotettua yhdellä.



Kuva 5. Pääkomponenttianalyysin toimintaperiaate kolmiulotteisen datan muunnoksesta kaksiulotteiseksi.

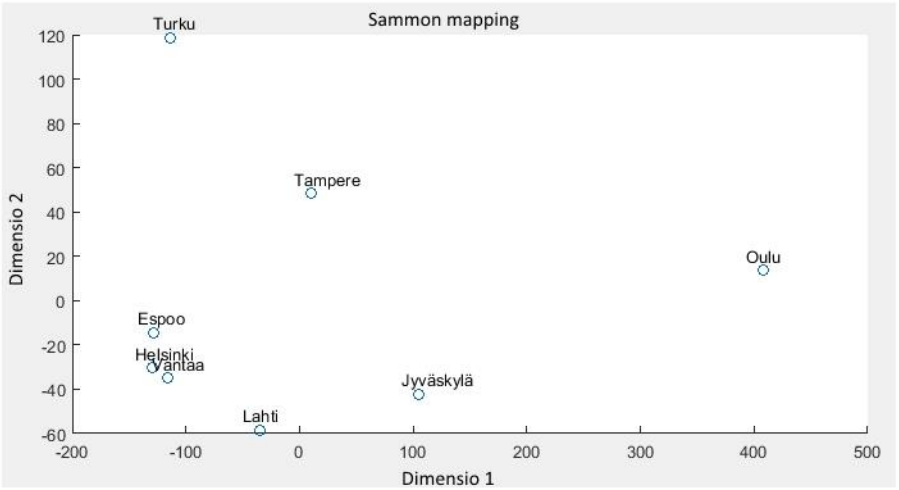
MDS (Multidimensional scaling) on epälineaarinen dimensionaalisuuden redusointi menetelmä. MDS algoritmi pyrkii sijoittamaan jokaisen datanäytteen N-dimensioisessa avaruudessa matalampi dimensioiseen avaruuteen niin, että datanäytteiden väliset etäisyydet säilyvät muunnoksen jälkeen mahdollisimman ennallaan. Etäisyysmittana voidaan käyttää esimerkiksi euklidista etäisyyttä. Täydellistä sovitusta dimensioiden välillä ei pystytä saavuttamaan mutta minimoitavan häviöfunktion (engl. loss function) avulla voidaan saavuttaa erittäin hyviä tuloksia. MDS algoritmi johtaa Sammon mapping algoritmiin, kun MDS algoritmin käyttämä minimoitava häviöfunktio normalisoidaan korkeadimensioisen avaruuden etäisyyksillä.

Siinä missä pääkomponenttianalyysi analysoi dataa kovarianssimatriisin avulla saaden selville vektorit datan suurimmalle vaihtelulle, käyttää MDS datanäytteiden etäisyystietoa ja häviöfunktiota dimensionaalisuuden vähentämisessä.

Kuvassa 6 on esitetty esimerkki Sammon mapping algoritmin toiminnasta. Esimerkissä algoritmi saa datasettinä matriisissa Suomen kahdeksan suurimman kaupungin maantieteelliset etäisyydet toisiinsa nähden. Tämän jälkeen algoritmi pudottaa 8 dimensioisen datan kahteen dimensioon. Lopputuloksesta huomataan, että kaupunkien paikat kuvaajassa vastaavat melko tarkasti kaupunkien paikkoja Suomen kartalla. Tämä selittyy sillä, että MDS pyrkii säilyttämään mahdollisimman tarkasti alkuperäisen datan etäisyydet.

Symmetrinen etäisyysmatriisi

	Helsinki	Espoo	Tampere	Vantaa	Oulu	Turku	Jyväskylä	Lahti
Helsinki	0	16	161	15	540	150	235	99
Espoo	16	0	151	24	537	134	234	103
Tampere	161	151	0	150	400	143	131	116
Vantaa	15	24	150	0	526	154	220	84
Oulu	540	537	400	526	0	533	309	449
Turku	150	134	143	154	533	0	271	194
Jyväskylä	235	234	131	220	309	271	0	140
Lahti	99	103	116	84	449	194	140	0



Kuva 6. Sammon mapping algoritmin tuottama lopputulos Suomen kaupunkien sijainneista toisiinsa nähden etäisyysmatriisin perusteella.

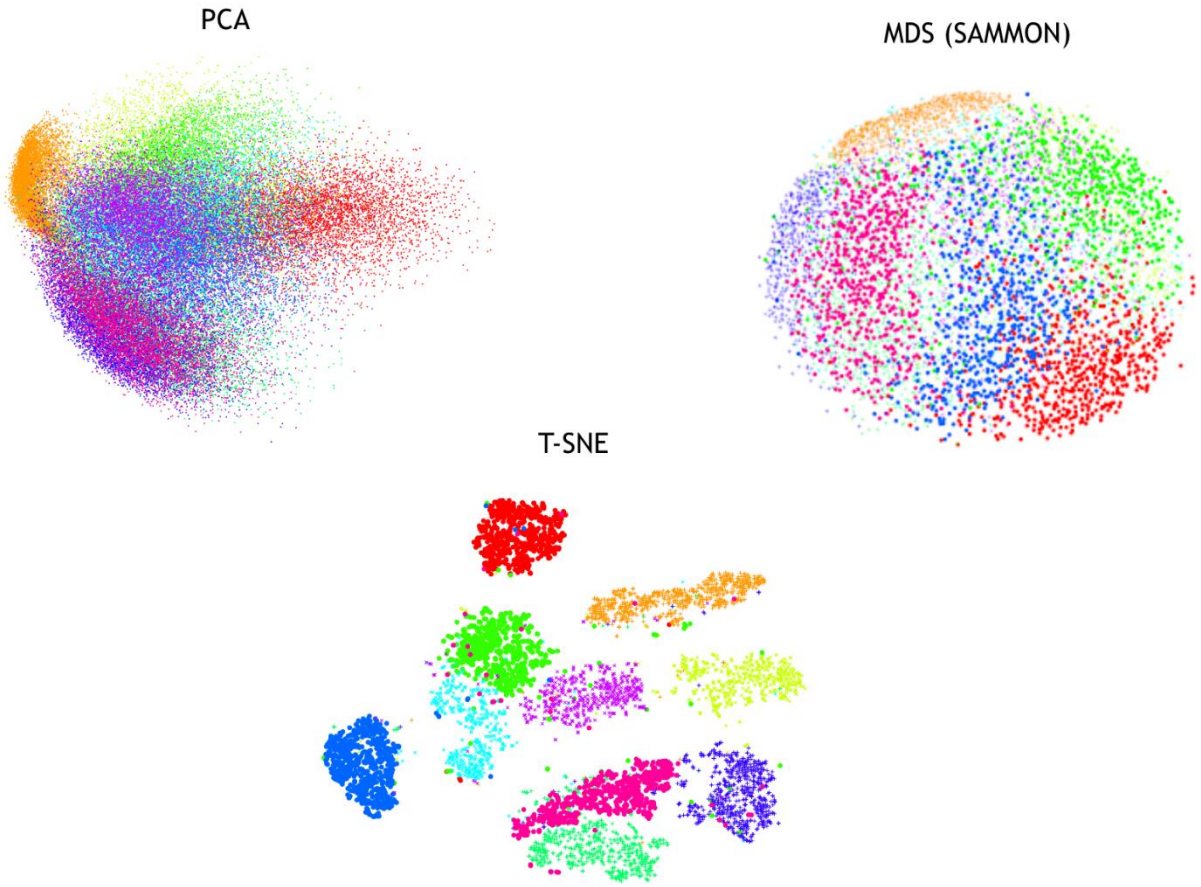
t-SNE (t-distribution stochastic neighbor embedding) on vuonna 2008 kehitetty epälineaarinen dimensionaalisuuden redusointi ja datan visualisointi menetelmä. Se pyrkii säilyttämään sekä datan lokaalin rakenteen (samanlaiset näytteet lähellä toisiaan), että datan globaalin rakenteen (erilaiset näytteet kaukana toisistaan). Lineaaristen dimensionaalisuuden redusointi menetelmien kuten pääkomponenttianalyysin huono puoli on, että ne eivät pysty löytämään datan mahdollisia epälineaarisia ominaisuuksia, toisin kuin esimerkiksi t-SNE tai MDS.

t-SNE algoritmi muodostaa aluksi todennäköisyystiheysjakauman jokaiselle näytteelle korkeammassa dimensiossa, niin että samanlaiset näytteet saavat mahdollisimman suuren arvon ja erilaiset mahdollisimman pienen. Seuraavaksi t-SNE algoritmi määrittää samankaltaisen todennäköisyystiheysjakauman matalammassa dimensiossa sattumanvaraisesti sijoitetuille näytteille. Lopuksi algoritmi yrittää minimoida

Kullback-Leibler divergenssin, jonka seurauksena todennäköisyystiheysjakaumat vastaavat toisiaan dimensioiden välillä.

t-SNE algoritmia voi käyttää käytännössä minkä tahansa korkea dimensioisen datan analysoimiseen mutta erityisesti sitä on käytetty lääketieteelliseen kuvantamiseen, kasvonpiirteiden tunnistukseen sekä puheen tunnistukseen.

Kuvassa 7 on esitetty PCA, t-SNE ja MDS algoritmien tuottamat lopputulokset samalle korkea dimensioiselle datalle. Siitä huomataan, kuinka epälineaariset menetelmät t-SNE ja MDS onnistuvat erottelemaan ryppäitä paremmin kuin lineaarinen PCA.



Kuva 7. Dimensionaalisuuden redusointi PCA, MDS ja t-SNE algoritmeilla samalle datasetille

Tehtävä 2

Tässä tehtävässä sinun tulee pienentää kahden eri datasetin dimensionaalisuutta PCA, MDS ja t-SNE algoritmeilla niin, että datat voidaan esittää kaksiulotteisessa koordinaatistossa. Lisäksi sinun tulee arvioida dimensionaalisuuden vähentämisen lopputulosta ja algoritmien toimintaa.

Datasetteinä käytetään entuudestaan tuttuja MNIST datasettiä sekä harjoituksessa 3 luotua **handdetection.pkl** datasettiä. Korvaa aivan aluksi tehtäväkansiossa oleva handdetection.pkl datasetti harjoituksessa 3 luomallasi vastaavalla datasetillä. Mikäli et ole luonut datasettiä aiemmin, käytä jo kansiossa olevaa käsimerkkidatasettiä.

Molemmissa dataseteissä käytetään piirteinä kuvien pikseleiden arvoja eli MNIST datasetissä lähdetään liikkeelle 28x28 = 784 piirteestä ja käsimerkki-datasetissä on alussa 100x150 = 15000 piirrettä.

Muokkaa tiedostoa **dimensionalreduction.py**. Tässäkin tehtävässä main() funktio on annettuna sinulle valmiiksi ja sinun tulee toteuttaa apufunktioiden sisältö.

Train_tsne(X_data), Train_PCA(X_data) ja Train_MDS(X_data)

(Käydään yhdellä kertaa läpi kaikkien toteutettavien funktioiden sisältö)

1. Ota ylös ajanhetki ennen algoritmin suorittamista.

```
t0 = time()
```

2. Suorita algoritmilla dimensioiden vähentäminen.

```
X_tsne = TSNE(n_components=2, metric='sqeuclidean').fit_transform(X_data)
```

```
X_PCA = PCA(n_components=2).fit_transform(X_data)
```

```
X_MDS = MDS().fit_transform(X_data)
```

3. Ota ylös ajanhetki heti redusoinnin jälkeen.

```
t1 = time()
```

4. Laske aikojen t1 ja t0 välinen ero saadaksesi selville algoritmin suoritusaika.

5. Normalisoi redusoitu data välille [0,1].

```
X_scaled = MinMaxScaling(xxx)
```

,missä xxx on X_tsne, X_PCA tai X_MDS algoritmista riippuen.

6. Funktiot palauttaa muunnetun datan ja ajan, joka kului algoritmin suorittamiseen (X_scaled, time_difference).

Kun saat toimimaan dimensionaalisuuden redusointi funktiot oikein, ajamalla kooditiedosto komentoriville printataan kummallekin datasetille käytettyjen algoritmien suoritusajat. Tämän lisäksi redusoidun datan kompleksisuutta voidaan arvioida opettamalla kaksiulotteisten lopputulosten pohjalta knn-luokittelijat, sillä molemmille dataseteille luokkatiedot näytteille on olemassa. Komentoriville printataan myös redusoitujen lopputulosten pohjalta opetettujen knn-luokittelijoiden F1-scoren arvot. Huomaa, että nyt testisetti-opetussetti jakoa ei tehdä, sillä knn-luokittelijan tarkoitus on ainoastaan vertailla redusoitujen datojen kompleksisuutta toisiinsa nähden F1-scoren perusteella.

Lisäksi kooditiedosto piirtää kaksiulotteiset kuvaajat redusointien lopputuloksista molemmille dataseteille. Kuvaajiin on myös piirretty esimerkki datanäytteitä kuvaamaan niiden sijaintia kuvaajassa.

Kirjoita tehtävästä 2 raportti, johon on liitetty kooditiedoston tuottamat kuvaajat. Taulukoi raporttiin algoritmien suoritusajat ja f1-scoren arvot molemmille dataseteille. Arvioi suoritusaikojen, f1-scoren arvojen ja kuvaajien perusteella, mikä algoritmeista erottelee datan parhaiten ja mikä algoritmeista toimii nopeiten (KYSYMYS1)? Mitä voit kertoa PCA algoritmin kuvaajien perusteella datasettien datan riippuvuuksista (KYSYMYS2)?

Palauta

Palauta tiedostot earthquake.py ja dimensionalreduction.py sekä raportit tehtävistä 1 ja 2 pakattuna tiedostona (.zip tai .rar) Optiman palautuslaatikkoon Harjoitus 4 **27.4.2018 klo 23:59** mennessä. Tästä harjoituksesta on mahdollisuus tienata 5 pistettä (2.5p + 2.5p).