# Podatkovne karte

Naslov tega poglavja je nenavaden, oziroma bolje, nestandarden. Moral bi se glasiti "zmanjšanje dimenzij podatkov", ali pa "projekcije in vektorske vložitve. A ostajamo na kartah. Cilj tega poglavja je pregledati nekaj tehnik, ki lahko podatke, to je primere opisane v atributnem jeziku ali pa primere, med katerimi lahko izračunamo razdaljo, predstavimo kot točke v razsevnem diagramu. Torej, v dveh, ali, če je res potrebno, v treh dimenzijah. Pogoje, ki jim mora taka vizualizacija zadostiti, bodo odvisni od izbrane tehnike. Pri eni bomo na primer želeli, da so točke, to je, primeri, v vizualizaciji čimbolj razpršeni. Pri drugi bomo zahtevali, da razdalje med primeri čimbolj verno odsevajo razdalje v originalnem prostoru primerov. Spet pri tretji nas bo skrbelo samo ohranjanje soseščine.

Poglavje o podatkovnih kartah namenoma sledi poglavju o gručenju. Ti dve tehniki se pogosto uporabljata skupaj. Namreč, večina tehnik gručenja ni namenjena vizualizaciji podatkov. Izjema tu sta na primer hierarhično gručenje in sorodna tehnika zlivanja sosedov. A vse ostale tehnike samo poiščejo gruče, samega postopka oziroma rezultate pa ne znajo predstaviti grafično. S kartami podatkov želimo predstaviti odnose med podatki grafično, to je poiskati sosede oziroma v 2D ali 3D predstaviti oddaljenosti med primeri. V to predstavitev lahko vpnemo, na primer z uporabo barv, rezultate gručenja. Če bodo rezultate skladni, to je, bodo primeri istih gruč skupaj tudi v podatkovni karti, je to potrditev, da odkrite gruče dejansko obstajajo in da jih lahko tudi grafično, z vizualizacijo, predstavimo.

Podatkovne karte služijo tudi razlagi. Morda tu ne bi smeli zapisati "tudi", saj je prav razlaga glavna, oziroma karte podatkov izdeluje prav zato, da bi lahko z eno samo grafično predstavitvijo podatke predstavili, o njih razglabljali, in poiskali zanimive dele kart, ki jih lahko razložimo.

Če bodo osnovni podatki predstavljeni atributno, potem njihovo kartiranje pomeni, da želimo zmanjšati dimenzijo iz $m$ dimenzij, torej iz prostora značilk, v dve ali tri dimenziji. Pravzaprav nas tu, kjer pišemo po papirju, zanimata dve dimenziji, saj treh dimenzij ne moremo predstaviti. Podobno tudi na računalniškem ekranu. Predstavitev v treh dimenzijah je možna na računalnikih možna le ob uporabi interaktivnih vizualizacij, ali pa s posebno opremo za trodimenzionalne prikaze. Zato se bomo tu, v tem poglavju, osredotočili na dvodimenzionalne karte, bodo pa vse tehnike, ki jih bomo predstavili, uporabne tudi za trodimenzionalne prikaze.

Pred opisi še opozorilo: razen za PCA, spodaj pravzaprav ne opišemo, kako pridemo do

rešitev. Pri vseh metodah podamo kriterijske funkcije, a ker za večino teh (izjema je ponovno PCA) ni analitične rešitve, samo omenimo, da za to obstaja numerična. Ta je osnovana na pristopu gradientnega spusta, s katerim se bomo v prihodnjih poglavjih še veliko ukvarjali. Nasploh bodo tehnike, ki jih opisujemo v tem in naslednjem poglavju temeljile na določitvi kriterijske funkcije, ki jo bomo potem prepustili, skupaj s podatki, postopku numerične optimizacije, ki bo za vse tehnike pravzaprav enak.

# Metoda glavnih komponent

Cilj metode glavnih komponent (angl. *principal component analysis*, PCA) je poiskati najbolj informativno dvodimenzionalno predstavitev podatkov. Želimo najti takšne smeri v prostoru značilk, kjer so podatki najbolj razpršeni, torej kjer imajo največjo varianco. S tem zmanjšamo dimenzijo podatkov iz $m$ (število značilk) v 2 (ali 3), hkrati pa ohranimo čim več informacij.

PCA je matematično gledano projekcija podatkov iz višedimenzionalnega prostora v nižje dimenzionalni prostor. Namesto da bi hranili vse značilke, poiščemo nove osi (komponente), ki so linearne kombinacije originalnih značilk, in na te nove osi projiciramo podatke. Nove osi so izbrane tako, da maksimirajo varianco podatkov.

## Matematična izpeljava

### 1. Osrediščenje podatkov

Naj bo $X$ podatkovna matrika dimenzij $n \times m$, kjer je $n$ število primerov in $m$ število značilk. Privzamemo, da so podatki osrediščeni, kar pomeni, da za vsako značilko velja:

$$\frac{1}{n}\sum_{i=1}^{n} X_{i,j} = 0, \quad \text{za vsak } j = 1, 2, \ldots, m$$

Če podatki niso osrediščeni, jih predhodno osrediščimo.

### 2. Iskanje prve komponente $\mathbf{u}_1$

Iščemo vektor $\mathbf{u}_1$ (dimenzije $m \times 1$), ki določa prvo glavno komponento — smer v prostoru značilk, kamor, če projiciramo podatke, bo varianca projekcij največja. Projekcija podatkovne matrike $X$ na $\mathbf{u}_1$ je:

$$z = X\mathbf{u}_1$$

Varianca projekcije $z$ je:

$$\mathrm{Var}(z) = \frac{1}{n}\|X\mathbf{u}_1\|^2 = \frac{1}{n}\mathbf{u}_1^T X^T X \mathbf{u}_1$$

Ker želimo maksimalno varianco, je naša kriterijska funkcija:

$$\max_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1$$

kjer je $S = \frac{1}{n}X^T X$ **kovariančna matrika** podatkov.

Da projekcija ne postane "neskončna", omejimo dolžino vektorja $\mathbf{u}_1$ na 1. To je tudi pogoj, ki mu mora zadostiti kriterijska funkcija:

$$\mathbf{u}_1^T \mathbf{u}_1 = 1$$

## 3. Optimizacija in rešitev

Optimizacijo rešimo z uporabo Lagrangeovih multiplikatorjev. Postavimo Lagrangeovo funkcijo:

$$L(\mathbf{u}_1, \lambda) = \mathbf{u}_1^T S \mathbf{u}_1 - \lambda(\mathbf{u}_1^T \mathbf{u}_1 - 1)$$

Poiščemo stacionarne točke:

$$\frac{\partial L}{\partial \mathbf{u}_1} = 2S\mathbf{u}_1 - 2\lambda\mathbf{u}_1 = 0$$

$$S\mathbf{u}_1 = \lambda\mathbf{u}_1$$

Zgornja enačba je lastna enačba kovariančne matrike. Torej je $\mathbf{u}_1$ lastni vektor kovariančne matrike $S$, pripadajoča lastna vrednost $\lambda$ pa je:

$$S\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

## 4. Rešitev

Varianca projekcije je podana z:

$$\mathrm{Var}(z) = \frac{1}{n}\|X\mathbf{u}_1\|^2 = \mathbf{u}_1^T S \mathbf{u}_1$$

Ker pa iz lastne enačbe velja:

$$S\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

sledi:

$$\mathbf{u}_1^T S \mathbf{u}_1 = \mathbf{u}_1^T \lambda_1 \mathbf{u}_1 = \lambda_1 (\mathbf{u}_1^T \mathbf{u}_1) = \lambda_1$$

ker smo predpostavili, da je $\mathbf{u}_1^T \mathbf{u}_1 = 1$. Zato velja:

$$\mathrm{Var}(z) = \lambda_1$$

kar pomeni, da je lastna vrednost $\lambda_1$ enaka varianci podatkov v smeri prve glavne komponente $\mathbf{u}_1$.

Ker želimo maksimizirati varianco projekcije, vzamemo lastni vektor, ki ustreza največji lastni vrednosti $\lambda_1$. Varianca v smeri $\mathbf{u}_1$ je enaka tej lastni vrednosti:

$$\mathrm{Var}(z) = \lambda_1$$

## Graf pojasnjenih varianc

Rezultate metode glavnih komponent pogosto predstavimo z grafom pojasnjenih varianc, ki ga imenujemo scree diagram. Na osi $x$ prikazujemo zaporedne glavne komponente (1., 2., 3., ...), na osi $y$ pa delež variance, ki jo posamezna komponenta pojasni. Za vsako komponento $k$ izračunamo pojasnjeno varianco kot razmerje med njeno lastno vrednostjo $\lambda_k$ in vsoto vseh lastnih vrednosti:

$$\text{Pojasnjena varianca}_k = \frac{\lambda_k}{\sum_{j=1}^{m} \lambda_j}$$

Graf pojasnjenih varianc omogoča vizualno oceno, koliko komponent je smiselno ohraniti. Ponavadi iščemo "koleno" grafa, točko, kjer se dodatna pojasnjena varianca bistveno zmanjša, kar pomeni, da nadaljnje komponente prispevajo zelo malo informacij.

## Razlaga

Glavne osi, ki jih določi PCA, so predstavljene kot linearne kombinacije osnovnih atributov, kar pomeni, da je vsaka komponenta utežena vsota originalnih značilk. Če je $\mathbf{u}_1$ prva glavna komponenta, potem je vsota:

$$\mathbf{u}_1 = (w_1, w_2, \ldots, w_m)^T$$

kjer so $w_j$ uteži za posamezno značilko. Absolutna vrednost uteži $|w_j|$ nam pove, koliko prispeva posamezni atribut k tej komponenti — večja kot je utež, pomembnejši je vpliv značilke.

Pri interpretaciji uteži moramo upoštevati, da so bili podatki predhodno normalizirani, torej, da imajo vse značilke podobno lestvico (npr. srednja vrednost 0 in standardni odklon 1). To je pomembno, ker PCA sicer favorizira značilke z večjo numerično razpršenostjo, ne glede na njihovo dejansko informacijsko vrednost. Normalizacija torej omogoča, da prispevki k komponentam temeljijo na strukturi podatkov, ne na različnih merskih enotah ali razponih značilk.

## Alternativna numerična rešitev

V praksi, ko želimo podatke projicirati v dve dimenziji, potrebujemo le prvi dve glavni komponenti. Ni treba računati vseh $m$ lastnih vektorjev kovariančne matrike, kar bi bilo računsko zahtevno za velike $m$. Namesto tega lahko uporabimo pohitrene postopke, kot sta potenčna metoda in Gram-Schmidtova ortogonalizacija.

Potenčna metoda je preprost algoritem za iskanje največjega lastnega vektorja simetrične matrike $S$, torej matrike, kot je naša kovariančna matrika. Deluje tako, da poljubni začetni vektor večkrat množimo z $S$ in vsakič normiramo rezultat:

$$\mathbf{v}^{(k+1)} = \frac{S\mathbf{v}^{(k)}}{\|S\mathbf{v}^{(k)}\|}$$

Postopek ponavljamo, dokler se $\mathbf{v}^{(k)}$ ne stabilizira. Rezultat je prvi lastni vektor $\mathbf{u}_1$, ki ustreza največji lastni vrednosti $\lambda_1$.

Ko najdemo $\mathbf{u}_1$, želimo še drugi lastni vektor $\mathbf{u}_2$, ki je ortogonalen na $\mathbf{u}_1$. Da to dosežemo, lahko z Gram-Schmidtovo ortogonalizacijo "počistimo" novo iskanje:

1. Poiščemo približek $\mathbf{v}$ za naslednji lastni vektor (npr. s potenčno metodo).
2. Odštejemo projekcijo na $\mathbf{u}_1$:

$$\mathbf{v}_\perp = \mathbf{v} - (\mathbf{v}^T\mathbf{u}_1)\mathbf{u}_1$$

3. Normiramo $\mathbf{v}_\perp$, da dobimo $\mathbf{u}_2$.

Polni izračun vseh $m$ lastnih vektorjev in vrednosti zahteva čas $O(m^3)$, ker gre za splošno reševanje lastne enačbe. Če pa potrebujemo le prvi dve komponenti, lahko s potenčno metodo in Gram-Schmidtom pridemo do želenih rezultatov v času $O(m^2)$, saj množimo le matriko z vektorjem. To je bistveno hitreje, zlasti za zelo visoke dimenzije, kjer je $m$ veliko.

Takšen pristop omogoča učinkovito hiter izračun dveh glavnih osi in s tem pripravo dvodimenzionalne projekcije podatkov za vizualizacijo.

## Regularizacija

Z regularizacijo se bomo podrobneje ukvarali v kasnejših poglavjih. Tu le omenimo, da je tudi za glavne komponente ta možna in da nam lahko pomaga pri gradnji enostavnejših modelov ki jih lažje razložiti.

Predpostavimo namreč, da pri iskanju glavnih komponent uvedemo tudi regularizacijo uteži vektorjev $\mathbf{u}_1, \mathbf{u}_2, \ldots$, s katerimi določimo posamezne komponente. Cilj take regularizacije je nadzorovati kompleksnost komponent in preprečiti, da bi posamezne značilke imele prevelike uteži. Namen torej je, da dobimo enostavnejše, bolj razložljive komponente, kjer vsak atribut prispeva omejeno (ali pa sploh ne). S tem zmanjšamo tveganje, da bi model sledil šumu, in povečamo možnost interpretacije, saj so komponente enostavnejše in bolj pregledne. Prednost regularizacije je torej v tem, da dobimo stabilne in pogosto redke (angl. sparse) rešitve, kar pomeni, da se v posamezni komponenti pojavijo le nekatere značilke z večjimi utežmi, druge pa imajo ničelne ali skoraj ničelne uteži. To omogoča lažjo razlago komponent, saj je jasno, katere značilke so pomembne.

Obstajata dve glavni obliki regularizacije uteži:

1. **L2 regularizacija (Ridge)**: kaznujemo prevelike vrednosti uteži, tako da dodamo pogoj:

$$\sum_{j=1}^{m} u_j^2 \leq c$$

   kjer je $c$ konstantna meja.
   L2 regularizacija daje gladke rešitve, kjer so vse uteži majhne, vendar večinoma različne od nič.

2. **L1 regularizacija (Lasso)**: kaznujemo vsoto absolutnih vrednosti uteži, s čimer spodbujamo ničelne uteži:

$$\sum_{j=1}^{m} |u_j| \leq c$$

   L1 regularizacija vodi do redkih (sparse) komponent, kjer so mnoge uteži enake nič. Tako so glavne komponente definirane le s podmnožico značilk.

Projekcijo z regulariziranimi utežmi dobimo tako, da v osnovni optimizacijski problem za glavno komponento vključimo še omejitveni pogoj, npr.:

$$\max_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1 \quad \text{ob pogoju} \quad \sum_{j=1}^{m} |u_j| \leq c$$

ali za L2:

$$\max_{\mathbf{u}_1} \mathbf{u}_1^T S \mathbf{u}_1 \quad \text{ob pogoju} \quad \sum_{j=1}^{m} u_j^2 \leq c$$

Tako dobljene komponente omogočajo lažjo interpretacijo, ker so pogosto sestavljene le iz nekaj ključnih atributov, kar je še posebej pomembno pri podatkih z veliko značilkami.

# Večrazredno lestvičenje

Metoda večrazrednega lestvičenja (angl. *multidimensional scaling*, MDS) poišče predstavitev podatkov v nizki dimenziji (navadno v dveh ali treh dimenzijah), kjer bodo medsebojne razdalje med točkami čim bolj podobne razdaljam primerov v originalnem prostoru. Namen metode je torej vizualizacija podatkov, ki temelji na ohranjanju razdalj med primeri. Vhod v MDS so torej medsebojne razdalje med podatkih; metoda torej ne zahteva atributnega zapisa podatkov. Metoda torej deluje neposredno na razdaljah in ne na atributih.

Kriterijska funkcija, ki jo metoda minimizira, je vsota kvadratov razlik med originalnimi in novo dobljenimi razdaljami:

$$\min_{\mathbf{z}_1,\ldots,\mathbf{z}_n} \sum_{i<j} \left(d_{ij} - \|\mathbf{z}_i - \mathbf{z}_j\|\right)^2$$

kjer je $d_{ij}$ podana razdalja med primeroma $i$ in $j$, $\|\mathbf{z}_i - \mathbf{z}_j\|$ pa razdalja v novi nizkodimenzionalni predstavitvi.

Iskanje rešitve tega problema, torej, vložitev primerov v novi nizkodimenzionalni prostor, poteka s postopki numerične optimizacije, saj gre za nelinearen problem brez analitične rešitve. To je drugače od metode PCA, za katero smo lahko našli analitično rešitev. Postopek iskanja rešitve MDS poteka v naslednjih korakih:

1. Najprej izberemo **začetno postavitev točk** $\mathbf{z}_1, \ldots, \mathbf{z}_n$, na primer naključno v ravnini. V praksi mnogokrat izhajamo iz postavitve točk, ki jo pridobimo z izračunom glavnih komponent.
2. Nato postopoma **izboljšujemo postavitev** tako, da premikamo točke $\mathbf{z}_i$ v smeri, ki zmanjša vrednost kriterijske funkcije. Za ta namen uporabimo **gradientni spust** ali druge optimizacijske algoritme (npr. iterativno večkratno prilagajanje, SMACOF algoritem).
3. Pri gradientnem spustu računamo **gradient kriterijske funkcije** po vsaki koordinati $\mathbf{z}_i$, to pomeni, da za vsako točko izračunamo, v katero smer jo moramo premakniti, da bo kriterijska funkcija manjša.
4. Postopek ponavljamo, dokler se vrednost kriterijske funkcije ne neha pomembno zmanjševati, torej dokler ne dosežemo **konvergence**.

Problem MDS je pogosto zelo zahteven za računalniško izvedbo, ker število parov točk narašča kvadratno z $n$ — to pomeni, da za $n$ točk moramo primerjati $n(n-1)/2$ parov, kar je zelo veliko za večje množice podatkov. Zaradi te počasnosti so razviti tudi hitrejši ali aproksimativni algoritmi, ki pri iskanju rešitve upoštevajo le del razdalj (npr. razdalje do najbližjih sosedov), ali pa uporabljajo pohitrene metode, kot so stohastični gradientni spust, kjer se v vsakem koraku posodobi le naključna podmnožica točk. Ena izmed znanih pohitrenih različic je metoda SMACOF (angl. *Scaling by Majorizing a Complicated Function*), ki v vsakem koraku reši približek originalnega problema in tako hitreje konvergira.

Pomembno je poudariti, da MDS ni projekcija, ker nove koordinate točk niso linearne kombinacije originalnih atributov, temveč je vložitev v nov vektorski prostor. Nove dimenzije nimajo pomena, saj jih določimo samo zato, da čim bolje ohranimo razdalje. Poleg tega je rešitev invariantna na rotacije, zrcaljenja in premike — če bi vse točke zavrteli ali premaknili, bi

bila rešitev še vedno enakovredna, saj bi razdalje ostale enake.

Glavna razlika med vložitvijo in projekcijo je torej v tem, da pri projekciji (kot pri metodi glavnih komponent) nove osi nastanejo kot kombinacije obstoječih značilk in imajo včasih možno interpretacijo, medtem ko pri vložitvi (kot pri MDS) nove osi nimajo pomena in jih določimo izključno za vizualizacijo na podlagi ohranjanja razdalj.

Slabost metode MDS je, da poskuša ohraniti vse razdalje med pari podatkovnih točk, zato pogosto daje prevelik poudarek na ohranjanje velikih razdalj med oddaljenimi pari, namesto da bi se osredotočila na ohranjanje bližnjih sosedstev, ki so za razumevanje lokalne strukture podatkov pomembnejša. Posledično lahko MDS slabše prikaže lokalne gruče in sosednje primere.

Primer: če imamo dve točki, ki sta v originalnem prostoru zelo blizu skupaj, recimo na razdalji 1, in dve točki, ki sta zelo daleč narazen, recimo na razdalji 100, in če pri vložitvi vse razdalje odstopajo za 10 %, bo to pomenilo, da se razdalja 1 spremeni v 1,1, medtem ko se razdalja 100 spremeni v 110. Čeprav je napaka v odstotkih enaka, bo napaka pri veliki razdalji (10 enot) bistveno večja v absolutnem smislu kot pri majhni razdalji (0,1 enote). Zato bo MDS pri optimizaciji poskušal bolj popraviti velike napake pri oddaljenih točkah, pri čemer lahko zanemari točne razdalje med bližnjimi točkami, ki pa so pogosto najpomembnejše za razumevanje strukture podatkov.

# Ohranjenje soseščin (t-SNE)

Metoda t-SNE (angl. *t-distributed Stochastic Neighbor Embedding*) je namenjena iskanju dobre predstavitve podatkov, kjer bodo podatkovni primeri, ki so si podobni, blizu skupaj, in tisti, ki so si različni, daleč narazen. Glavna ideja t-SNE je, da se osredotoči na ohranjanje lokalne strukture podatkov, torej da sosednje točke v visoki dimenziji ostanejo sosednje tudi v dvodimenzionalni predstavitvi.

Metoda t-SNE najprej izračuna, kako verjetno je, da bi bili podatkovni pari sosedje v originalnem prostoru. Nato v nizki dimenziji postavitev podatkovne točk tako, da so verjetnosti sosedstva točk čim bolj podobne tistim v originalnem prostoru. Za razliko od metod, ki ohranjajo razdalje (kot je večrazredno lestvičenje), t-SNE ohranja sosedstva in je zato posebej primerna za odkrivanje gruč in lokalnih vzorcev v podatkih.

Matematično t-SNE deluje tako, da za vsak par podatkovnih točk $i$ in $j$ najprej določi

verjetnost $p_{ij}$, ki meri, kako blizu sta si točki v originalnem prostoru. To verjetnost izračunamo s pomočjo Gaussove porazdelitve:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

Parameter $\sigma_i$ določa, kako daleč segajo sosednje točke okoli podatkovne točke $\mathbf{x}_i$. Večja vrednost $\sigma_i$ pomeni, da bo več oddaljenih točk obravnavanih kot sosedje, manjša vrednost pa omeji sosedstvo na bližje točke. Da se $\sigma_i$ ustrezno nastavi za vsako točko, izberemo globalni parameter, imenovan *perplexnost* (*perplexity*), ki določa pričakovano število sosedov. Algoritem nato prilagodi $\sigma_i$ tako, da število sosedov, določenih s porazdelitvijo verjetnosti, približno ustreza izbrani perplexnosti. Perplexnost torej uravnava kompromis med lokalnimi in globalnimi odnosi: manjše vrednosti poudarijo majhne gruče, večje vrednosti pa zajamejo širšo strukturo podatkov.

V nizki dimenziji, kjer imamo točke $\mathbf{z}_1, \ldots, \mathbf{z}_n$, definiramo podobno verjetnost $q_{ij}$, ki meri, kako blizu sta si točki v 2D prostoru. Namesto Gaussove porazdelitve uporabimo Studentovo t porazdelitev z eno stopnjo prostosti (porazdelitev t-distribucije), ki omogoča večje razdalje in s tem bolj naravno razporeditev oddaljenih točk:

$$q_{ij} = \frac{(1 + \|\mathbf{z}_i - \mathbf{z}_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|\mathbf{z}_k - \mathbf{z}_l\|^2)^{-1}}$$

Glavna optimizacijska naloga je, da minimiziramo razliko med verjetnostmi $p_{ij}$ in $q_{ij}$. Za merjenje te razlike uporabimo Kullback-Leibler (KL) divergenco:

$$\min_{\mathbf{z}_1,\ldots,\mathbf{z}_n} KL(P\|Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Kriterijska funkcija torej meri, kako daleč so verjetnosti sosedstva v 2D od tistih v originalnem prostoru. Manjša kot je ta razlika, boljše je ujemanje med 2D in originalnim prostorom.

Reševanje problema poteka z iterativno numerično optimizacijo, najpogosteje z gradientnim spustom. V vsakem koraku izračunamo gradient kriterijske funkcije po koordinatah točk $\mathbf{z}_i$ in jih ustrezno popravimo, da se zmanjša vrednost kriterija. Zaradi narave kriterijske funkcije in velikega števila parov točk je t-SNE računsko zahteven in razmeroma počasen, še posebej za

velike podatkovne množice. Da bi to težavo zmanjšali, obstajajo različne pohitrene različice, kot sta Barnes-Hut t-SNE in optimizacija z uporabo stohastičnih metod.

Pomembno je znova razumeti, da t-SNE ni projekcija, temveč vložitev v nov vektorski prostor. Nove dimenzije ne predstavljajo nobene linearne kombinacije originalnih značilk in nimajo posebnega pomena. t-SNE poišče takšno postavitev točk, ki je glede sosedstev najbolj podobna originalni strukturi podatkov. Ker gre za vložitev, je rešitev, tako kot pri večdimenzionalnem lestvičenju, invariantna na rotacije, zrcaljenja in translacije.

Pri uporabi t-SNE je pomembno, da točke v vložitvi opazujemo predvsem s stališča njihove soseščine. To pomeni, da je smiselno gledati, katere točke so blizu skupaj, saj t-SNE ohranja predvsem lokalne odnose. Oddaljenost med bolj oddaljenimi gručami pa ni nujno sorazmerna z njihovo dejansko oddaljenostjo v originalnem prostoru. Dve gruči, ki sta v vložitvi zelo narazen, bi lahko bili v resnici bližje, kot kaže slika, ali pa bi bili še bolj oddaljeni. Zato pri interpretaciji rezultatov t-SNE ne smemo razlagati razdalj med gručami, temveč le notranjo strukturo posameznih gruč in njihova sosedstva.

# Vložitve podatkov na osnovi mnogoterosti in grafov (UMAP)

Metoda UMAP (angl. *Uniform Manifold Approximation and Projection*) temelji na teoriji mnogoterosti (manifolds) in teoriji grafov in je namenjena iskanju predstavitve podatkov, kjer bodo ohranjene tako lokalne značilnosti (sosedstva) kot tudi globalna struktura, kot so medgručne razdalje. V primerjavi s t-SNE je UMAP morda manj občutljiv na parametre in bolje ohranja splošno strukturo podatkov.

Osnovna ideja metode je, da v visoki dimenziji najprej zgradi graf sosedstev, ki opisuje lokalne povezave med podatkovnimi točkami, nato pa v nizki dimenziji išče takšno postavitev točk, ki ohranja ta sosedstva.

Matematično UMAP najprej določi verjetnosti $p_{ij}$, ki merijo, kako močno je točka $j$ povezana s točko $i$ v visokodimenzionalnem prostoru. Te verjetnosti so definirane z uporabo razdalj in funkcije:

$$p_{ij} = \exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i}{\sigma_i}\right)$$

kjer je $d(\mathbf{x}_i, \mathbf{x}_j)$ razdalja med točkama, $\rho_i$ minimalna razdalja do prvega soseda (da se zagotovi lokalna prilagoditev), $\sigma_i$ pa širina območja sosedstva, ki se določi tako, da povprečno število sosedov ustreza željeni perplexnosti. V nizki dimenziji (npr. 2D) UMAP določi podobne verjetnosti $q_{ij}$, ki opisujejo povezave med točkami v novi predstavitvi:

$$q_{ij} = \frac{1}{1 + a\|\mathbf{z}_i - \mathbf{z}_j\|^{2b}}$$

kjer sta $a$ in $b$ parametra, ki določata obliko funkcije in se izbereta tako, da čim bolje modelirata porazdelitev razdalj.

Kriterijska funkcija, ki jo UMAP minimizira, je podobna divergenci Kullback-Leibler, a temelji na binarni križni entropiji med verjetnostmi $p_{ij}$ in $q_{ij}$:

$$C = \sum_{i \neq j} \left[ p_{ij} \log q_{ij} + (1 - p_{ij}) \log(1 - q_{ij}) \right]$$

S to funkcijo UMAP poišče postavitev točk tako, da bodo pari točk, ki so povezani v visokodimenzionalnem prostoru, tudi blizu skupaj v nizki dimenziji, in pari, ki niso povezani, daleč narazen. Podobno, kot pri t-SNE, optimizacijo izvajamo z gradientnim spustom (o tem več v naslednjih poglavjih), kjer se v vsakem koraku popravljajo koordinate točk v nizki dimenziji, da se zmanjša vrednost kriterija.

UMAP je torej metoda, ki omogoča hitro in učinkovito vložitev podatkov v nizko dimenzijo, pri čemer ohranja tako lokalne kot delno tudi globalne strukture, in je zaradi tega primerna za vizualizacijo velikih in kompleksnih podatkovnih množic.

# Drugi pristopi

Poleg zgoraj opisanih obstaja danes še velik razred vložitvenih pristopov, ki temeljijo na nevronskih mrežah. Glavna prednost teh je, da se lahko učimo vložitev za poljubne vrste vhodnih podatkov, ne le za tiste, ki jih lahko opišemo s tabelaričnimi značilkami. Tako lahko s pomočjo nevronskih mrež izdelamo vložitve tudi za slike, besedila, zvok in kompleksne kombinacije teh podatkov.

Vložitev se v teh primerih nevronska mreža nauči kot rezultat učenja predstavitve podatkov. Nevronska mreža preslika vhodne podatke v vektorski prostor nizke ali poljubno določene dimenzije. Ker so to parametrične metode, jih lahko po učenju uporabimo tudi za nove, še

nevidene podatke, ali pa celo sestavimo nevronsko mrežo tako, da ta lahko generira nove podatke.

Velika prednost teh pristopov je, da lahko poljubno določimo kriterijsko funkcijo, s katero določimo, kaj naj bo vložitev — na primer, da naj bodo podobni podatki blizu skupaj, različni pa daleč narazen, ali pa da nam nevronska mreža dobro loči med podatki različnih razredov. V praksi pogosto uporabljamo kontrastne funkcije, ki učijo, da so pari podobnih primerov blizu, pari različnih primerov pa daleč. Primeri takih pristopov so avtokodirniki, ki stisnejo podatke v vmesno plast z manjšim številom dimenzij, ter metode na osnovi konrastnega učenja, kjer se vložitev uči na osnovi primerjav med pari ali trojicami podatkov.

A o nevronskih mrežah, avtokodirnikih in podobnih strojih in napravah še nismo govorili in bo za to v naslednjih poglavjih še čas. Tu le spomnimo, da metode, ki jih omenjamo zgoraj, niso edine in so danes skoraj klasične, čeprav sta t-SNE in UMAP med nedavno predlaganimi. Ali bodo vse te izginile in jih bodo nadomestile samo tehnike, ki bodo zgrajene na osnovi nevronskih mrež pa bo pokazal čas.

# Razlaga podatkovnih kart

O tej temi, razen čisto malo pri metodi glavnih komponent, sploh nismo govorili. A je razlaga nadvse pomembna in je ta pravzaprav tista, zaradi katerih podatkovne karte sploh gradimo. Čakajoč pisce teh zapiskov, da te opremijo s primeri (tudi s temi, ki smo jih obravnavali na predavanjih), tu povejmo le, da je razlaga kart podobna razlagi skupin, ki smo jo opisali v prejšnjem poglavju: izberemo skupino točk in za to ugotovimo, katere značilke in njihove vrednosti so ji lastne in kako te ločijo izbrano skupino od ostalih primerov. Idealno bi sicer bilo, da nam take skupine in njihovo razlago poiščejo algoritmi avtomatično ter pri tem ugotovitve povzamejo, na primer z uporabo velikih jezikovnih modelov, a tovrstnih programov, kjer bi to delovalo tako, kot smo na kratko opisali, (skoraj) ni in je prilonosti za njihov razvoj še veliko.