Klasteriranje k-sredinama

Apolonio, Petar

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Petar Apolonio

KLASTERIRANJE K - SREDINAMA

Diplomski rad

Voditelj rada:
Prof. dr. sc. Zlatko Drmač

Zagreb, Ožujak, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Od početka razvijanja IT sektora u svijetu, usporedno se provodila digitalizacija raznih procesa oko nas. Podaci su se skupljali velikom brzinom, te je obrada skupljenih podataka postala znatno teža. Kako bi se uočila slična svojstva među podacima i otkrile neke nove njihove karakteristike, prirodno se pojavila potreba za klasteriranjem. Samim time je i obrada podataka postala lakša i brža.

Jedan od prvih algoritama za klasteriranje podataka je klasteriranje k - sredinama. Iako je ravijen prije više od 50 godina, zbog svoje jednostavnosti, dobrih rezultata i lake implementacije se još uvijek često koristi. Danas skoro da i ne postoji područje u kojem se podaci ne klasteriraju, počevši od biomedicine, obrade slika i tekstualnih dokumenata, pa do svakodnevnog praćenja i klasteriranja kupaca prema njihovim potrebama. Algoritam k - sredina se razvijao kroz vrijeme, te nudi različite varijacije koji se koriste obzirom na strukturu i veličinu podataka, te potrebe korisnika.

Navedeni algoritam je glavna tema ovoga rada koji je podijeljen u tri poglavlja. U prvom poglavlju se opisuje klaster analiza, osnovni pojmovi i glavna podjela algoritama klasteriranja. U drugom poglavlju je prikazan algoritam k - sredina u svom osnovnom obliku, te njegove najkorištenije varijacije. U trećem poglavlju su dani primjeri obrade slike i usporedba osnovnog algoritma k - sredina sa jednom njegovom varijacijom - algoritmom jezgrenih k - sredina. Na kraju rada se nalazi dodatak s kodovima korištenim u izradi trećeg poglavlja, napisani u programskom okruženju MATLAB.

Poglavlje 1

Klaster analiza

Klaster analiza predstavlja statističku tehniku kojom se utvrđuju relativno homogene grupe objekata, tj za dani skup podataka U treba odraditi k podskupova (klastera) C_i , $i = 1, 2, ..., k$ koji su homogeni i/ili dobro separirani u odnosu na mjerne varijable. Elementi pojedinog klastera C_i su sličniji jedan drugome, nego elementima izvan tog klastera. Koristi se u mnogim područjima kao što su strojno učenje, prepoznavanje uzoraka, analiza slika, bioinformatika i sl. Skup $\mathcal{Z} = \{ C_1, C_2, ..., C_k \}$ zovemo klastering. Klastering može biti particija skupa.

Definicija 1. *Particija Skupa*

Neka je X skup elemenata, $|X| \geq 2$ *i* $2 \leq k \leq |X|$ *prirodan broj. Skup* $\{C_1, C_2, ..., C_k\}$ *podskupova od X za koje vrijedi:*

1. $\bigcup_{i=1}^{k} C_i = X$
2. $i \neq j \Rightarrow C_i \cap C_j = \emptyset$, $i, j = 1, 2, ..., k$
3. $|C_i| \neq 0$, $i = 1, 2, ..., k$

zovemo particija skupa X, a C_i , $i = 1, 2, ..., k$, *klasterima u skupu X*

Ukoliko klastering $\mathcal{Z} = \{ C_1, C_2, ..., C_k \}$ ima svojstvo da za bilo koja njegova dva elementa vrijedi da je jedan od njih sadržan u drugom, tj.

$$(\forall i = 1, 2, ..., k),\ (\forall j = 1, 2, ..., k),\ C_i \cap C_j \in \{C_i, C_j\},$$

$\mathcal{Z}$ nazivamo hijerarhijom. Kada su skup U i svi njegovi jednočlani podskupovi sadržani u klasteringu $\mathcal{Z}$, odosni vrijedi:

$$\forall\, x \in U, \{x\} \in \mathcal{Z} \quad \text{i} \quad U \in \mathcal{Z},$$

$\mathcal{Z}$ zovemo potpunom hijerarhijom.

Problem klasteriranja se može shvatiti i kao problem optimizacije na način da trebamo naći klastering $\mathcal{Z}^*$ za koji vrijedi :

$$P(\mathcal{Z}^*) = \min_{\mathcal{Z}\in\Theta} P(\mathcal{Z})$$

gdje je $P : \Theta \rightarrow$R funkcija kriterija, a Θ skup svih mogućih klasteringa na skupu U.

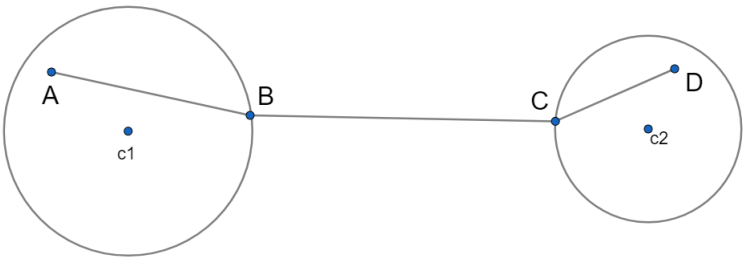
Koraci klasterske analize su [11]:

1. određivanje ciljeva klaster analize,
2. određivanje istraživačkog obrasca,
3. određivanje pretpostavki,
4. formiranje i procjena broja klastera,
5. interpretacija klastera,
6. procjena klaster analize i profiliranje klastera

1.1 Vrste klastera

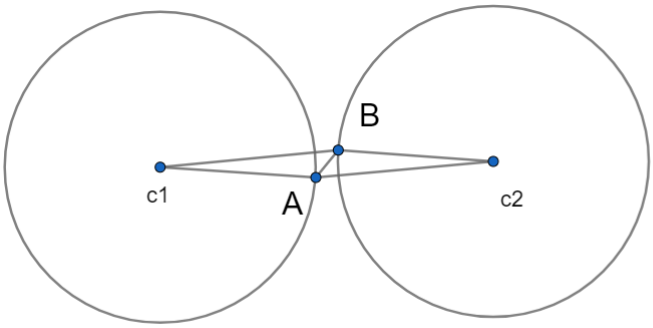
Klasteri mogu biti [12]:

1. Potpuno odvojeni klasteri - Svaka točka je bliža svim točkama svog klastera, nego bilo kojoj drugoj točki



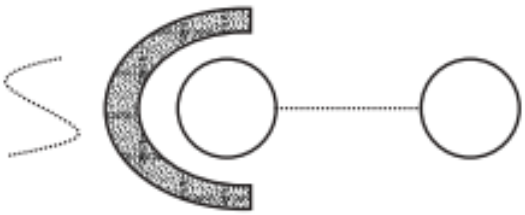
Slika 1.1: Potpuno odvojeni klasteri

2. Klasteri zasnovani na centrima - Svaka točka je bliža centru svog klastera, nego bilo kojem drugom centru



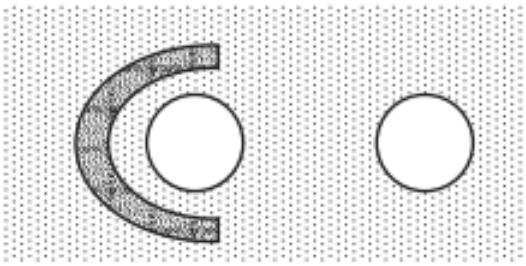
Slika 1.2: Klasteri zasnovani na centrima

3. Klasteri zasnovani na susjedstvu - Svaka točka je bliža najmanje jednoj točki unutar svog klastera, nego bilo kojoj točki iz drugih klastera



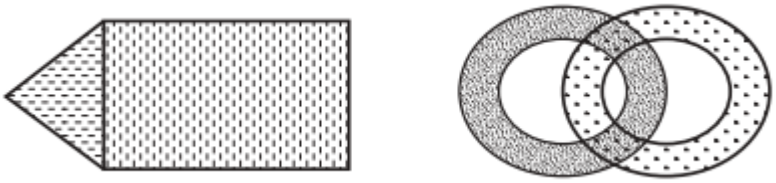
Slika 1.3: Klasteri zasnovani na susjedstvu

4. Klasteri zasnovani na gustoći - Klasteri su područja visoke gustoće, odvojeni područjima male gustoće



Slika 1.4: Klasteri zasnovani na gustoći

5. Konceptualni klasteri - Točke unutar klastera dijele neko općenito svojstvo koje potječe iz cijelog skupa podataka (točke mogu biti u više klastera)



Slika 1.5: Konceptualni klasteri

1.2 Vrste algoritama klasteriranja

Obzirom na veći broj algoritama klasteriranja, postavlja se pitanje koji algoritam odabrati za konkretan problem. Nužno ne vrijedi činjenica da, ako je algoritam bio dobar na jednom skupu pdataka, da će biti učikovit i na nekom drugom skupu podataka koji ima različita svojstva. Dobar algoritam ne bi trebao biti osjetljiv na količinu podataka i broj varijabli. Redoslijed unosa podataka je također nešto na što treba paziti zato što je moguće da za isti skup podataka i isti algoritam, obzirom na redoslijed unosa podataka, dobijemo različite rezultate. U konačnici, dobar algoritam bi se svakako trebao dobro nositi sa podacima koji nisu strogo nisko varijabilni, već postoje podaci koji odstupaju od prosjeka (tzv. *outlieri*).

Hijerarhijsko klasteriranje

Hijerarhijsko klasteriranje, poznato još kao i ugniježđeno klasteriranje, je vrsta klasteriranja koje stvara (aglomerativni način) ili prekida (divizivni način) hijerarhiju klastera. Hijerarhija klastera ima oblik stabla, tj. na jednom kraju stabla je klaster koji sadrži sve podatke, dok se na drugom kraju nalaze klasteri sa pojedinačnim podacima. Stablo koje prikazuje hijerarhijsko klasteriranje zove se dendrogram. Dalje u radu ćemo sa $D(\cdot,\cdot)$ označavati udaljenost među dva klastera, dok sa $d(\cdot,\cdot)$ udaljenost među dvije točke.

Algoritam 1 Algoritam aglomerativog načina hijerarhijskog klasteriranja

•$INPUT$: skup podataka $\mathcal{X}$, funkcija udaljenosti među dva klastera D, funkcija udaljenosti među dvije točke d

neka svaki $x_i \in \mathcal{X}$ je sam u svom klasteru C_i

dok god postoji dva ili više klastera ponavljaj :

1. nađi dva klastera C_i i C_j sa najmanjoj udaljenosti, tj. i i j za koje vrijedi: $D(C_i, C_j) = min_{k\neq r} D(C_k, C_r)$
2. klastere C_i i C_j spoji u novi klaster, tj. $C_s = C_i \cup \mathrm{C}_j$
3. C_s dodaj u skup svih klastera, a C_i i C_j izbaci iz tog skupa

•$OUTPUT$: jedan klaster sa svim elementima skupa $\mathcal{X}$

Za potrebe računanja udaljenosti među višečlanim klasterima A i B mogu se koristiti jedna od sljedećih udaljenosti:

1. Maksimum (eng. $complete - linkage\ clustering$) :

$$D(A, B) = max\{d(a, b) : a \in A,\ b \in B\}.$$

   Navedena udaljenost između dva klastera se definira kao udaljenost između dva najudaljenija elementa tih klastera. To su zapravo dva najrazličitija elementa između navedena dva klastera.

2. Minimum (eng. $single - linkage\ clustering$)

$$D(A, B) = min\{d(a, b) : a \in A,\ b \in B\}.$$

Navedena udaljenost između dva klastera se definira kao udaljenost između dva najbliža elementa tih klastera. To su zapravo dva najsličnija elementa između navedena dva klastera.

3. Metoda prosjeka (eng. *Mean or average linkage clustering*)

$$D(A, B) = \frac{1}{|A|\,|B|} \sum_{a \in A} \sum_{b \in B} d(a, b).$$

.

Udaljenost između klastera A i B je prosjek udaljenosti među svim mogućim parovima elemenata klastera A i B.

4. Centroid - metoda (eng. *Centroid linkage clustering*)

$$D(A, B) = \|c_A - c_B\|,$$

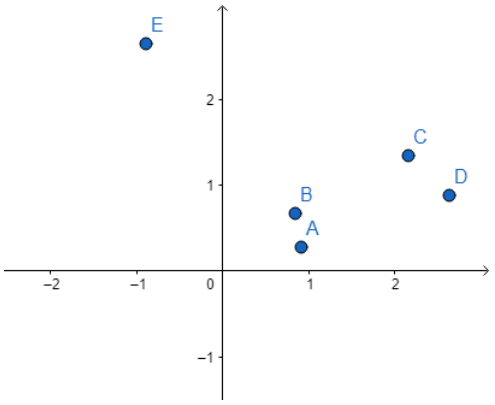
gdje su c_A i c_B centri klastera A i B, a $\|\cdot\|$ predstavlja bilo koju normu.

5. Wardova metoda [3]

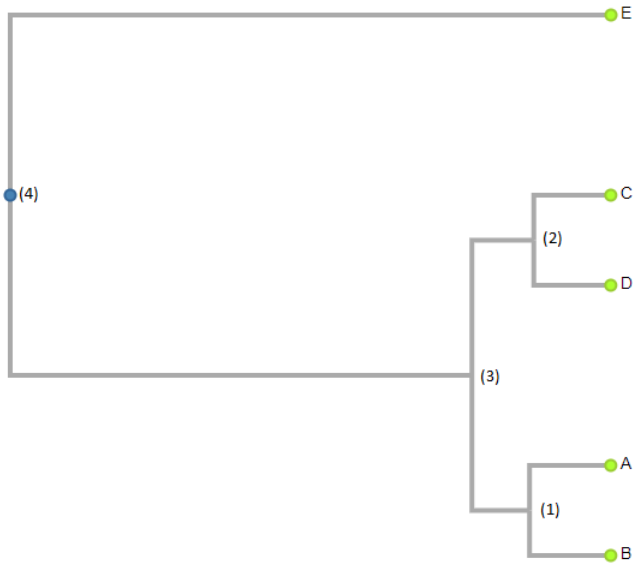
$$D(A, B) = \frac{n_A n_B}{n_A + n_B} \|c_A - c_B\|.$$

Kod ove moteode n_A i n_B označavaju broj elemenata klastera A, odnosno B. Elementi se povezuju u klaster tako da varijanca unutar klastera bude minimalna, odnosno maksimizira se homogenost unutar klastera. $\|\cdot\|$ kao i u centroid - metodi predstavlja bilo koju normu.

Zbog korištenja norme u Wardovoj metod i cetroid - metodi, da bi ih mogli koristiti, bitno je da nam podaci dolaze iz unitarnog prostora. Kao primjer aglomerativnog hijerarhijskog klasteriranja ćemo dati skup na slici 1.6 i klasteriranje na slici 1.7.



Slika 1.6: Skup podataka



Slika 1.7: Dendrogram skupa sa slike (1.6)

Na Slici 1.7 vidimo prikaz aglomerativnog hijerarhijskog klasteriranja. Na početku imamo 5 klastera, te svaki sadrži točno jedan element skupa. U prvom koraku želimo spojiti elemente u veće klastere. Uobičajena je praksa da se uzimaju dva najbliža elementa. U ovom primjeru, kao udaljenost između dvije točke koristila se Euklidska udaljenost, a za

udaljenost između dva klastera koristila se udaljenost minimuma(2). Sa slike (1.6) vidimo da su A i B dvije najbliže točke našeg skupa, te je na dendrogramu čvor u kojem se spajaju navedene točke označen s (1). Dalje je jasno da u sljedećem koraku treba spojiti točke C i D. Udaljenost između njih je manja nego udaljenost bilo koje od tih točaka prema točki E ili klasteru {A,B}. U trećem koraku se spajaju klasteri {A,B} i {C,D}, a u četvrtom se svi elementi spoje u jedan klaster, kako je i označeno na dendrogramu. Mjere udaljenosti između dva elementa i između dva klastera se proizvoljno odaberu.

Divizivno klasteriranje je obrnuto od aglomerativnog. Krećemo sa svim elementima u istom klasteru, te se klaster rekurzivno razbija na manje. Gotovi smo kada je svaki element u različitom klasteru ili kada je zadovoljen kriterij zaustavljanja. Definiramo način na koji računamo kvalitetu klastera i toleranciju, te ukoliko je kvaliteta klastera iznad tolerancije, tada takav klaster nema smisla više dijeliti nego ga ostavljamo u obliku koji je. Divizivno klasteriranje je kompleksnije od aglomerativnog načina. Njegova je prednost to što je točniji ukoliko ne moramo raspisati stablo sve do jednočlanih klastera. U tom slučaju daje bolju hijerarhiju. Također, aglomerativni način nema u sebi informaciju o distribuciji cijelog skupa, dok divizivni posjeduje, što u konačnici može rezultirati puno boljim rješenjem. Najpoznatiji algoritmi divizivnog hijerahijskog klasteriranja su DIANA i MONA. [1] Algoritam DIANA se može koristiti u svim prilikama, dok algoritam MONA je prigodan kada imamo binarne podatke. U svakom koraku algoritma DIANA, gleda se klaster C^* s najvećim radijusom $r(C^*)$. Radijus klastera definiramo kao najveću udaljenost između dva elementa tog klastera. Da bi podijelio taj klaster u nova dva klastera, algoritam prvo pronalazi jedan njegov element koji se najviše razlikuje od ostatka klastera. Npr. to može biti element x koji ima najveću prosječnu udaljenost $s(x)$ od ostatka klastera i njega zovemo djelitelj klastera. Prosječnu udaljenost računamo na način:

$$\text{za bilo koji } x_i \text{ iz klastera } C = \{x_1, x_2, ..., x_m\} \text{ imamo } s(x_i) = \frac{\sum_{k \neq i} d(x_i, x_k)}{m - 1}$$

gdje nam $d(x_i, x_k)$ označava Euklidsku udaljenost dvije točke.
Tada za svaki element koji nije djelitelj klastera računamo udaljenost do djelitelja klastera i udaljenost do dijela klastera u kojem se ne nalaze promatrani element i djelitelj klastera. Ukoliko udaljenost promatranog elementa do djelitelja klastera bude manja, tada taj element izdvajamo iz postojećeg i dodjeljujemo novom klasteru u kojem se već nalazi djelitelj klastera.

Algoritam 2 DIANA

•$INPUT$: skup pdoataka X, mjera udaljenosti d, tolerancija, mjera kvalitete klastera q neka su svi elementi $\{x_1, x_2, ..., x_n\}$ u istom klasteru

dok god postoji bar jedan klaster s dva ili više elemenata ponavljaj :

1. izračunaj kvalitetu svih postojećih klastera
2. Pronađi klaster $C = \{x_1, x_2, ..., x_m\}$ s najvećim radijusom $r(C)$ čija kvaliteta ne prelazi toleranciju
3. Za svaki element tog klastera x_i izračunaj prosječnu udaljenost od ostatka klastera $s(x)$
4. Odaberi element x_{i^*} s najvećom prosječnom udaljenosti od ostatka klastera
5. Za svaki element kalstera $x_i \neq x_{i^*}$ izračunaj udaljenost $d(x_i, x_{i^*})$ te $d(x_i, \{x_j : j \neq i^*, j \neq i\})$
6. Sve elemente x_i za koje vrijed $d(x_i, x_{i^*}) \leq d(x_i, \{x_j : j \neq i^*, j \neq i\})$ skupa sa x_{i^*} prebaci u novi klaster C^*

•$OUTPUT$: broj klastera, vektor u kojem je zapisano za svaku točku kojem klasteru pripada

Primjerom ćemo pokazati rad algoritma. Mjera kvalitete klastera nije definirana, kako bi demonstrirali metodu razdvajanja klastera do jednočlanih klastera.

Imamo skup točaka $\{A, B, C, D, E\}$ čije su meuđusobne udaljenosti dane tablicom (1.1). Vidimo da točka E ima najveću prosječnu udaljenost od ostatka klastera ($s(E) = 3$), te je on naš djelitelj klastera u ovoj iteraciji. Nadalje, računamo udaljenosti točaka A-D do dijela klastera u kojem se ne nalaze promatrana točka i točka F.

Tablica 1.1: Tablica udaljenosti

	A	B	C	D	E	S
A	0	1	2	3	4	2.5
B	1	0	3	1	5	2.5
C	2	3	0	2	2	2.25
D	3	1	2	0	1	1.75
E	4	5	2	1	0	3

Na tablici (1.2) vidimo da su točke C i D bliže točki F nego ostatku klastera. Za računanje udaljenosti među klasterima smo koristili udaljenost maksimum (1) Dakle, iz postojećeg klastera $\{A, B, C, D, E\}$ dobijemo dva klastera $\{A, B\}$ i $\{C, D, E\}$.

Tablica 1.2: Podjela klastera {A,B,C,D,E}

	A-D	E
A	3	4
B	3	5
C	3	2
D	3	1

Iz tablice udaljenosti (1.1) vidimo da $r(\{A, B\}) = d(A, B) = 1 \leq r(\{C, D, E\}) = d(C, D) = 2$. Stoga dalje promatramo klaster $\{C, D, E\}$.

Tablica 1.3: Tablica udaljenosti za {C,D,E}

	C	D	E	s
C	0	2	2	2
D	2	0	1	1.5
E	2	1	0	1.5

Tablica 1.4: Podjela klastera {C,D,E}

	D-E	C
D	1	2
E	1	1

U navedenom klasteru izdvaja se točka C koju postavljamo za djelitelja klastera. Na tablici (1.4) vidimo da u novi klaster možemo premjestiti točku C, a točku E možemo ostaviti u klasteru s točkom E ili premjestiti skupa s točkom C u novi klaster. Neka točka E ostane s točkom D u istom klasteru, dakle klaster {C,D,E} se dijeli na klastere {C} i {D,E}.
U idućem koraku nastavljamo s {D,E} zato što imamo slučaj $r(\{A, B\}) = r(\{D, E\}) = 1$, pa možemo uzeti bilo koji klaster. Nakon što radvojimo klaster {D,E}, odradimo isto i sa klasterom {A,B}. Time smo došli do kraja, tj. skup {A,B,C,D,E} smo razdvojili u 5 jednočlanih klastera.

Tablica 1.5: Tablica udaljenosti {D,E}

	D	E
D	0	1
E	1	0

Tablica 1.6: Tablica udaljenosti {A,B}

	A	B
A	0	1
B	1	0

Particijsko klasteriranje

Particijsko ili neugniježđeno klasteriranje je podjela skupa podataka u disjunktne podskupove (klastere) na način da je jedan element skupa u točno jednom podskupu. Broj klastera mora biti unaprijed određen, te oni ne mogu biti prazni. Većina algoritama particijskog klasteriranja minimizira funkciju cilja, te oni rade dobro kada klastere možemo linearno odvojiti. Loša strana algoritama je velika osjetljivost na šumove i stršeće vrijednosti (tzv. outliere). Također ne treba ni zaboraviti problem pronalaska optimalnog broja klastera. Najpoznatiji algoritmi particijskog klasteriranja su klasteriranje k - sredinama i klasteriranje k - medoidima. Klasteriranje k - medoidima umjesto centroida koristi medoide, tj. objekte iz skupa koji predstavljaju određeni klaster. Najpoznatiji algoritam koji koristi ovu metodu zove se *Particioniranje oko medoida* (*eng*. PAM - Partitioning around medoids).

Algoritam 3 PAM

•$INPUT$: broj klastera k, skup podataka X, funkcija udaljenosti, funkcija kvalitete

1. Proizvoljno odaberi iz skupa k elemenata za početne medoide
2. Preostale elemente raspodijeli po klasterima tako da udaljenost elementa prema medoidu dodijeljenog klastera bude manje nego prema drugim medoidima
3. Zamijeni svaki od medoida s jednim od preostalih objekata sve dok se kvaliteta klasteriranja ne poveća
4. Iteriraj do konvergencije

•$OUTPUT$: particija skupa X, medoid svakog klastera

U Algoritmu (3) korisnik sam bira funkciju udaljenosti, te funkciju kvalitete klastera. Klasteriranje k - sredinama je jedan od najpoznatijih algoritama za klasteriranje. U sljedećem poglavlju ćemo objasniti algoritam i dati neke njegove varijacije.

Poglavlje 2

K - sredine

Klasteriranje k - sredinama jedan je od najkorištenijih algoritama klasteriranja. Iako ga je James MacQueen predstavio još 1967. godine, zbog svoje jednostavnosti i dobrih rezultata, algoritam se i danas često koristi. On je vrsta nenadziranog učenja, te se koristi kod obrade neobilježenih podataka, tj podataka koji koji nemaju definirane kategorije. Glavni cilj ovog algoritma je podjela podataka u k grupa. Algoritam radi iterativno, te kao rezultat daje k grupa i njihove predstavnike.

2.1 Osnovni algoritam k - sredina

Stuart P. Lloyd je još 1957. godine predstavio algoritam koji particionira skup u manje klastere, računa njihove centre, te onda razmješta elemente skupa po klasterima ovisno o njihovoj udaljenosti od centara klastera. Taj algoritam je vrlo sličan osnovnom algoritmu k - sredina, pa iz tog razloga se danas u literaturi osnovni algoritam k - sredina još zove i Lloydov algoritam [4].

Neka je $X = \{x_1, x_2,...,x_n\}$ skup podataka iz d - dimenzionalnog Euklidskog prostora $\mathbb{R}^d$. Cilj algoritma je particionirati skup X na k - klastera. Varijabla k mora biti unaprijed zadana. Algoritam svakom klasteru dodijeljuje predstavnika (centar) na način:

$$\text{za klaster } C_i \text{ , } i \in \{1, 2, ..., k\} \textit{ imamo}$$

$$c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x \ . \tag{2.1}$$

Kao što vidimo, centar klastera je zapravo i samo središte klastera, ali ne mora biti i njegov element. Također, svakoj izračunatoj particiji C pridružujemo i vrijednost ciljne funkcije F. Osnovni algoritam k - sredina koristi Euklidsku metriku na $\mathbb{R}^d$:

$$F(C) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - c_i\|_2^2 \ . \tag{2.2}$$

Cilj algoritma nam je pronaći idealnu particiju tako da funkcija cilja ima minimalnu vrijednost. Ako bolje pogledamo, vidimo da zapravo minimiziramo varijancu klastera, tj tražimo najhomogenije klastere.

Algoritam 4 Osnovni algoritam k - sredina

• $INPUT$: broj klastera k, maksimalan broj iteracija, tolerancija, skup podataka X

1. inicijaliziramo početne centre c_i , $i = 1, 2, ..., k$
2. za svaki $x \in X$ računamo vrijednost $sk(x) \in \{1, 2, ..., k\}$ (redni broj klastera kojem dodijeljujemo x) , tj. pronađemo najbliži centar c_i , te pridružimo x klasteru C_i

   Za računanje udaljenosti među točkama koristimo Euklidsku udaljenost:

   $$d(p, q) = d(q, p) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}\ , \quad p = (p_1, p_2, ..., p_n),\ q = (q_1, q_2, ..., q_n) \in \mathrm{R}^n$$

3. za svaki klaster C_i izračunamo njegov novi centar c_i pomoću formule (2.1)
4. izračunamo vijednost funkcije cilja F trenutne particije pomoću formule (2.2)
5. ponavljamo korake 3, 4 i 5 do konvergencije

• $OUTPUT$: skup centara $\{c_1, c_2, ..., c_k\}$, vektor članstva, vrijednost funkcije cilja F

Vektor članstva iz Algoritma (4) je vektor u koji se za svaku točku $x \in X$ spremi vrijednost $sk(x)$.

Algoritam je iterativan, te staje ukoliko ograničimo maksimalni broj iteracija ili postavimo neki kriterij zaustavljanja. Na početku algoritma moramo zadati parametar tolerancije koji nam služi u kriteriju zaustavljanja. U svakoj iteraciji i računamo vrijednost funkcije cilja $F(C^{(i)})$ za izračunatu particiju skupa. Ukoliko je $|\ F(C^{(i)}) - F(C^{(i-1)})|$ manja od zadane tolerancije, algoritam se zaustavlja. Navedeni kriterij zaustavljanja je jedan od najkorištenijih kriterija u Lloyd-ovom algoritmu. Vidimo da u koracima 3. i 4. Algoritma (4) svakom iteracijom dobivamo homogenije klastere. Samim time se smanjuje i funkcija cilja. Kako je funkcija cilja nenegativna i monotono opada zaključujemo da je konvergencija funckije cilja zajamčena. Postoji mogućnost da za rješenje dobijemo lokalni

minimum, koji nije nužno najbolje rješenje. Iz tog razloga se preporuča da se algoritam na istom skupu podataka pokrene više puta sa različitim inicijalnim centrima.

Teorem 2.1.1. *Vrijednost funkcije cilja F algoritma k - sredina monotono opada.*

Dokaz. Neka je $F^{(s)}$ vrijednost funkcije cilja (2.2) u s - toj iteraciji.

$$\mathrm{F}^{(s)} = F(C^{(s)}) = \sum_{i=1}^{k} \sum_{x \in C_i^{(s)}} \|x - c_i^{(s)}\|_2^2 \geq \sum_{i=1}^{k} \sum_{x \in C_i^{(s)}} \|x - c_{sk(x)}^{(s)}\|_2^2 =$$
$$= \sum_{i=1}^{k} \sum_{x \in C_i^{(s+1)}} \|x - c_{sk(x)}^{(s)}\|_2^2 \geq \sum_{i=1}^{k} \sum_{x \in C_{i+1}^{(s)}} \|x - c_i^{(s+1)}\|_2^2 = F(C^{(s+1)}) = F^{(s+1)}$$

Prva nejednakost vrijedi zbog toga što minimiziramo $\|x - c_i^{(s)}\|_2^2$ pridružujući točku x nekom bližem centru. Pokažimo da vrijedi i druga nejednakost:
za proizvoljan $i \in \{1, 2, ..., k\}$ želimo naći točku $y = (y_1, y_2, ..., y_d)$ koja minimizira izraz $\sum_{x \in C_i^{(s+1)}} \|x - y\|_2^2$

$$\min_y \sum_{x \in C_i^{(s+1)}} \|x - y\|_2^2 = \min_y \sum_{x \in C_i^{(s+1)}} \Big(\langle x\,,\ x \rangle - 2\langle x\,,\ y \rangle + \langle y\,,\ y \rangle \Big)\ .$$

Deriviramo li izraz parcijalno po l-toj dimenziji od y i dobiveni izraz izjednačimo s nulom, dobijemo sljedeći izraz [6]:

$$\sum_{x \in C_i^{(s+1)}} 2y_l \ - \ 2 \sum_{x \in C_i^{(s+1)}} x_l = 0 \quad , t\,j.$$

$$y_l = \frac{1}{|C_i|} \sum_{x \in C_i^{(s+1)}} x_l \ .$$

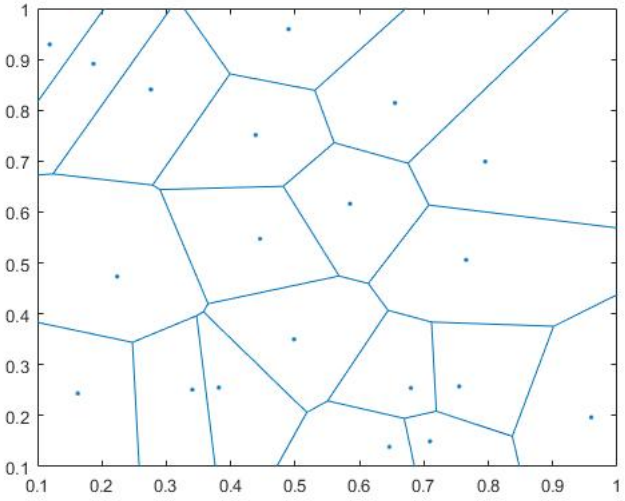
Vidimo da se vektor y po kordinatama podudara s novim centrom c_i izračunatim formulom (2.1). Kako je i bio proizvoljan, zaključujemo da tvrdnja vrijedi za svaki $i = 1, 2, ..., k$. Dakle, skup novih centara minimizira izraz $\sum_{i=1}^{k} \sum_{x \in C_i^{(s+1)}} \|x - c_{sk(x)}^{(s)}\|_2^2$, tj druga nejednakost vrijedi.

□

Možemo primjetiti da nam algoritam daje klastere koji su odvojeni hiperravninama. Npr. pogledajmo slučaj $k = 2$:
$x \in C_1 \Leftrightarrow \|x - c_1\|_2^2 \leq \|x - c_2\|_2^2$
Skup točaka koje su jednako udaljene od c_1 i c_2 dijele klastere C_1 i C_2 , a to je zapravo hiperravnina okomita na c_1 - c_2 koja prolazi polovištem c_1 - c_2 . Općenito, algoritam razdjeljuje elemente skupa hiperravninama, što odgovara Voronoievom dijagramu skupa predstavnika. Zbog toga se ovaj algoritam još zove i Vornoieva iteracija.

Definicija 2. *Neka je zadan skup* $S = \{ p_i \in R^n,\ i = 1, 2, ..., q \}$ *za neki* $q \in \mathbb{N}$. *Voronoieva ćelija (eng. Voronoi cell) pridružena točki* p_i *je skup*
$V(p_i) = \{x \in R^n : \|x - p_i\|_2 \leq \|x - p_j\|_2, \forall j = 1, 2, ..., q\}$.

Voronoiev dijagram skupa S je unija Voronoievih ćelija, odnosno $V(S) = \bigcup_{i \in \{1,2,...,q\}} V(p_i)$.



Slika 2.1: Voronoiev dijagram

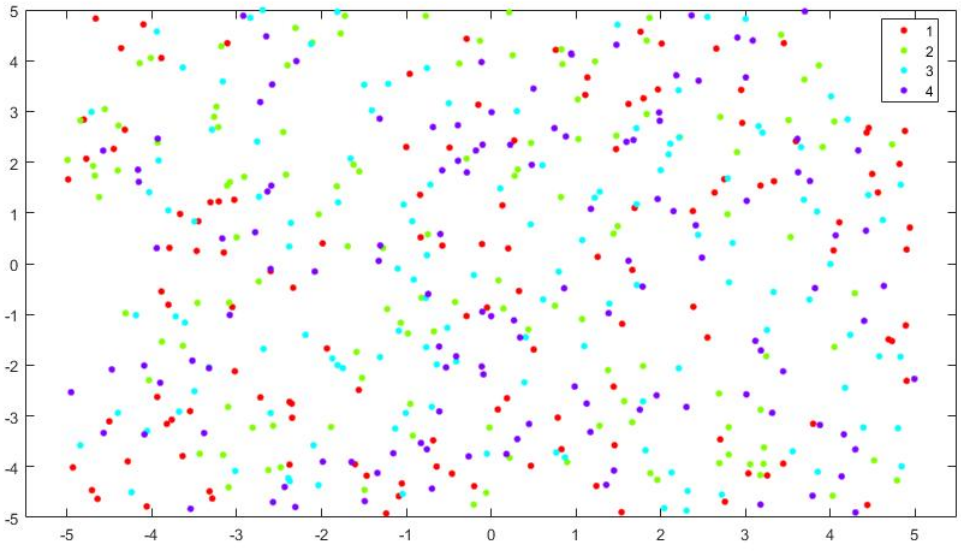
Na slici (2.1) vidimo Voronoiev dijagram 20 slučajno odabranih točki. Linije u dijagramu su granice među Vornoievim ćelijama.

Složenost algoritma je relativno mala. Za pridruživanje n točki nekom od k klastera imamo nk računanja udaljenosti, te nk uspoređivanja da bi pronašli najbližeg predstavnika. Dodamo li utjecaj dimenzije na računanje i uspoređivanje, dobijemo složenost $O(nkd)$. U računanju novih predstavnika imamo nd zbrajanja i kd dijeljenja. Kako je $k \leq n$, složenost ovog koraka iznosi $O(nd)$. Vidimo da je složenost jedne iteracije $O(nkd)$, dakle složenost cijelog algoritma je $O(nkdt)$ gdje je t broj iteracija.

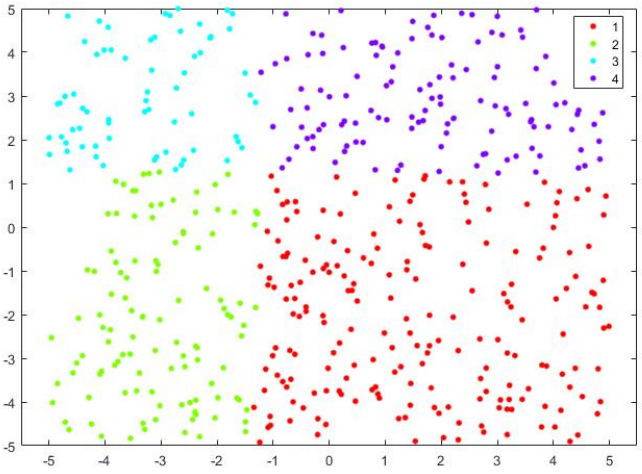
Prednosti algoritma k - sredina su
- jednostavnost i brzina
- lakoća interpretiranja rezultata
- vrlo dobro radi na linearno odvojivim skupovima podataka
- relativno mala složenost algoritma

Mane algoritma k - sredina su
- osjetljivost na incijalizirane centre
- nemogućnost računanja optimalnog parametra k
- osjetljivost na šumove i outliere čija pojava pridonosi pogrešnom klasteriranju
- loše ponašanje ukoliko u skupu podataka već postoje klasteri različite veličine i gustoće
- mogu nastati prazni klasteri

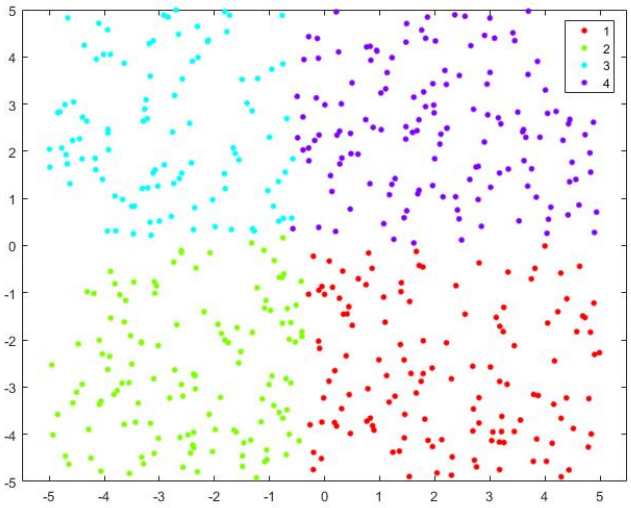
U nastavku ćemo pokazati rezultat algoritma na 500 slučajno odabranih točki iz kvadrata ($[-5,5]$ x $[-5,5]$) u 4 klastera. Početni centri su bili točke $(0,0)$, $(-5,0)$, $(-5,5)$ i $(0,5)$, a algoritam je završio nakon 10 iteracija.



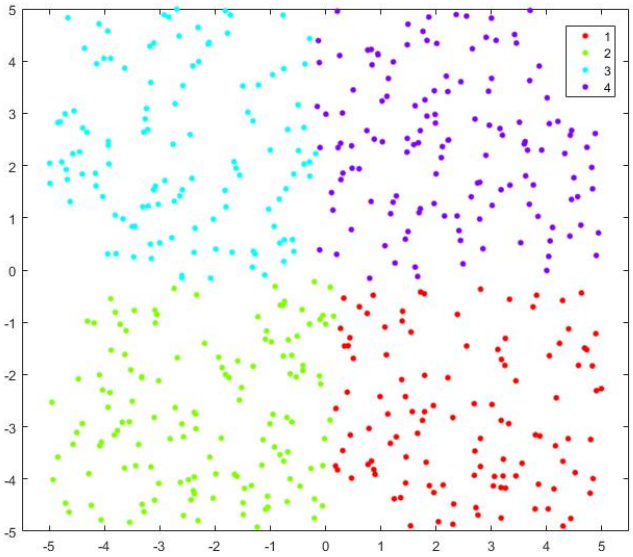
Slika 2.2: Početni razmještaj točki u klastere



Slika 2.3: Razmještaj nakon prve iteracije



Slika 2.4: Razmještaj nakon treće iteracije



Slika 2.5: Razmještaj nakon zadnje iteracije

2.2 Fuzzy k - sredine

Fuzzy k - sredine (tzv. *meko* klasteriranje) je algoritam klasteriranja kod kojeg element skupa pripada svakom klasteru s nekom određenom težinom. Algoritam je razvio Joe Dunn 1973. godine, a Jim Bezdek ju je unaprijedio 1981. godine. Najviše se koristi u prepoznavanju uzoraka [5]. Zasniva se na minimiziranju funkcije cilja:

$$F_m = \sum_{i=1}^{n} \sum_{j=1}^{k} (\mu_{ij})^m \|x_i - c_j\|^2 \ , \tag{2.3}$$

gdje je n kardinalnost skupa podataka, $m \geq 1$ parametar zamućenosti, $\mu_{ij} \in [0,1]$ je težina koja opisuje u kolikoj mjeri x_i pripada j-tom klasteru, a c_j je centar j-tog klastera. Za μ_{ij} još vrijedi

$$\sum_{j=1}^{k} \mu_{ij} = 1 \ . \tag{2.4}$$

U ovom slučaju nismo ograničeni samo na Euklidsku normu već možemo koristiti i ostale. Algoritam konvergira lokalnom minimumu ili sedlastoj točki.

Algoritam 5 Fuzzy k - sredine

•$INPUT$: skup podataka X, broj klastera k, matrica težina U

1. Pomoću U računaj centre klastera koje spremi u vektoru $C = [c_j]$ na način:

$$c_j = \frac{\sum_{i=1}^{n} \mu_{ij}^m \cdot x_i}{\sum_{i=1}^{n} \mu_{ij}^m}, j = 1, 2, ..., k \tag{2.5}$$

2. Ažururaj matricu U na način

$$\mu_{ij} = \frac{1}{\sum_{l=1}^{k} \left(\frac{\|x_i - c_j\|}{\|x_i - c_l\|} \right)^{\frac{2}{m-1}}}, i = 1, 2, ..., n, j = 1, 2, ..., k \tag{2.6}$$

3. Ako je dosegnut maksimalan broj iteracija ili postignut kriterij zaustavljanja, onda STOP. Inače vrati se na 1. korak.

•$OUTPUT$: matrica težina U, centri klastera, vrijednost funkcije cilja

Fuzzy k - sredine imaju primjenu u različitim područjima kao što su bioinformatika, trgovina, analiza slika i sl. Uglavnom se koriste u situacijama gdje jedan objekt skupa želimo povezati sa više klastera, npr. kupce želimo povezati sa različitim brandovima proizvoda. Često se fuzzy k - sredine transforimiraju u čvrsto klasteriranje na način da element skupa dodijelimo samo klasteru za koji taj element ima najveću težinu.

2.3 Sferne k - sredine

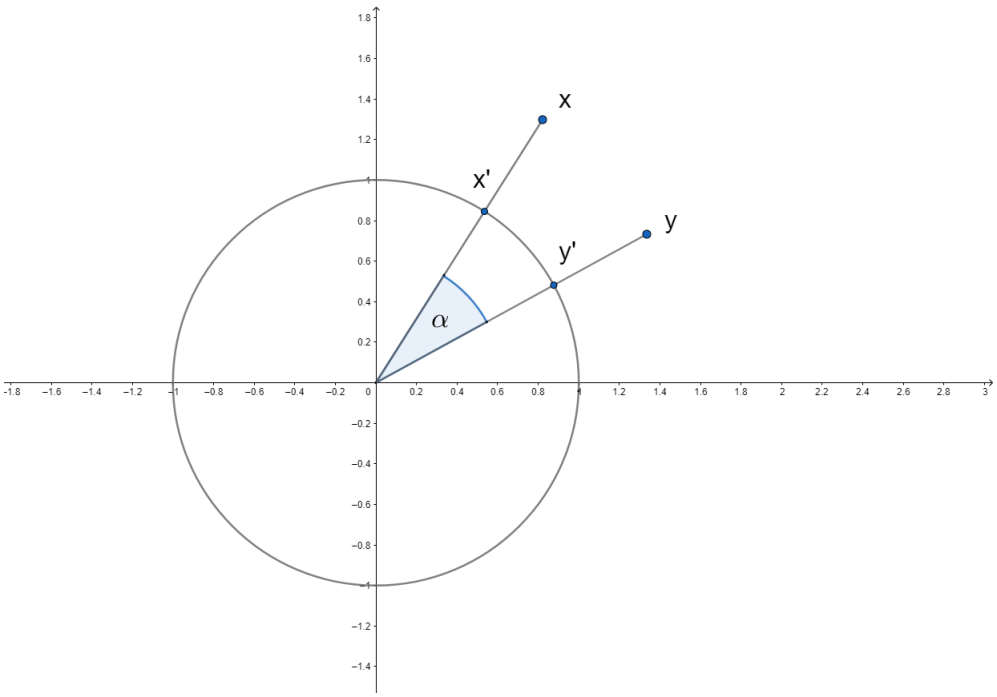
Zamjenom Euklidske udaljenosti, koja se koristi u osnovno algoritmu, sa kosinusovom sličnosti dobijemo sferne k - sredine [7]. Kosinus kuta između dva vektora x i y, $x, y \in \mathbb{R}^n$, se računa na način:

$$d(x, y) = \cos \angle(x, y) = \frac{\langle x, y \rangle}{\|x\|_2 \cdot \|y\|_2} \ , \tag{2.7}$$

gdje je kut $\angle(x, y) \in [0, \frac{\pi}{2}]$, prikazan na slici (2.6).

Vektori x i y se nalaze na jediničnoj kružnici. Stoga je njihova norma jednaka 1, pa formulu (2.7) možemo zapisati na jednostavniji način:

$$d(x, y) = \langle x, y \rangle = \sum_{i=1}^{n} x_i \cdot y_i \ . \tag{2.8}$$



Slika 2.6: Kut među dvije točke

Kako smo promijenili računanje udaljenosti, tako se mijenja i funkcija cilja i računanje novih centroida. Iako radimo sa normaliziranim podacima, centroidi i dalje ne moraju biti norme 1. Zbog toga uvodimo pojam koncept vektor klastera C_i na način:

$$s(C_i) = \sum_{x \in C_i} x \ , \tag{2.9}$$

$$c_i = \frac{s(C_i)}{\|s(C_i)\|} \ . \tag{2.10}$$

Vidimo da je koncept vektor c_i sada normiran. Također uvodimo pojam kvalitete klastera C_i:

$$q(C_i) = \sum_{x \in C_i} x^T \cdot c_i = \|s(C_i)\| \ . \tag{2.11}$$

Za $\emptyset$ pišemo q($\emptyset$) = 0. Kvaliteta klastera nam predstavlja homogenost klastera. Kako radimo sa kosinusovom udaljenosti, vidimo da je homogenost klastera veća, što je i vrijednost kvalitete klastera veća. Dalje kvalitetu klastera koristimo u funkciji cilja koju defini-

ramo kao zbroj kvaliteta svih klastera:

$$F(C_i) = \sum_{i=}^{k} q(C_i) = \sum_{i=}^{k} \sum_{x \in C_i} x^T \cdot c_i \ . \tag{2.12}$$

Tako definirana funkcija monotono raste kroz iteracije, što je obrnuto od osnovnog algoritma *k* - sredina. To možemo pripisati kosinusovoj udaljenosti koja raste, što je kut među točkama manji.

Algoritam 6 Sferne k - sredine

•$INPUT$: skup X, inicijalna inicijalna particija skupa X, broj klastera k, maksimalan broj iteracija

1. Odredi incijalne konceptne vektore $c_i^{(0)}$, $i = 1, 2, ...,$ k koji odgovaraju i - tom klasteru, te postavimo brojač iteracija $t = 0$

2. $\forall\ x \in X$, pomoću kosinusove sličnosti, pronađi najbliži koncept vektor $c_{i^*(x)}$, tj.

$$i^*(x) = \arg\max_j x^T \cdot c_j^t. \tag{2.13}$$

   Nadalje, izračunaj novu particiju $\{C_i^{(t+1)}\}_{i=1}^k$ induciranu starim konceptnim vektorima:

$$C_i^{(t+1)} = \{x \in X : i^*(x) = i\}, 1 \leq l \leq k \tag{2.14}$$

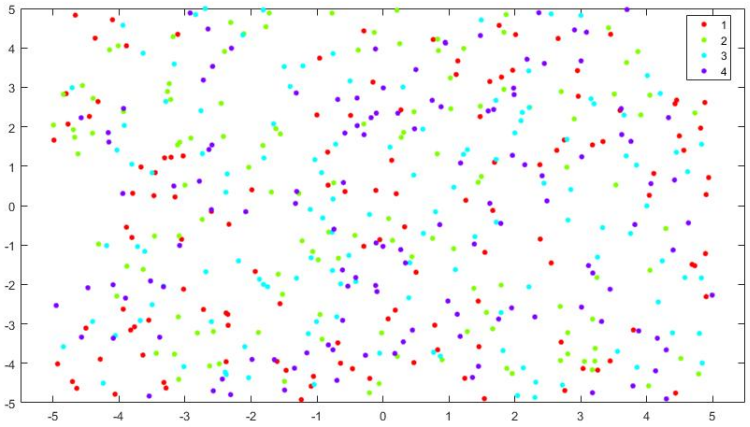
3. Izračunaj nove konceptne vektore prema klasterima određenim u koraku 3.

$$c_i^{(t+1)} = \frac{s(C_i^{(t+1)})}{||s(C_i^{(t+1)})||} \tag{2.15}$$

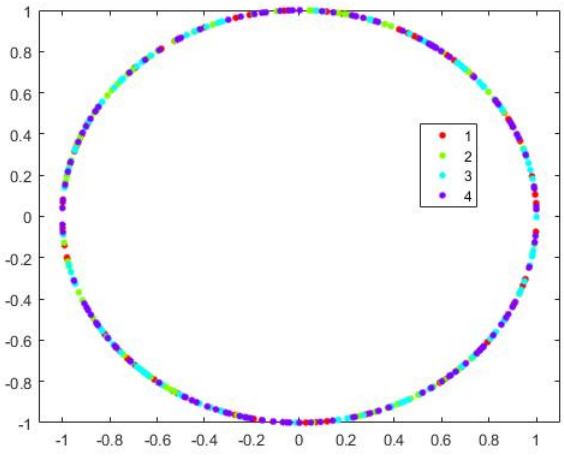
4. Ukoliko je postignut maksimalan broj iteracija ili postignut kriterij zaustavljanja onda STOP. Inače povečaj t za 1 i vrati se na korak 2.

•$OUTPUT$: centri klastera, vektor pripadnosti točke klasteru, vrijednost funkcije cilja

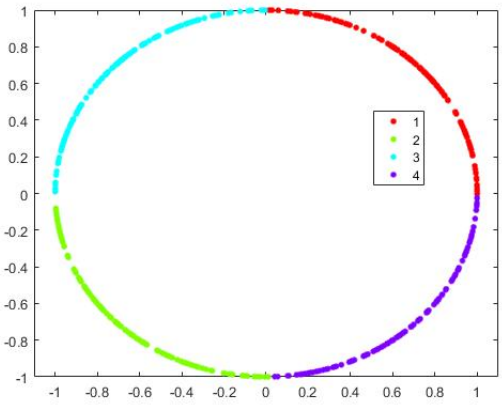
Primjerom ćemo pokazati način kako sferne *k* - sredine klasteriraju 500 slučajnih točki iz kvadrata ([−5, 5] x [−5, 5]) u 4 klastera.



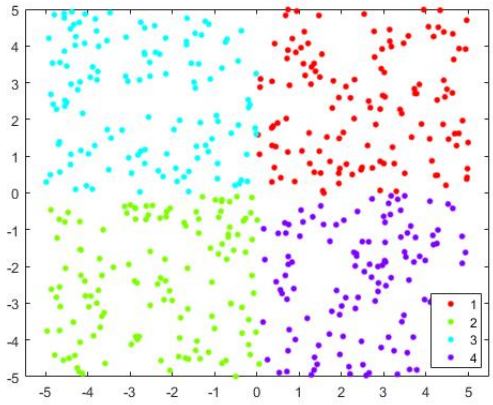
Slika 2.7: Početni razmještaj točki u klastere



Slika 2.8: Projekcija točki na jediničnu kružnicu



Slika 2.9: Klasteriranje projeciranih točki



Slika 2.10: Klasteriranje početnih točki

Prva varijacija

Neka imamo 30 tekstualnih dokumenata, po 10 iz tri različita izvora, i tih 30 dokumenata ima ukupno 1073 riječi. Zbog velike dimenzionalnosti prostora i zbog toga što se većina riječi nalazi u svih 30 dokumenata, kosinusova udaljenost među tih 30 tekstualnih dokumenata je vrlo mala. Također, neka postoji osjetna razlika u kosinusovoj udaljenosti među dokumentima iz istog izvora i kosinusovoj udaljenosti među dokumentima iz različitih izvora. Sferni algoritam k - sredina se za skoro sve različite početne klastere ni ne pomakne,

tj. završi odmah nakon prve iteracije i završni klasteri su isti kao i početni. Pogledamo li dokument x i jedan od početnih klastera C_i, takav da $x \notin C_i$, vdimo da, zbog malog broja dokumenata i male kosinusove udaljenosti među dokumentima, $x^T c_i$ ispadne poprilično mala vrijednost za proizvoljan početni klastering. S druge strane, za x i C_l takvi da vrijedi $x \in C_l$ kosinusova udaljenost $x^T c_l$ se može razdvojiti na zbroj dva izraza :

$$x^T c_l = \frac{1}{\|s(C_l)\|} + \sum_{y \in C_l \setminus x} \frac{x^T y}{\|s(C_l)\|} \quad .$$

gdje prvi izraz dolazi zbog toga što je x norme 1 i njegovog doprinosa u računanju centra c_l. Za skupove podataka koji sadrže malo podataka, a visoko su dimenzionalni, prvi izraz u gornjem zbroju je puno veći od $x^T c_i$, $x \notin C_i$. Zbog toga algoritam sfernih k - sredina ostavlja podatke u početnom klasteru. Taj problem možemo riješiti unaprijeđenjem algoritma sfernih k - sredina metodom prve varijacije [9].

Definicija 3. *Prva varijacija particije* $\{C_i\}_{i=1}^k$ *je particija* $\{C_i'\}_{i=1}^k$ *nastala tako da jedan vektor x izbacimo iz klastera* C_i *i ubacimo u neki drugi klaster* C_j *iste particije.*

Skup svih prvih varijacija particije $\{C_i\}_{i=1}^k$ označavamo s $\mathcal{PV}(\{C_i\}_{i=1}^k)$. Kako svaku točku možemo permjestiti u jedan od $(k-1)$ klastera, navedeni skup ima $n \cdot (k-1)$ elemenata. U svakoj iteraciji algoritma formiramo skup svih prvih varijacija particije, te iz tog skupa uzimamo particiju $\{C_i^*\}_{i=1}^k$ za koju vrijedi:

$$F(\{C_i^*\}_{i=1}^k) \geq F(\{C_i'\}_{i=1}^k) \ , \tag{2.16}$$

gdje je $(\{C_i'\}_{i=1}^k)$ bilo koja prva varijacija particije $\{C_i\}_{i=1}^k$.
Takvu particiju označimo s nextPV$(\{C_i\}_{i=1}^k)$.

Algoritam 7 Prva varijacija

•$INPUT$: skup podataka X, broj klastera k

1. Odredi inicijalnu particiju $\{C_i^{(0)}\}_{i=1}^k$ skupa X, te postavi brojač iteracija $t = 0$

2. Izračunaj novu particiju $\{C_i^{(t+1)}\}_{i=1}^k$:

$$\{C_i^{t+1}\}_{i=1}^k = nextPV(\{C_i\}_{i=1}^k) \tag{2.17}$$

3. Ukoliko je postignut maksimalan broj iteracija ili postignut kriterij zaustavljanja onda STOP. Inače povećaj t za 1 i vrati se na korak 2.

•$OUTPUT$: klastering skupa X, vrijednost funkcije cilja

Većina računanja u algoritmu otpada na računanje razlike kvaliteta klastera:

$$q(C_i^{(t)} \setminus \{x\}) - q(C_i^{(t)}) \quad \text{i} \quad q(C_j^{(t)} \cup \{x\}) - q(C_j^{(t)})$$

gdje je funckija q definirana s (2.11)
Gornje izraze možemo raspisati na način:

$$q(C_i^{(t)} \setminus \{x\}) - q(C_i^{(t)}) = (\|s(C_i^{(t)}))\|^2 - 2\|s(C_i^{(t)})\|x^T c(C_i^{(t)}) + 1)^{1/2} - \|s(C_i^{(t)})\| \tag{2.18}$$

i

$$q(C_j^{(t)} \cup \{x\}) - q(C_j^{(t)})) = (\|s(C_j^{(t)}))\|^2 + 2\|s(C_j^{(t)})\|x^T c(C_j^{(t)}) + 1)^{1/2} - \|s(C_j^{(t)})\| \tag{2.19}$$

Oni nam služe za pronalazak optimalne prve varijacije particije $\{C_i\}_{i=1}^k$ tako da vrijedi (2.16). Također, vrijednosti $\|s(C_l^{(t)}))\|$ i $x^T c(C_l^{(t)})$,$x \in X$, $l = 1, 2, ..., k$, računamo i u svakoj iteraciji sfernih k - sredina. Kada su te vrijednosti dostupne, iteracija prve varijacije ima $O(n + k)$ složenost, gdje je n broj elemenata skupa.

2.4 Jezgrene k - sredine

Algoritam jezgrenih k - sredina je generalizacija osnovnog k means algoritma. Točke se implicitno preslikavaju u prostor više dimenzije, te tako jezgrene k - sredine mogu pronaći klastere koji nisu linearno odvojeni u početnom prostoru. Ovo predstavlja golemu prednost nad osnovnim algoritmom k - sredina i omogućava nam klasteriranje točaka ako nam je dana pozitivno definitna matrica s vrijednostima sličnosti [8] .
Jezgrene k - sredine koriste funkciju ϕ da bi se točke preslikale u prostor više dimenzije. Domena funkcije ϕ je prostor iz kojeg dolaze podaci, a kodomena je prostor više dimenzije u koji preslikavamo podatke.

Kada u tom novom prostoru primjenimo osnovni algoritam k - sredina, linearne granice koje se pojavljuju su zapravo nelinearne u prostoru sa početnom dimenzijom. Cilj algoritma je minimizirati funkciju cilja:

$$F(\{C_i\}_{i=1}^k) = \sum_{i=1}^{k} \sum_{a_i \in C_i} \|\phi(a_i) - c_i\|^2 \ , \tag{2.20}$$

gdje je

$$c_i = \frac{\sum_{a_i \in C_i} \phi(a_i)}{|C_i|}, i = 1, 2, ..., k. \tag{2.21}$$

Obzirom da $\|\phi(a_i) - c_i\|^2 = \langle \phi(a_i) - c_i \ , \ \phi(a_i) - c_i \rangle$, gdje je $\langle \cdot, \cdot \rangle$ skalarni produkt, vrijedi sljedeće:

$$\|\phi(a_i) - c_i\|^2 = \phi(a_i) \cdot \phi(a_i) - \frac{2 \sum_{a_j \in C_i} \phi(a_i) \cdot \phi(a_j)}{|C_i|} + \frac{\sum_{a_j, a_l \in C_i} \phi(a_j) \cdot \phi(a_l)}{|C_i|^2} \ . \tag{2.22}$$

Vidimo da je u računanju Euklidske udaljenosti korišten samo skalarni produkt. Kao rezultat, kad bismo imali *jezgrenu* matricu K, gdje je $K_{ij} = \phi(a_i) \cdot \phi(a_j)$, mogli bismo računati udaljenosti među točkama i centroidima bez da znamo kako zapravo izgledaju $\phi(a_i)$ i $\phi(a_j)$. Može se pokazati da svaka semidefinitna matrica se može smatrati jezgrenom matricom. Najčešće korištene jezgrene funkcije se nalaze u Tablici 2.1; $\kappa(\mathrm{a}_i, a_j) = K_{ij}$.

Tablica 2.1: Popularne jezgrene funkcije

Polonomijalna jezgra	$\kappa(a_i, a_j) = (a_i \cdot a_j + c)^d$
Gaussova jezgra	$\kappa(a_i, a_j) = exp(-\|a_i - a_j\|^2 / 2\alpha^2)$
Sigmondova jezgra	$\kappa(a_i, a_j) = tanh(c(a_i \cdot a_j) + \theta)$

Jezgrene k - sredine sa težinama

Jezgrene k - sredine možemo poopćiti dodavanjem težina uz elemente skupa a_i u funkciju cilja [10] . Označavamo ih s $\omega(a_i)$ ili kraće ω_i i vrijedi

$$\omega_i \geq 0 \ .$$

Funkcija cilja je definirana na način:

$$F(\{C_i\}_{i=1}^k) = \sum_{i=1}^{k} \sum_{a_i \in C_i} \omega_i \cdot \|\phi(a_i) - c_i\|^2 \ . \tag{2.23}$$

Centroidi c_i su najbolji predstavnici klastera te vrijedi:

$$c_i = \frac{\sum_{a_i \in C_i} \omega_i \phi(a_i)}{\sum_{a_i \in C_i} \omega_i} \ . \tag{2.24}$$

Iz istog razloga kao i kod jezgrenih k - sredina bez težina, računanje udaljenosti $\|\phi(a_i) - c_i\|^2$ možemo raspisati na način:

$$\|\phi(a_i) - c_i\|^2 = \phi(a_i) \cdot \phi(a_i) - \frac{2 \sum_{a_j \in C_i} \omega_j \phi(a_i) \cdot \phi(a_j)}{\sum_{a_j \in C_i} \omega_j} + \frac{\sum_{a_j, a_l \in C_i} \omega_j \omega_l \phi(a_j) \cdot \phi(a_l)}{(\sum_{a_j \in C_i} \omega_j)^2} \ . \tag{2.25}$$

Koristeći jezgrenu matricu K i činjenicu da vrijedi $K_{ij} = \phi(a_i) \cdot \phi(a_j)$, raspis (2.25) možemo prepisati na način:

$$\|\phi(a_i) - c_i\|^2 = K_{ii} - \frac{2\sum_{a_j \in C_i} \omega_j K_{ij}}{\sum_{a_j \in C_i} \omega_j} + \frac{\sum_{a_j, a_l \in C_i} \omega, \omega_l K_{jl}}{(\sum_{a_j \in C_i} \omega_j)^2}. \tag{2.26}$$

Algoritam 8 Jezgrene k - sredine s težinama

•$INPUT$: jezgrena matrica K, broj klastera k, vektor težina svake točke ω, skup podataka X

1. Odredi inicijalnu particiju $\{C_i^{(0)}\}_{i=1}^k$ skupa X, te postavi brojač iteracija $t = 0$
2. Za svaku točku $a \in X$ pronađi novi index klastera:

$$j^*(a) = \arg\min_j \|\phi(a) - c_i\|^2 \tag{2.27}$$

   koristeći raspis (2.25)

3. Izračunaj nove klastere na način:

$$C_j^{(t+1)} = \{a : j^*(a) = j\} \tag{2.28}$$

4. Ukoliko je postignut maksimalan broj iteracija ili postignut kriterij zaustavljanja onda STOP. Inače povečaj t za 1 i vrati se na korak 2.

•$OUTPUT$: vektor članstva u kojem je zapisano za svaki $x \in X$ kojem klasteru pripada

2.5 Harmonijske k - sredine

Harmonijske k - sredine je algoritam za klasteriranje koji je Bin Zhang predstavio 1999. godine, a 2002. su ga modificirali Greg Hammerly i Charles Elkan. To je algoritam koji se zasniva na računanju centara, te u funkciji cilja koristi *harmonijsku sredinu* udaljenosti između točaka skupa i izračunatih centara [13].

Definicija 4. *Harmonijska sredina skupa* $X = \{x_1, x_2, ..., x_n\}$ *definirana je s:*

$$HA(X) = \frac{n}{\sum_{k=1}^n \frac{1}{x_k}} \tag{2.29}$$

Formula (2.29) ima svojstvo da ako je neki od elemenata skupa X mali, tada je i vrijednost sredine mala. Ukoliko su svi elementi veliki, tada je i vrijednost sredine velika. Harmonijska sredina se ponaša slično kao i funkcija minimum, ali ipak daje određene težine drugim vrijednostima. Funkcija cilja za osnovni algoritam k - sredina je definirana na način:

$$F(X) = \sum_{i=1}^{n} HA(\{\|x_i - c_l\|^2 : l = 1, 2, ..., k\}) = \sum_{i=1}^{n} \frac{k}{\sum_{l=1}^{k} \frac{1}{\|x_i - c_l\|^2}} \tag{2.30}$$

gdje je c_l centar klastera C_l, l = 1, 2, ..., k.

Za algoritam harmonijskih k - sredina ćemo promijeniti potenciju norme razlike točke i centra, te umijesto 2 imamo parametar p :

$$F(X) = \sum_{i=1}^{n} \frac{k}{\sum_{l=1}^{k} \frac{1}{\|x_i - c_l\|^p}}. \tag{2.31}$$

Pokazalo se da formula (2.32) radi bolje za $p \geq 2$.

Kod osnovnog algoritma k - sredina može doći do problema u područjima visoke gustoće točki i centara. Može dogoditi da se jedan centar ne može pomaknuti od točki s kojima je spojen, iako je drugi centar u blizini. Taj drugi centar može doprinijeti lokalno lošijem rješenju, ali globalni efekt jednog od ta dva centra može pridonijeti boljem klasteriranju. Kako bi nadišao ovaj problem, algoritam harmonijskih k - sredina koristi težine pomoću kojih se određuje članstvo neke točke određenom klasteru. Kako funkcija cilja, za svaku točku, računa udaljenost do svakog centra, vrlo je osjetljiva kada postoje dva ili više centara koji su blizu istoj točki. Algoritam prirodno premješta jedan ili više takvih centara u prostore gdje postoji točka koja nema niti jedan centar u svojoj blizini. To će smanjiti vrijednost funkcije cilja. Također, algoritam dodjeljuje različite težine svakoj točki, ovisno o harmonijskoj sredini. Harmonijska sredina će pridružiti veliku težinu točki koja u svojoj blizini nema niti jedan centar, a malu težinu točki sa jednim ili više centara u svojoj blizini [13]. Funkcija za ažuriranje težina je definirana na način:

$$\omega(x_i) = \frac{\sum_{j=1}^{k} \|x_i - c_j\|^{-p-2}}{\left(\sum_{j=1}^{k} \|x_i - c_j\|^{-p}\right)^2}. \tag{2.32}$$

Na temelju uspordbe općenitog algoritma harmonijskih k - sredina i algoritma k - sredina, Zhang je otkrio da za $3 \leq p \leq 3.5$ harmonijske k - sredine nadmašuju k - sredine i konvergencija mnogo manje ovisi o izboru početnih vrijednosti [4].

Poglavlje 3

Primjeri

3.1 Obrada slike

Algoritam k - sredina, već u svom osnovnom obliku, može dati jako dobre rezultate. Iz tog razloga se često koristi u obradi slika za kvantizaciju boja. Nakon što se definira željeni broj boja na slici k, slika se obradi algoritmom k - sredina. Svaka boja će biti pridružena jednom od k klastera, te u prikazu te slike, umjesto točne boje koja se nalazi na slici na određenom pikselu, koristimo centroid klastera u kojem se nalazi ta boja. U nastavku vidimo par primjera slika klasteriranih za različiti k. Primjeri su napravljeni u programskom okruženju MATLAB, koji sliku dimenzija ($n1$ x $n2$) učita kao tenzor dimenzija ($n1$ x $n2$ x $n3$). Dakle za svaki piksel slike dobijemo trodimenzionalni vektor. Navedena uređena trojka predstavlja točno jednu boju u RGB modelu. RGB model je model boja u kojem se određenom nijansom crvene, zelene i plave boje dobiju sve ostale boje, te svaka od njih je određena jedinstvenom uređenom trojkom brojeva. Nadalje, navedeni tenzor sam pretvorio u matricu dimenzije (($n1$ x $n2$) x 3) na kojoj sam primijenio osnovni algoritam k - sredina, dakle za računanje udaljenosti se koristila Euklidska udaljenost. Za svaku boju na slici nam algoritam vrati kojem klasteru pripada, te koji je centar tog klastera, pa lako možemo reproducirati sliku samo sa k boja. Za računanje početnih centara koristio se algoritam *k-means++* [2] čiji kod se nalazi u sljedećem poglavlju.

1. Slika u sivim tonovima

Slika 3.1: Originalna slika kamermana

Slika 3.2: Slika kamermana, $k = 2$

Slika 3.3: Slika kamermana, $k = 5$

2. Slika s jasno definiranim granicama

Slika 3.4: Originalna slika skakača na snijegu

Slika 3.5: Slika skakača na snijegu, $k = 4$

Slika 3.6: Slika skakača na snijegu, $k = 8$

Slika 3.7: Slika skakača na snijegu, $k = 16$

3. Slika prirode s puno detalja

Slika 3.8: Originalna slika prirode

Slika 3.9: Slika prirode, $k = 4$

Slika 3.10: Slika prirode, $k = 8$

Slika 3.11: Slika prirode, $k = 16$

4. Slika osobe (Lenna)

Slika 3.12: Originalna slika osobe

Slika 3.13: Slika osobe, $k = 4$

Slika 3.14: Slika osobe, $k = 8$

Slika 3.15: Slika osobe, $k = 16$

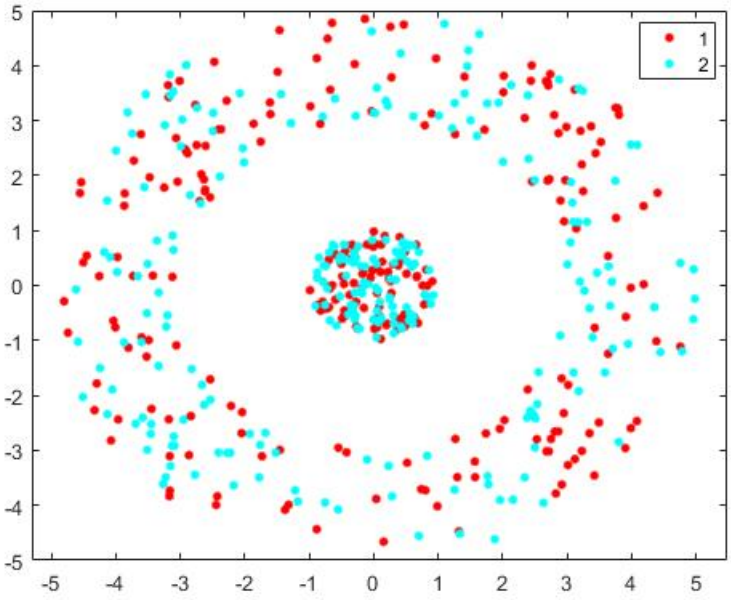
Na primjerima se jasno vidi kako smanjivanjem broja željenih boja na slici, dobijemo fokus na najvažnijim elementima slike. Na taj način možemo segmentirati sliku, te izvući nama najbitnije elemente slike koji bi nam mogli koristiti u daljnjoj obradi.

3.2 Usporedba osnovog i jezgrenog algoritma k - sredina

Kako smo već naveli, osnovi algoritam radi dobro kada su klasteri linearno odvojivi. Tada lako prepozna hiperravnine koje ih dijele, te napravi dobar razmještaj elemenata skupa po klasterima. U nastavku ćemo dati primjer gdje točke nisu linearno odvojive. U tom slučaju osnovni algoritam nam ne daje dobro rješenje i potrebno je pronaći novi način za klasteriranje. Primjer je napravljem u programskom okruženju MATLAB.

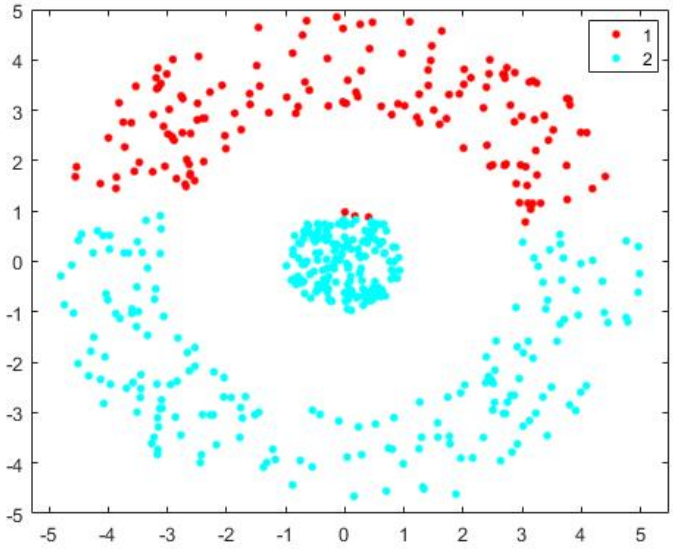
Točke na slici (3.16) ćemo klasterirati u 2 klastera. Na slici (3.17) vidimo rezultat osnovnog algoritma. Algoritam je pronašao hiperravninu koja ne sadrži niti jednu točku skupa. Ovisno o početnim centrima, algoritam daje različita rješenja i ima različito trajanje. U našem slučaju, početni centri su birani slučajnim putem.

U nastavku dajemo rješenje dobiveno algoritmom jezgrenih k - sredina. Kao jezgrenu funkciju smo odabrali Gaussovu jezgrenu funkciju definiranu u Tablici (2.1). Inicijalni klastering je slučajno odabran, a algoritmu je trebalo 4 iteracije do rješenja. Krajnji rezultat je isti kao i rezultat treće iteracije koji se nalazi u nastavku, na grafu (3.20). Ukoliko ne želimo klastere zasnovane na centrima, koje nam daje osnovni algoritam k - sredina, već

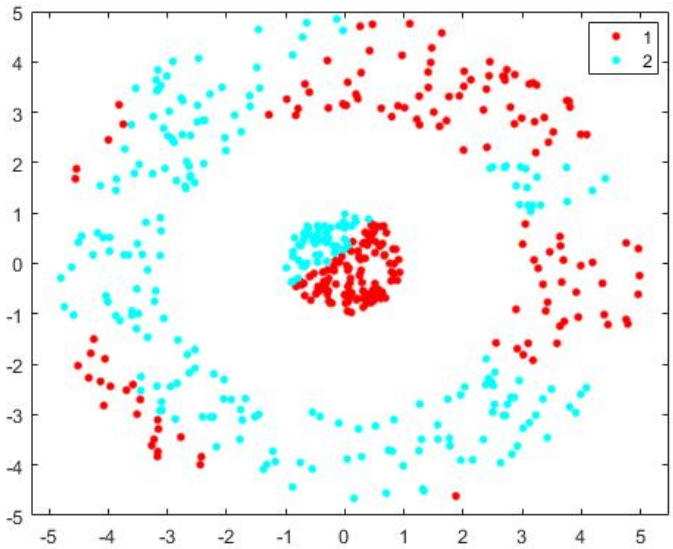


Slika 3.16: Početni raspored točki

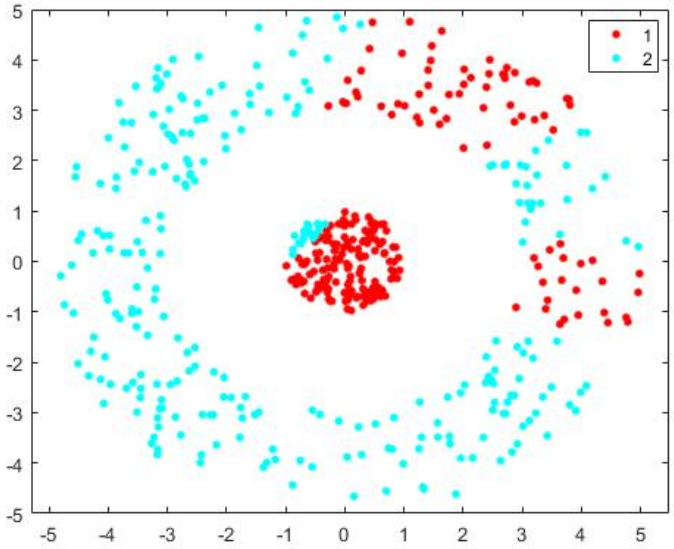
klastere zasnovane na gustoći, vidimo da nam algoritam jezgrenih k - sredina daje puno bolje rješenje.



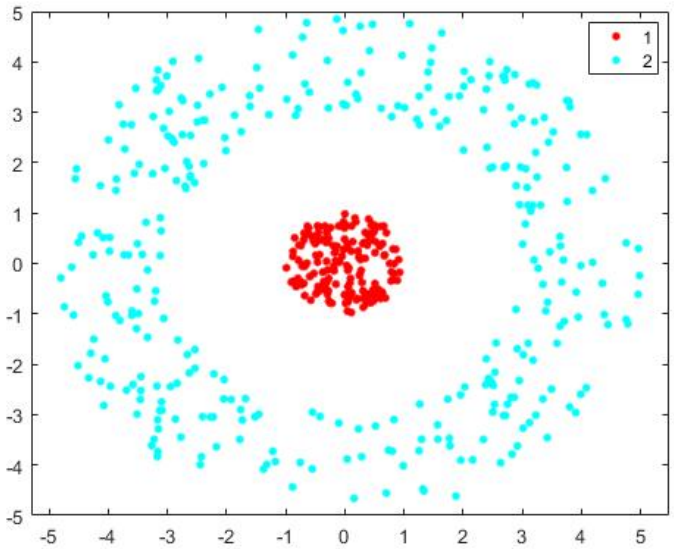
Slika 3.17: Klasteriranje osnovnim algoritmom



Slika 3.18: Jezgreni algoritam, 1. iteracija



Slika 3.19: Jezgreni algoritam, 2. iteracija



Slika 3.20: Jezgreni algoritam, 3. iteracija

Implementacija u MATLAB-u

U nastavku su prikazani prikazani funkcije osnovnih *k* - sredina, jezgrenih *k* - sredina i metode za inicijalizaciju centara *kmeans* + + napisanih u programskom okruženju MATLAB.

1. Osnovni algoritam *k* - sredina:

```
function [ centers, F, X, V, iter ] = Lloyd_k_means( k
    , X, tol, max_iter, init_centers )
%Lloydov k-means algoritam
%   ulazni podaci:
%       - k: zeljeni broj klastera
%       - X: matrica tocaka dimenzije n_x_d gdje je
    broj tocaka, a d
%            dimenzija prostora
%       - tol: tolerancija
%       - max_iter: maksimalan broj iteracija
%       - init_centers: pocetni razmjestaj centara
%
%   izlazni podaci:
%       - centers: zavrsni razmjestaj centara
%       - F: vrijednost funckije cilja
%       - X: ulazna matrica podataka
%       - V: vektor razmjestaja tocki po klasterima
%       - iter: broj izvedenih iteracija


%%inicijaliziranje varijabli
[n,d] = size(X);
flag = 1;
centers = zeros(k,d);
```

```
23 F = zeros(1);
24
25 V = zeros(n,1);
26 iter = 0;
27
28 %% pridruzivanje tocki najblizem klasteru i racunanje
       vrijednosti funkcije cilja
29 for i = 1:n
30     temp = zeros(k,1) ;
31     for j = 1:k
32         temp(j) = norm(X(i,:) - init_centers(j,:));
33         if temp(j) == min(temp(1:j))
34             V(i) = j;
35         end
36     end
37     F(1) = F(1) + norm(X(i,:) - init_centers(V(i),:));
38 end
39
40 %% iteracije
41
42 while(flag == 1)
43
44   iter = iter + 1;
45   F(iter + 1) = 0;
46   sume = zeros(k,d);
47   brojac = zeros(k,1);
48   for i = 1:n
49       sume(V(i),:) = sume(V(i),:) + X(i,:);
50       brojac(V(i)) = brojac(V(i)) +1;
51   end
52
53   %% racunanje novih centara
54   for i = 1:k
55     if (brojac(i) ~= 0)
56         centers(i,:) = sume(i,:) / brojac(i);
57     end
58   end
59
60
```

```
61    %% dodijeljivanje tocki najblizem centru i racunanje
          vrijednosti funkcije cilja
62     for i = 1:n
63       temp = zeros(k,1);
64       for j = 1:k
65           temp(j) = norm(X(i,:) - centers(j,:));
66           if temp(j) == min(temp(1:j))
67               V(i) = j;
68           end
69       end
70       F(iter+1) = F(iter+1) + norm(X(i,:) - init_centers(
             V(i),:));
71
72     end
73
74     %% kriterij zaustavljanja
75
76     if( abs(F(iter + 1) - F(iter)) < tol || iter ==
          max_iter)
77       flag = 0;
78     end
79
80
81
82  end
83
84  end
```

2. Algoritam za inicijaliziranje centara *k-means++*

```
1  function [ centers ] = kmeans_pp( X, k )
2  %racunanje inicijalnih centara za k-means algoritam
3  %INPUT:
4  %    - X: skup podataka (nxd matrica; n - broj podataka,
       d - dimenzija)
5  %    - k: broj klastera
```

```
%OUTPUT:
%    - centers: centri klastera

[n,d]= size(X);
centers = X(round((n-1)*rand)+1  ,:);
memb = ones(n,1);
D = zeros(n,1);
for i = 2:k
     for l = 1:n
        D(l) = norm(X(l,:) - centers(memb(l),:));
     end
     D = D.^2;
     D = D/sum(D);
     cumprobs = cumsum(D);
     temp = rand;
     for j = 1:n
        if (temp < cumprobs(j) )
            centers(i,:) = X(j,:);
            break
        end

     end
end
end
```

3. Gaussova jezgrena funkcija

```
function K = Gauss_jezg_f(X, Y, s)
% Gaussova jezgrena funkcija
% Input:
%    X: n1 x d matrica podataka
%    Y: n2 x d matrica podataka
%    s: sigma
% Ouput:
%    K: n1 x n2 jezgrena matrica

```

```
%% inicijalizacija varijabli
if nargin < 3
    s = 1;
end
[n1,d] = size(X);
[n2,d] = size(Y);
K = [];

%% punjenje jezgrene tablice
for i=1:n1
    for j=1:n2
        temp = X(i,:) - Y(j,:);
        temp = norm(temp)^2;
        K(i,j) = exp(temp/(-2*s^2));
    end
end
end
```

4. Jezgrene *k* - sredine

```
function [V, F, iter] = kernel_kmeans(X, k, max_iter,
    jezg_f)
% Jezgrene k - sredine
% Input:
%   X: n x d skup podataka
%   k: broj klastera
%   max_iter: maksimalan broj iteracija
%   jezg_f: jezgrena funkcija
%
% Output:
%   V: n x 1 vektor clanstva
%   F: vrijednost funkcije cilja
%   iter: broj izvedenih iteracija


n = size(X,1);
```

```
16 last_V = zeros(n,1);
17 iter = 0;
18
19 %% inicijalizacija pocetne particije
20 V = ceil(k*rand(1,n))';
21
22 %% odabir jezgrene funkcije
23 if nargin < 4
24     jezg_f = @Gauss_jezg_f;
25 end
26 K = jezg_f(X,X);
27
28 %% iteriranje
29
30 while (any(V ~= last_V) || iter > max_iter)
31     iter = iter+1;
32     [~,~,last_V(:)] = unique(V);
33     E = sparse(1:n,last_V,1);
34     Temp = full(sum(E,1));
35     for i = 1:k
36         E(:,i) = E(:,i)./Temp(i);
37     end
38     T = K*E;
39     s= zeros(500,k);
40     for i = 1:500
41         s(i,:) = T(i,:)-dot(T,E,1)/2;
42     end
43     [val, V] = max(s,[],2);
44 end
45 F = trace(K)-2*sum(val);
46 end
```

Bibliografija

[1] M. Hubert A. Struyf i P. J. Rousseeuw, *Clustering in an Object-Oriented Environment*, Journal of Statistical Software (1987).

[2] D. Arthur i S. Vassilvitskii, *K-means++: The Advantages of Careful Seeding*, SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (2007).

[3] V. Batagelj, *Generalized Ward and Related Clustering Problems*, Classification and Related Methods of Data Analysis (1988).

[4] H. Beigi, *Fundamentals of Speaker Recognition*, Springer, 2011.

[5] D. J. Bora i A. K. Gupta, *A Comparative study Between Fuzzy Clustering Algorithm and Hard Clustering Algorithm*, International Journal of Computer Trends and Technology (IJCTT) **10** (2014), br. 2.

[6] L. Bottou i Y. Bengio, *Convergence Properties of the K-Means Algorithms*, (1995).

[7] I. Dhillon i D. Modha, *Concept decomposition for large sparse text data using clustering*, (2001).

[8] Y. Guan I. Dhillon i Kulis B., *A unied view of kernel k-means,spectral clustering and graph cuts*, (2004).

[9] Y. Guan I. Dhillon i J.Kogan, *Iterative Clustering of High Dimensional Text Data Augmented by Local Search*, (2012).

[10] Y. Guan I. Dhillon i B. Kulis, *Kernel k-means,spectral clustering and normalized cuts*, (2004).

[11] W. C. Black J. F. Hair i B. J. Babin, *Multivariate Data Analysis: A Global Perspective*, Pearson education, 2010.

[12] M. Steinbach P. N. Tan i V. Kumar, *Introduction to Data Mining*, Addison-Wesley Longman Publishing Co., 2005.

[13] Alper Ünler Zülal Güngör, *K-Harmonic means data clustering with tabu-search method*, (2007).

Sažetak

U ovom radu prikazan je algoritam klasteriranja k - sredinama. Započinje davanjem osnovnih pojmova i opisom podjele algoritama klasteriranja na hijerarhije i particijske. U nastavku se opisuju osnovni algoritam k - sredina i njegove varijacije: fuzzy k - sredine, sferne k- sredine, jezgrene k - sredine i harmonijske k - sredine. Također, za sferne k - sredine je prikazano unaprijeđenje metodom prve varijacije, dok za jezgrene k - sredine je prikazana nadogradnja uvođenjem težina. Na kraju rada je prikazana kvantizacija boja na slikama pomoću osnovnog algoritma k - sredina, te uspordba jezgrenog i osnovnog algoritma na primjeru. U dodatku se nalaze kodovi napisani u programskom okruženju MATLAB pomoću kojih su se dobili primjeri u trećem poglavlju.

Summary

This paper presents k - means clustering algorithm. It starts with basic concepts and it gives description of division of clustering algorithms into hierarchical and partitioning. Further, basic k-means algorithm and few of its variations such as Fuzzy k-means, Spherical k-means, Kernel k-means and Harmonic k-means, are described. Also, for Spherical k-means it's described improvement with First Variation method, and for Kernel k-means it is described upgrade by bringing weights into algorithm. At the end of this paper it is shown how we can do quantization of color in the images with basic k-means algorithm environment. Also, comparison between basic and Kernel k-means algorithms is given using an example. The Appendix contains MATLAB codes which are used to make examples in 3. chapter.

Životopis

Rođen sam dana 19.03.1993. u Splitu, ali sam odrastao u Šibeniku gdje sam završio Gimnaziju Antuna Vrančića, jezični smjer. Potom sam, 2011. godine, upisao Preddiplomski studij matematike na Prirodoslovno - Matematičkom fakultetu u Zagrebu, koji sam završio 2015. godine. Iste godine upisujem Diplomski studij Matematička statistika, također na Prirodoslovno - Matematičkom fakultetu u Zagrebu. Dobro se snalazim u programskim jezicima R i MATLAB. Također, dobro poznajem rad s bazama podataka i pisanje upita u PLSQL jeziku.