Ubrzanje k-sredina pomoću nejednakosti trokuta

Dorotea Rajšel, Iva Sokač

4. prosinca 2017.

Sažetak

k-means je metoda u data mining-u za grupiranje podataka, vrlo popularna u područjima gdje se barata s velikim količinama višedimenzionalnih podataka potrebnih za daljne analize. Lloydov algoritam (k-sredine) je najčešće korišteni algoritam za pronalaženje klastera u podacima. Postoje mnoge varijacije i nadogradnje ovog algoritma. Neke od njih koriste nejednakost trokuta da bi ga ubrzali. Primjer takvih su Elkanov i Hamerlyjev algoritam. Cilj našeg rada je objasniti ideje ova tri algoritma te usporediti njihovu efikasnost na različitim skupovima podataka.

Sadržaj

1 Uvod

Grupiranje podataka (klasteriranje) je tehnika kojom se stvaraju grupe takve da su elementi u istoj grupi (klasteru) "sličniji" nego elementi iz različitih grupa. Često je korištena u mnogim područjima kao što su tržišna istraživanja, statistička analiza, kompresija slika, strojno učenje, bioinformatika, društvene mreže itd.

Standardan i najpopularniji algoritam za ovaj problem je tzv. Lloydov algoritam (k-means). On počinje tako da za unaprijed zadan broj klastera k odabere k centara (najčešće na slučajan način) iz zadanog skupa X. Zatim ponavlja iduća dva koraka:
1) pridružuje točke klasteru s najbližim centrom
2) ažurira centre, tj. određuje centre za novi raspored točaka u klasterima
Algoritam staje kada prestane dolaziti do promjena u koraku (1) ili nakon unaprijed određenog broja koraka.

Klasteriranje spada pod NP-teške probleme, no ima mnogo heurističkih algoritama koji konvergiraju brzo prema lokalnom rješenju. Standardni algoritam za ovaj problem je Lloydov algoritam. Jedan od nedostataka ovog algoritma je to što je većina izračunatih udaljenosti objekata suvišna. Nije uvijek potrebno izračunati egzaktnu udaljenost točke i centra da bismo znali reći da li bi trebala ta točka biti dodijeljena tom centru ili ne. Jedan od načina za smanjenje broja nepotrebnih računanja je korištenje nejednakosti trokuta. Time ćemo u Elkanovom algoritmu broj računa udaljenosti reducirati s *nke* na približno n izračunatih udaljenosti, pri čemu je n broj podataka, k broj traženih klastera i e broj zahtjevanih iteracija.

Prirodno je zahtijevati da ubrzani algoritam k-sredina ima sljedeća svojstva:

- može početi od bilo koje inicijalizacije centara tako da se i ovaj dio procesa može optimizirati
- za dane jednake inicijalne centre producira jednake završne centre kao i standardni algoritam
- može koristiti bilo koju "black-box" metriku za udaljenost, tj. metriku čiju implementaciju, odnosno definiciju ne znamo

2 Korištenje nejednakosti trokuta u ubrzanju

Ovaj pristup ubrzanju k-sredina bazira se na nejednakosti trokuta: za svake tri točke x, y i z vrijedi $d(x,z) \leq d(x,y) + d(y,z)$. Ovo svojstvo posjeduju sve metrike.

Sljedeće leme nam pokazuju kako možemo iskoristiti nejednakost trokuta da bismo izbjegli suvišna računanja udaljenosti.

Lema 2.1. *Neka je x točka i b i c centri. Ako je $d(b,c) \geq 2d(x,b)$, onda je $d(x,c) \geq d(x,b)$.*

Dokaz. Iz nejednakosti trokuta za x, b i c znamo: $d(b,c) \leq d(b,x) + d(x,c)$.Dakle, $d(b,c) - d(x,b) \leq d(x,c)$. Iz pretpostavke slijedi da je $d(x,c) \geq d(b,c) - d(x,b) \geq 2d(x,b) - d(x,b) = d(x,b)$. □

Lema 2.2. *Neka je x točka i b i c centri. Tada je $d(x,c) \geq max\{0, d(x,b) - d(b,c)\}$.*

Dokaz. Znamo $d(x,c) \geq 0$. Iz nejednakosti $d(x,b) \leq d(x,c) + d(b,c)$ slijedi $d(x,c) \geq d(x,b) - d(b,c)$. □

Neka je x bilo koja točka našeg skupa, c centar kojemu je x trenutačno dodijeljen i neka je c' bilo koji drugi centar. Prva nam lema kaže da ukoliko je $\frac{1}{2}d(c,c') \geq d(x,c)$, onda je $d(x,c') \geq d(x,c)$ pa u ovom slučaju nije potrebno računati $d(x,c')$.

Pretpostavimo da ne znamo egzaktno $d(x,c)$, ali imamo gornju ogradu u za udaljenost točke od svog trenutačnog centra, tj. $u \geq d(x,c)$. Tada moramo izračunati $d(x,c')$ samo ako je $u > \frac{1}{2}d(c,c')$. Inače je $d(x,c) \leq u \leq \frac{1}{2}d(c,c') \leq d(x,c')$ pa nije potrebno računati udaljenosti. Štoviše, ako je $u < \frac{1}{2}min_{c' \neq c}d(c,c')$, znamo da se centar c pridijeljen točki x ne mijenja pa time izbjegnemo izračun udaljenosti vezanih za x.

Drugu lemu interpretiramo u idućem obliku. Neka je x bilo koja točka našeg skupa, neka je b bilo koji centar, te neka je s b' označena prijašnja verzija centra b (tj. b' je centar b, ali u prijašnjoj iteraciji). Pretpostavimo da smo u prijašnjoj iteraciji znali donju granicu l' takvu da $d(x,b') \geq l'$. Tada za donju granicu l u trenutnoj iteraciji možemo zaključiti sljedeće:

$$d(x,b) \geq max\{0, d(x,b') - d(b,b')\} \geq max\{0, l' - d(b,b')\} = l$$

Neformalno, ako je l' dobra aproksimacija udaljenosti između točke x i centra u prijašnjoj iteraciji i centar se nije puno pomaknuo, onda je l dobra aproksimacija udaljenosti između točke x i centra u trenutnoj iteraciji.

2.1 Elkanov algoritam

Algoritam koji Elkan predstavlja u svom članku [Elk03] je prva varijanta k-means metoda koja koristi donje ograde i prenosi informacije iz jedne iteracije u drugu.

Primijenimo li sada sve ove opservacije, vidimo da je Elkanov ubrzani k-mean algoritam dan sljedećim:

Prvo odaberemo početne centre, te za svaku točku x i svaki centar c postavimo donje ograde $l(x,c) = 0$. Zatim svaki x pridružimo njemu najbližem početnom centru, tj. centar od x je $c(x) = argmin_c d(x,c)$ (pri tom koristimo prvu lemu kako bismo izbjegli suvišne izračune). Svaki put kada izračunamo $d(x,c)$, postavimo $l(x,c) = d(x,c)$. Stavimo, $u(x) = min_c d(x,c)$.
Sada ponavljamo iduće korake dok ne dobijemo konvergenciju:

1. Za sve centre c i c' izračunaj $d(c,c')$. Za sve centre c izračunaj $s(c) = \frac{1}{2}min_{\forall c' \neq c}d(c,c')$.

2. Poistovjeti sve točke x za koje je $u(x) \leq s(c(x))$.

3. Za sve preostale točke x i centre c za koje je:

   - $c \neq c(x)$
   - $u(x) > l(x,c)$
   - $u(x) > \frac{1}{2}d(c(x),c)$

   (a) Ako je $r(x)$, onda izračunaj $d(x,c(x))$ i dodijeli $r(x) = false$.
   Inače, $d(x,c(x)) = u(x)$.

   (b) Ako je $d(x,c(x)) > l(x,c)$ ili je $d(x,c(x)) > \frac{1}{2}d(c(x),c)$, onda izračunaj $d(x,c)$
   Ako je $d(x,c) < d(x,c(x))$, onda $c(x) = c$.

4. Za svaki centar c, neka je $m(c)$ srednja vrijednost točaka pridruženih centru c.

5. Za svaku točku c i centar c, stavimo

$$l(x, c) = max\{l(x, c) - d(c, m(c)), 0\}$$

6. Za svaku točku x, stavimo

$$u(x) = u(x) + d(m(c(x)), c(x))$$

$$r(x) = true$$

7. Zamijenimo svaki centar c s $m(c)$.

Eksperimentalno je pokazano da je ovaj algoritam učinkovit za skupove dimenzije najviše 1000 te da mu učinkovitost raste s povećanjem broja klastera k. Za $k \geq 20$ Elkanov algoritam je puno brži od Lloydovog algoritma.

Ovaj algoritam je efikasan u smanjenju broja izračunatih udaljenosti jer su gornje ograde $u(x)$ i donje ograde $l(x, c)$ "tijesne" za većinu točaka x i centra c. Svakom iteracijom lokacija većine centara se promijeni jako malo pa su ograde "tijesne" i na početku sljedeće iteracije.

Peti korak ima smisla kada je metrika koju koristimo zapravo Euklidska. Inače moramo $m(c)$ definirati na neki drugi način tako da bude "predstavnik" klastera c.

2.2 Hamerlyjev algoritam

Ovaj algoritam predstavlja novu metodu ubrzanja Lloydovog algoritma koji nadograđuje Elkanov. Kao takav, iskorištava efikasno ažurirane ograde na udaljenosti i nejednakost trokuta u svrhu izbjegavanja suvišnih računanja udaljenosti.

Najdublja petlja u Lloydovom algoritmu, koja ide preko centroida, dio je gdje algoritam provede većinu vremena. Algoritam koji je predstavio Greg Hamerly u svom članku [Ham10] preskače ovu petlju u 80% slučajeva. To je omogućilo korištenje samo n donjih ograda umjesto nk, na udaljenost točke i drugog najbližeg centra te dodatno provjeravanje uvjeta prije ulaska u najdublju petlju. Hamerlyjev algoritam je najefikasniji za male i srednje dimenzije podataka, dok je Elkanov najefikasniji za velike dimenzije.

Da bismo predstavili algoritam, uvodimo definicije korištenih struktura. Strukture vezane uz centre:

- c_j - centar klustera $j, (1 \leq j \leq k)$,
- c'_j - (vektorska) suma svih točaka koje pripadaju klusteru j,
- q_j - broj točaka koje pripadaju klusteru j ,
- p_j - udaljenost za koju se pomaknuo c_j u zadnjoj iteraciji,
- s_j - udaljenost c_j od najbližeg drugog centra

Strukture koje se odnose na točke zadanog skupa:

- x_i - i-ta točka skupa $(1 \leq i \leq n)$

- a_i - indeks centra kojemu je točka x_i dodijeljena
- u_i - gornja ograda na udaljenost $d(x_i, c(a_i))$
- l_i - donja ograda na udaljenost točke x_i i drugog najbližeg centra toj točki

Slijedi pseudokod Hamerlyjevog algoritma preuzet iz članka [Ham10]:

Algorithm 1 K-means(dataset x, initial centers c)

```
1:  INITIALIZE(c, x, q, c', u, l, a)
2:  while not converged do
3:    for j = 1 to |c| do                                  {update s}
4:      s(j) ← min_{j'≠j} d(c(j'), c(j))
5:    for i = 1 to |x| do
6:      m ← max(s(a(i))/2, l(i))
7:      if u(i) > m then                                   {first bound test}
8:        u(i) ← d(x(i), c(a(i)))                          {tighten upper bound}
9:        if u(i) > m then                                 {second bound test}
10:         a' ← a(i)
11:         POINT-ALL-CTRS(x(i), c, a(i), u(i), l(i))
12:         if a' ≠ a(i) then
13:           update q(a'), q(a(i)), c'(a'), c'(a(i))
14:   MOVE-CENTERS(c', q, c, p)
15:   UPDATE-BOUNDS(p, a, u, l)
```

Algorithm 2 INITIALIZE(c, x, q, c', u, l, a)

```
1: for j = 1 to |c| do
2:   q(j) ← 0
3:   c'(j) ← 0⃗
4: for i = 1 to |x| do
5:   POINT-ALL-CTRS(x(i), c, a(i), u(i), l(i))
6:   q(a(i)) ← q(a(i)) + 1
7:   c'(a(i)) ← c'(a(i)) + x(i)
```

Algorithm 3 POINT-ALL-CTRS($x(i), c, a(i), u(i), l(i)$)

```
1: a(i) ← argmin_j d(x(i), c(j))
2: u(i) ← d(x(i), c(a(i)))
3: l(i) ← min_{j≠a(i)} d(x(i), c(j))
```

Algorithm 4 MOVE-CENTERS(c', q, c, p)

```
1: for j = 1 to |c| do
2:   c* ← c(j)
3:   c(j) ← c'(j)/q(j)
4:   p(j) ← d(c*, c(j))
```

Algorithm 5 UPDATE-BOUNDS(p, a, u, l)

```
1: r ← argmax_j p(j)
2: r' ← argmax_{j≠r} p(j)
3: for i = 1 to |u| do
4:   u(i) ← u(i) + p(a(i))
5:   if r = a(i) then
6:     l(i) ← l(i) − p(r')
7:   else
8:     l(i) ← l(i) − p(r)
```

Najveće ubrzanje se postiže kada (dvaput) provjeravamo da li je udaljenost točke od njenog dodijeljenog centra manja od donje ograde na udaljenost točke do drugog najbližeg centra ili od polovice udaljenosti dodijeljenog centra do njemu najbližem (linija 7 u "Algorithm 1" u pseudokodu). Ako je taj uvjet istinit, preskačemo najdublju algoritmu, tj. funkciju PointAllCtrs. Inače, ažuriramo gornju ogradu i opet provjeravamo uvjet da ne bismo nepotrebno ulazili u najdublju petlju. Dvostruka provjera ovog uvjeta je fundamentalna razlika Elkanovog i Hamerlyjevog algoritma te ona omogućava preskakanje najdublje petlje u većini slučajeva. Kod svakog pomicanja centra u funkciji MoveCenters bilježimo za koliko se centar pomaknuo te tu informaciju iskoristimo kod ažuriranja ograda u funkciji UpdateBounds. Gornja ograda neke točke se uveća za pomak njoj dodijeljenog centra. Donja ograda se umanji za najveći pomak osim u slučaju centra koji se najviše pomaknuo - tada se donja ograda umanji za drugi najveći pomak. Novi centroidi se definiraju na standardan način - kao aritmetičke sredine točaka koje se nalaze u istom klasteru. Također, Hamerlyjev algoritam ima nadogradnju "delta updates" koja nam osigurava da nemamo nepotrebnih zbrajanja vektora kada točke nisu mijenjale klastere.

2.3 compare-means i sort-means

Ideja ovog algoritma počiva na Orchardovoj metodi za pronalaženje najbližeg susjeda (Orchard 1991). Za danu točku x i potencijalnog najbližeg susjeda y zaključuje da neka druga točka z ne može biti bliža točki x od y ako vrijedi $||z-y|| > 2||x-y||$. Kako se u Lloydovom algoritmu zapravo za svaku točku traži najbliži susjed iz skupa centara, Orchardova metoda se može lako adaptirati za k-sredine. Taj algoritam se naziva compare-means. compare-means uspoređuje udaljenosti među centrima u svakoj iteraciji kako bi pametno izbjegao nepotrebna računanja udaljenosti između točaka i centara u sljedećoj iteraciji pomoću nejednakosti trokuta, kao što je i Orchard eliminirao potencijalne najbliže susjede. Ovaj pristup dovodi do većeg ubrzanja samo za manje dimenzije podataka, dok je za veće sporiji čak i od Lloydovog. Razlog tomu je potrebna memorija za kontroliranje i testiranje $O(k^2)$ udaljenosti među centrima. Sort-means poboljšava compare-means sortiranjem tablice udaljenosti među centrima. Time algoritam preskače centre koji nikako ne mogu biti najbliži danoj točki te štoviše prestane potpuno tražiti drugog kandidata za najbliži centar nakon što se prijeđe granica na udaljenost točke i centra. Za k koji nije prevelik, ovaj algoritam je dosta dobro poboljšanje.

2.4 Partial distance search

Partial distance search (PDS) je metoda zaustavljanja računanja udaljenosti na višedimenzionalnim skupovima kada zapravo računamo te udaljenosti da bismo ih usporedili s nekom minimalnom granciom. Ova metoda bi se mogla primijeniti u svim navedenim varijacijama k-sredina. U tijeku k-sredina, kada točki određujemo najbliži centar, mogli bismo stati u računanju udaljenosti točke i kandidata za novi najbliži centar kada parcijalna suma kvadrata norme prijeđe određenu granicu na tu udaljenost. Preciznije, za danu točku $x \in \mathbf{R}^d$, centar $c \in \mathbf{R}^d$ i udaljenost m želimo znati da li je $||x-c|| \leq m$, tj. kada je $||x-c||^2 = \sum_{i=1}^{d}(x_n - c_n)^2 < m^2$. Ako u nekom trenutku sumirajući po redu kvadrirane komponente vektora $x-c$ prijeđemo granicu m, možemo zaključiti da centar c ne može biti novi najbliži centar točki x pa se računanje zaustavlja. Ubrzanje ovom metodom se postiže tek za podatke velikih dimenzija.

3 Testni podaci i implementacija algoritama

3.1 Skupovi podataka

Implementirat ćemo Lloydov, Elkanov i Hamerlyjev algoritam u Matlabu. Podaci na kojima ćemo ih testirati preuzeti su s interneta (r15, pathbased, kddcup, covertype). Odabrani skupovi imaju različite strukture, dimenzije i broj podataka. Cilj nam je na njima vidjeti gdje je koji algoritam uspješniji, tj. zaključiti koji je algoritam najprikladniji za određene dimenzije podataka.

podaci	**broj podataka**	**dimenzija**	**kratak opis**
uniform random	50000	2	podaci iz uniformne distribucije
uniform random	50000	8	podaci iz uniformne distribucije
covtype	150000	55	klasifikacija pokrova zemlje
kddcup	95412	56	odazivi na donacije
pathbased	300	2	skup koji predstavlja oblik
r15	600	2	skup koji predstavlja oblik

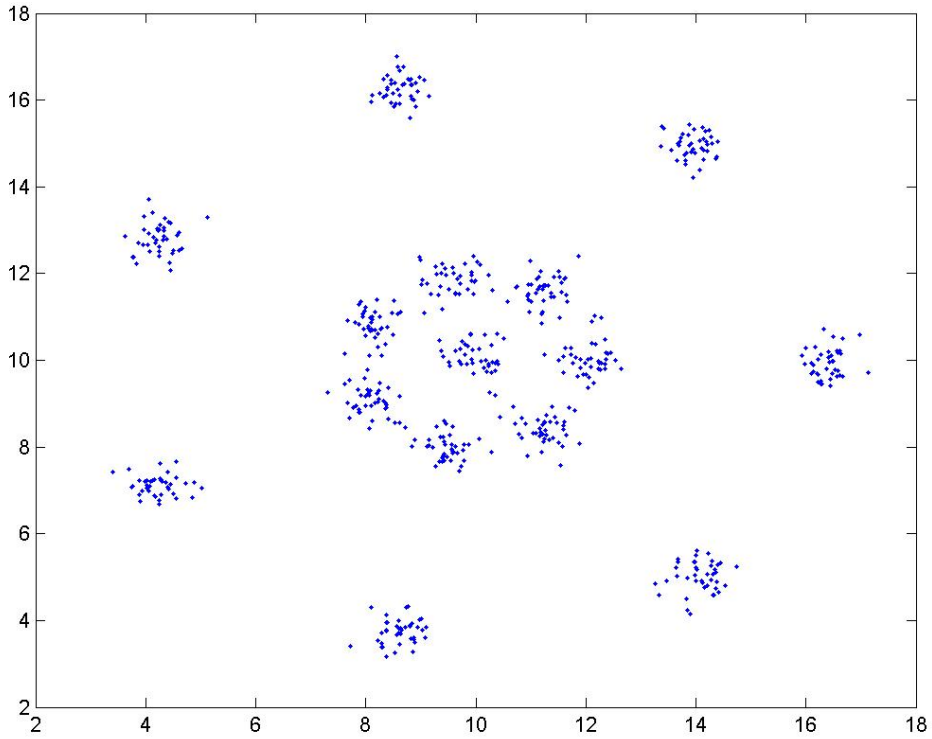
Tablica 1: Opis testnih podataka

3.2 Inicijalizacijske metode

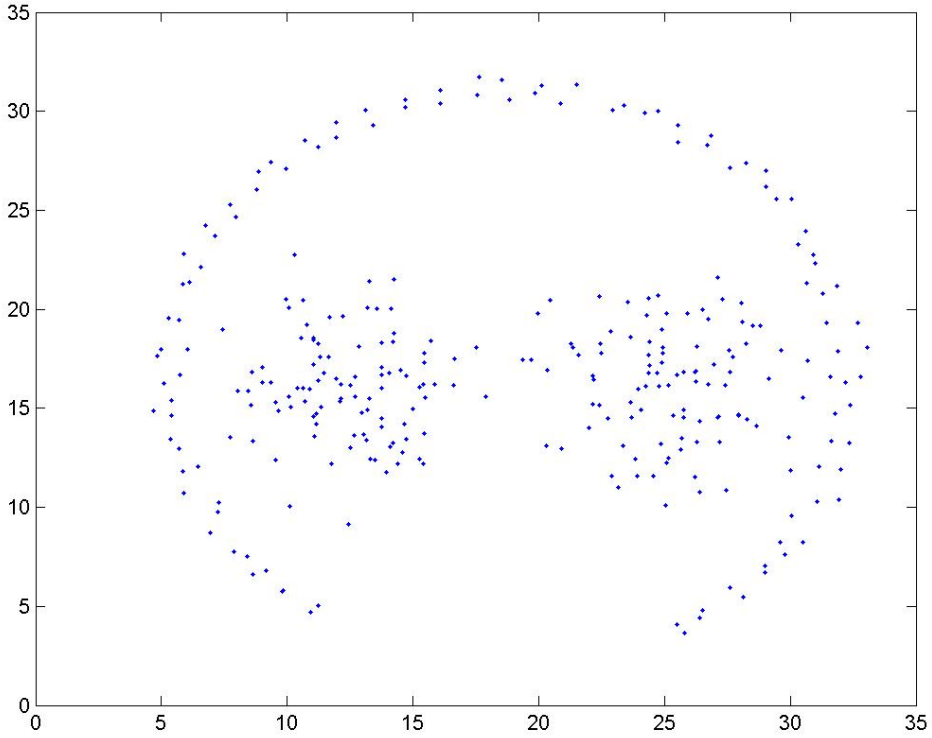
Predstavljeni algoritmi ne specificiraju inicijalne centre pa ih je potrebno definirati prije izvršavanja algoritma na neki način. Početna inicijalizacija je jako bitna za konvergenciju algoritma pa je poželjno što bolje odrediti početne centre. Postoji mnogo načina kako odabrati što bolju početnu inicijalizacije. Radi jednostavnosti, testirat ćemo podatke s centrima koje smo nasumično odabrali iz skupa podataka, kao što je to radio i Greg Hamerly pri testiranju implementacija. Spremit ćemo tako odabranu početnu inicijalizaciju za svaki slučaj koji testiramo da bismo mogli uspoređivati algoritme s istim početnim centrima.

3.3 Ispražnjenje klastera

Ponekad se u tijeku Lloydovog algoritma može dogoditi da se neki od klastera isprazni. U tom slučaju dolazi do dijeljenja s nulom (kada računamo aritmetičku sredinu). Kod k-sredina broj klastera je fiksan pa je potrebno odlučiti na koji način "napuniti" taj klaster. Najlakši način bi bio izabrati nasumičnu točku iz skupa, "izvaditi" je iz njenog klastera te proglasiti centroidom klastera koji se ispraznio.



Slika 1: skup podataka "r15"



Slika 2: skup podataka "pathbased"

4 Rezultati

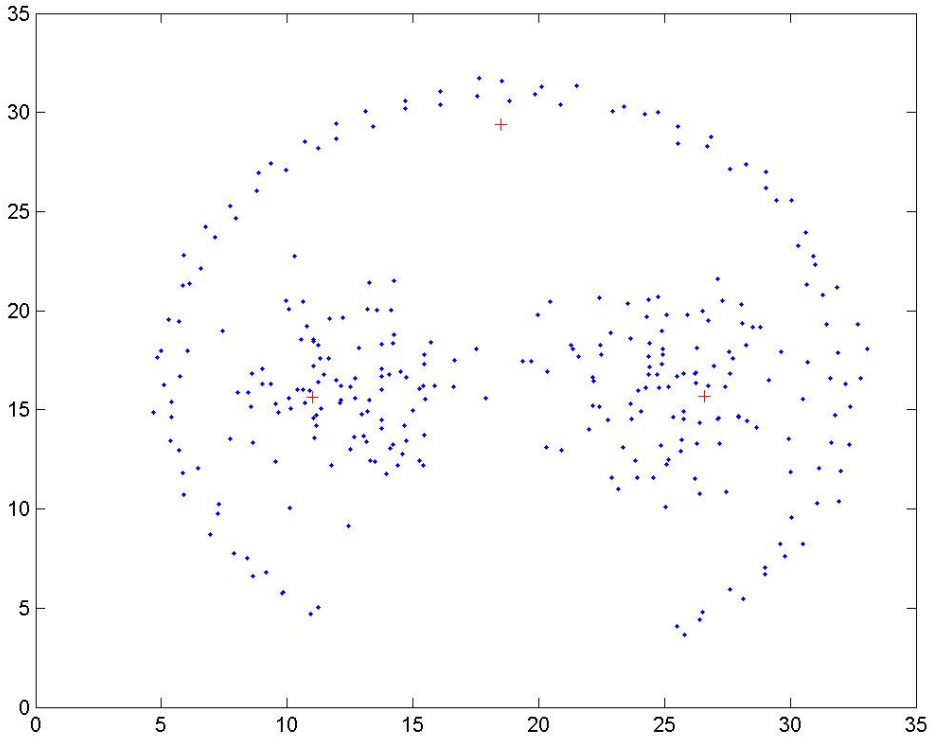
Naše implementacije Lloydovog, Elkanovog i Hamerlyjevog algoritma testirat ćemo na skupovima podataka različitih veličina i dimenzija. Skupovi generirani uniformnom distribucijom predstavljaju najgori slučaj za k-sredine jer taj skup nema nikakve strukture, dok ostali podaci dolaze iz primjene i za njih ima smisla tražiti klastere. Ipak, korisno je vidjeti kako se algoritmi ponašaju i u tom najgorem slučaju.

podaci		$k = 3$	$k = 20$	$k = 100$	$k = 500$
uniform random	Lloyd	113.23 s	929.58 s	5189.354 s	18427.7 s
n=50000	Elkan	188.19 s	904.89 s	-	-
d=2	Hamerly	25.36 s	264.33 s	-	-
uniform random	Lloyd	156.3 s	2157.29 s	6806.54 s	12998.5 s
n=50000	Elkan	263.72 s	-	-	-
d=8	Hamerly	163.79 s	3046.84 s	-	-
covtype	Lloyd	484 s	3331.47 s	41690.23 s	-
n=150000	Elkan	1304.83 s	49183 s	-	-
d=55	Hamerly	84.71 s	4442.98 s	82006 s	-
kddcup	Lloyd	109.1 s	2410 s	15357 s	-
n=95412	Elkan	463.8s	58325 s	-	-
d=56	Hamerly	24.23 s	2079 s	22799 s	-
pathbased	Lloyd	0.02 s	0.04 s	0.1 s	-
n=300	Elkan	0.23s	0.95 s	1.01s	-
d=2	Hamerly	0.08 s	0.32 s	0.46s	-
r15	Lloyd	0.03 s	0.06 s	0.29 s	0.61 s
n=600	Elkan	0.3 s	0.85 s	4.99 s	8.97 s
d=2	Hamerly	0.11 s	0.19 s	2.22 s	4.59 s

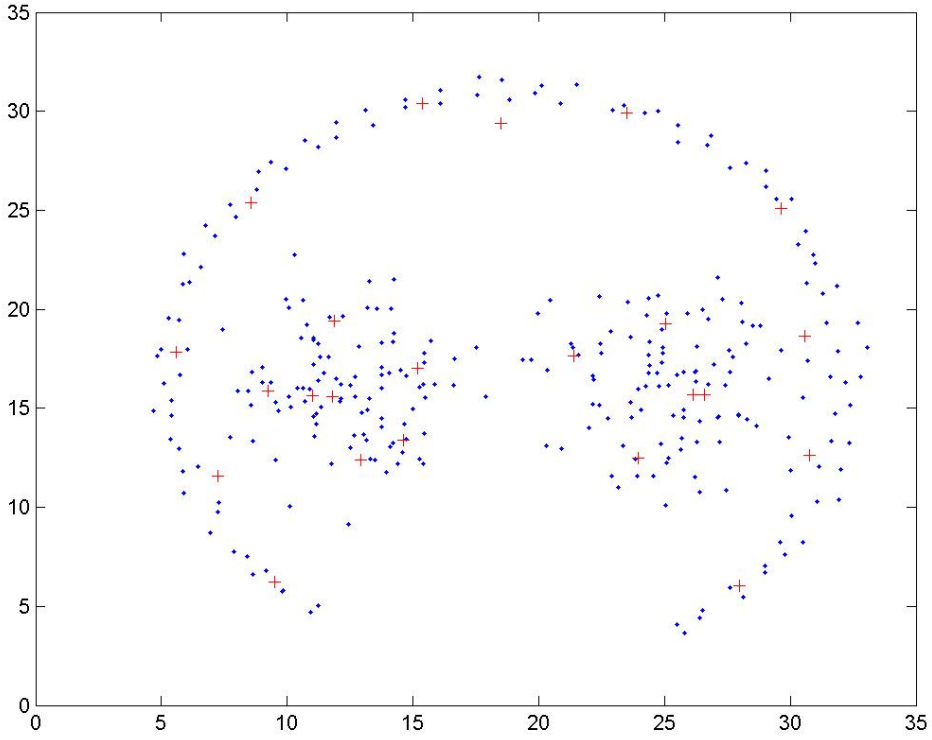
Tablica 2: Usporedba algoritama u vremenu izvršavanju

podaci		$k = 3$	$k = 20$	$k = 100$	$k = 500$
uniform random n=50000 d=2	Elkan Hamerly	49.26% 15.16%	49.54% 30.87%		
uniform random n=50000 d=8	Elkan Hamerly	99.99% 99.07%	 98.31%		
covtype n=50000 d=55	Elkan Hamerly	47.23% 22.03%	75.24% 60.16%	 76%	
kddcup n=95412 d=56	Elkan Hamerly	52.39% 17.96%	64.45% 52.30%	 85.49%	
pathbased n=300 d=2	Elkan Hamerly	44.19% 22.81%	48.54% 30.35%	50.58% 24.17%	
r15 n=600 d=2	Elkan Hamerly	70.19% 39%	36.88% 14.67%	50.32% 35.02%	38.94% 6.94%

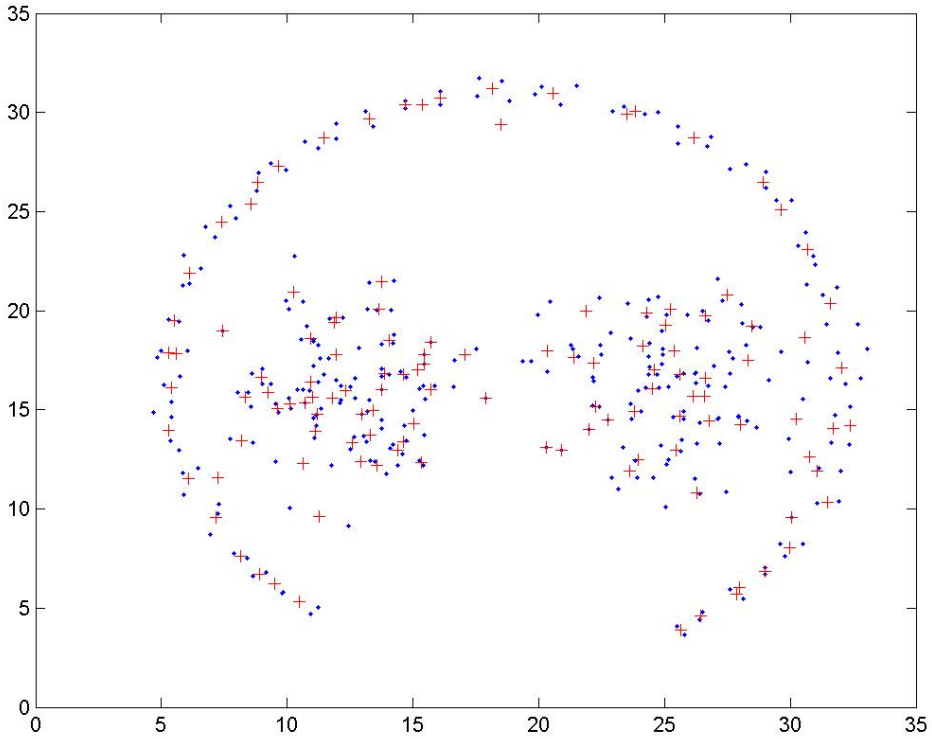
Tablica 3: Postotci ulaska algoritama u najdublju petlju



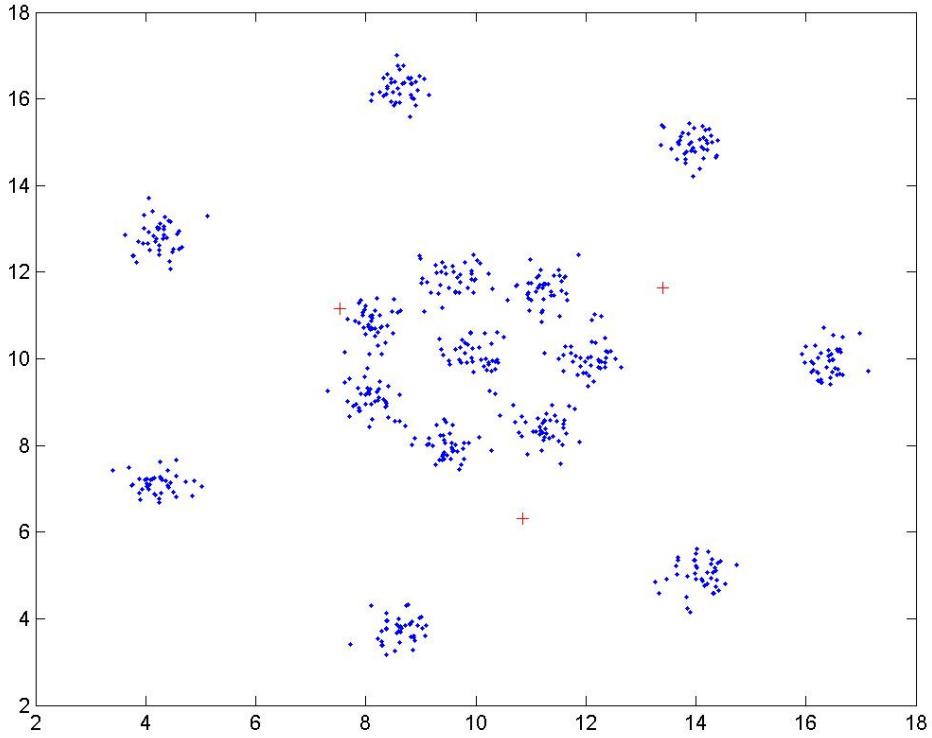
Slika 3: Skup podataka "pathbased" s centrima za k=3



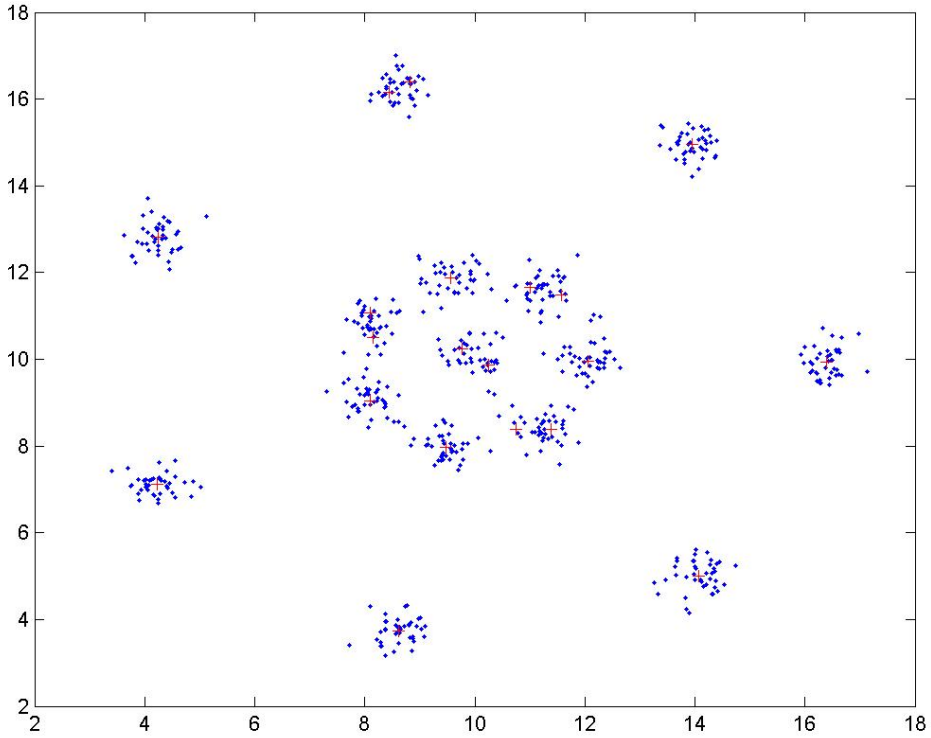
Slika 4: Skup podataka "pathbased" s centrima za k=20



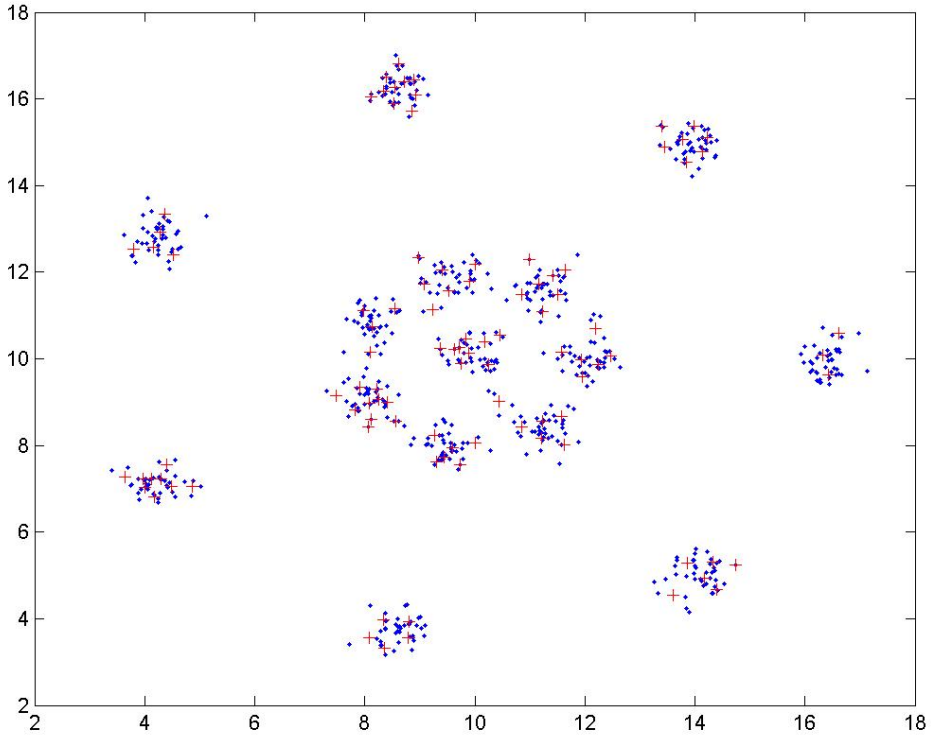
Slika 5: Skup podataka "pathbased" s centrima za k=100



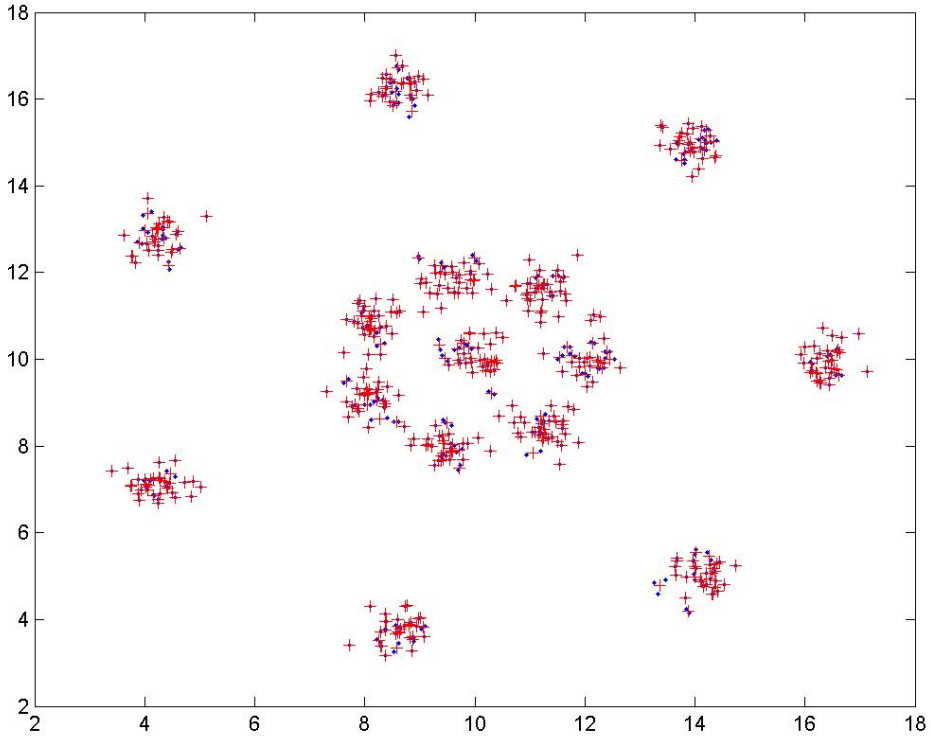
Slika 6: Skup podataka "r15" s centrima za k=3



Slika 7: Skup podataka "r15" s centrima za k=20



Slika 8: Skup podataka "r15" s centrima za k=100



Slika 9: Skup podataka "r15" s centrima za k=500

5 Zaključak

U ovom radu detaljno smo predstavile dvije nadogradnje Lloydovog algoritma koje su jednostavne za implementirati, a puno su efikasnije i brže.

Elkanov algoritam pomoću nejednakosti trokuta uvodi ograde na udaljenosti točaka od centara, odnosno među centrima, da bi izbjegao nepotrebne izračune udaljenosti. Za razliku od Elkanovog, Hamerly uvodi manji broj ograda: ograde na udaljenost točaka od najbližeg centra i drugog najbližeg centra. U Hamerlyjevom algoritmu manje se preskaču računanja udaljenosti pa s povećavanjem dimenzije podataka postaje sporiji od Elkanovog. Međutim, ograde iz Hamerlyjevog algoritma omogućavaju češće preskakanje dublje petlje od Elkanovog pa je zbog toga brži na manjim i srednjim dimenzijama.

Možemo reći da su Hamerlyjev i Elkanov algoritam u tom pogledu komplementarni. Greg Hamerly je predložio stvaranje hibridnog algoritma koji će spojiti prednosti oba ova algoritma; kraće vrijeme izvršavanja i na manjim i na većim dimenzijama. Na toj ideji se bazira "annulus algoritam".

Na našim podacima je uočeno da se najdublja petlja Hamerlyjevog algoritma preskače u prosječno 60% slučajeva (kada ne gledamo najgore slučajeve s uniformnom distribucijom), a u Elkanovom 50%, dok je vrijeme izvršavanja Elkanovog algoritma sporije od Hamerlyjevog i za veće dimenzije. Također, može se primijetiti da je na skupovima "pathbased" i "r15" koji su male dimenzije i veličine Lloydov algoritam ipak brži od svojih nadogradnji te brži i od Elkanovog na skupovima veće dimenzije. Razlog za odstupanja ovih rezultata od onih u radovima Elkana i Hamerlyja je razlika u baratanju s varijablama u našoj implementaciji u Matlabu. Kada bi kodovi bili u potpunosti vektorizirani i optimizirani u smislu korištenja Matlabovih rutina što efikasnije, smatramo da bi rezultati našeg rada bili sličniji rezultatima Elkana i Hamerlyja u svojim člancima.

Literatura

[Dra13] Jonathan Drake. Faster k-means clustering, master's thesis, 2013.

[Elk03] Charles Elkan. Using the triangle inequality to accelerate k-means. 2003.

[Ham10] Greg Hamerly. Making k-means even faster. 2010.