STROJNO UČENJE

5. Domaća Zadaća

Krešimir Špes
0036419866
ak. god. 2011. / 2012.

Za programski jezik sam odabrao python jer sam pored C++-a,najvještiji sam u tom jeziku.

Iako bi možda matlab bio bolje rješenje radi vizualizacije primjera, brži sam s pythonom.

Zbog potrebe vizualizacije primjera odlučio sam se na pure-python rješenje za pisanje TGA datoteka. TGA format je jako jednostavan, sličan BMP-u. Alternativa je korištenje python image library-a, ali ako ćete htjeti pokretati program onda morate instalirati taj modul, pa rekoh, da ne kompliciram.

Koristio sam gotovu implementaciju python TGA zapisivanja, preuzeto sa sljedeće adrese:

http://www.python.org.br/wiki/ImagemTGA

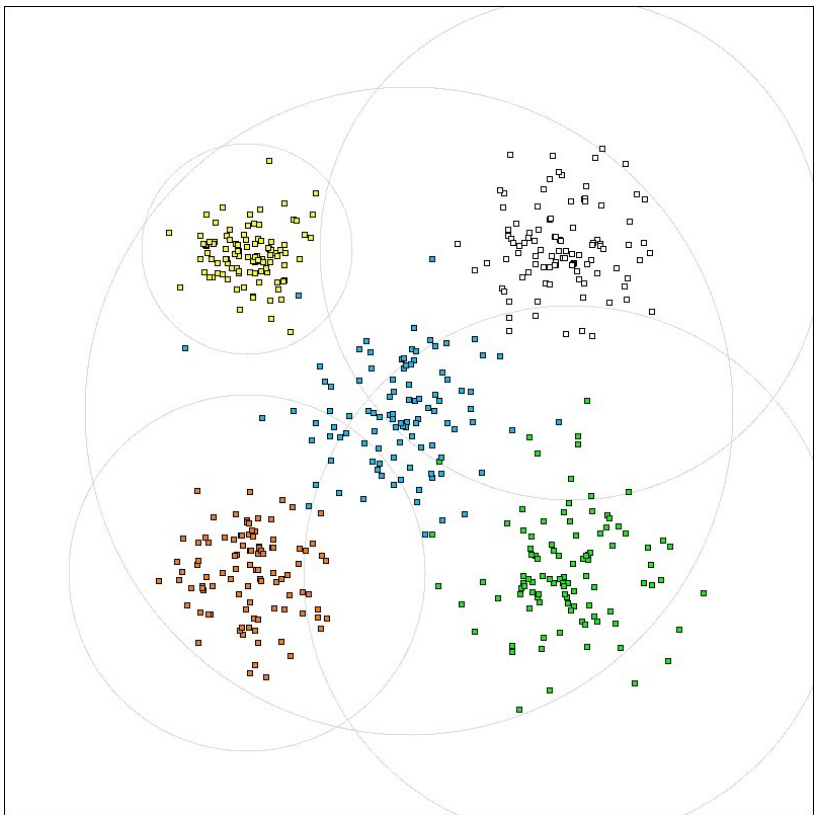
Kod sam u određenoj mjeri modificirao za potrebe ovog zadatka. Taj kod se nalazi u zad3/tga.py

Implementacija je rađena na MacOSX 10.7.2 i pythonu 2.7.1

a) Za generiranje primjera iz gaussovih izvora, korištena je funkcija random.gauss, standardna funkcija pyhonovog random modula.

Umjesto nasumičnih centara, odlučio sam se ipak za fiksan raspored centara radi lakše vizualizacije. Standardne devijacije distribucija izvora sam prilagodio tako da se vizualno mogu uočiti grupe ali da dolazi do djelomičnog preklapanja pojedinih grupa što bi trebalo malo zbuniti algoritme grupiranja a time i lakše vrednovati rezultate.

Slika generiranih primjera (krugovi označavaju gaussove izvore):



b) implementacija se nalazi u zad3/kmeans.py

Početna središta odabrao sam prvo nasumično u rasponu [0,100] što je dalo šarolike rezultate. Za tako postavljene centroide algoritam je konvergirao između 5 i 25 iteracija, no veći problem je nastao u tome ako je neki centroid predaleko postavljen, dogodilo se da je u tom centroidu 0 primjera dok su se ostali rasporedili po grupama.

Kasnije sam implementirao pametniji pristup odabira centroida, kako je sugerirano u bilješci, uz male modifikacije:

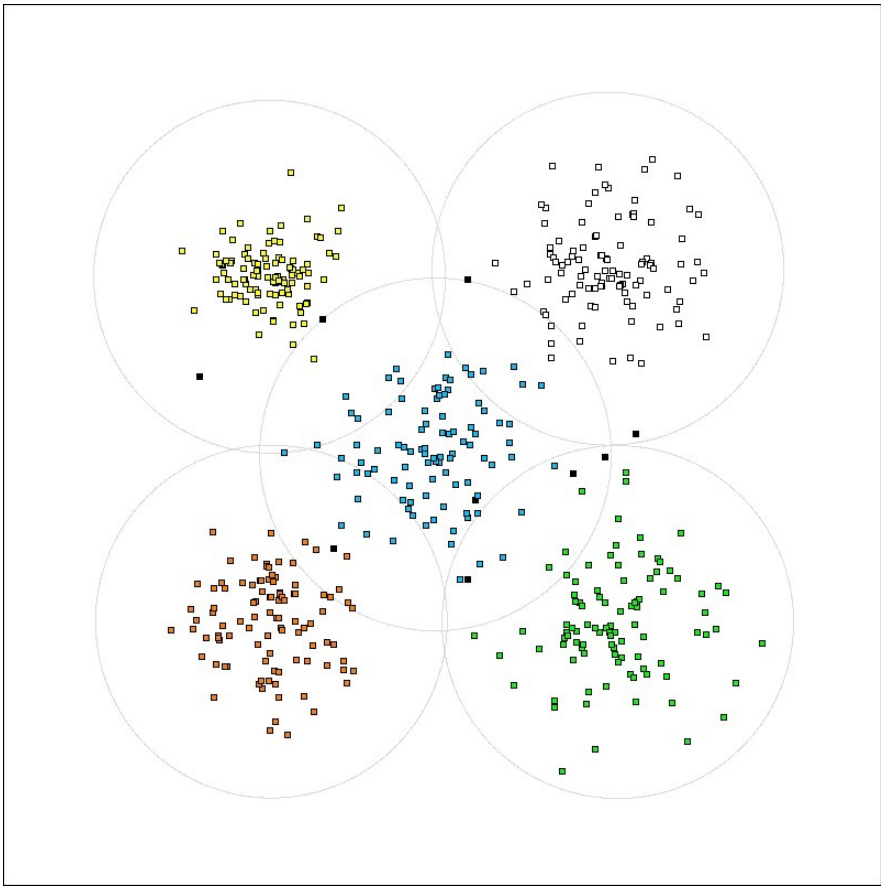
1) izračuna se središte svih uzoraka
2) procijeni se radijus kruga koji bi obujmio sve primjere
3) iz središta uzoraka odabiru se početni centroidi tako da se računaju točke na kružnici koju odgovaraju 360/K kutevima te kružnice.
4) Kako bi se uvelo malo nedetrminizma, variram te kuteve za +- 25% te radijus kruga između 75%-100% procijenjenog radijusa

Na taj način algoritam konvergira u 6-12 koraka i za zadane prijmjere uvijek nađe dobre grupe, tj. Ne događaju se prazne grupe.

Koristeći nasumično postavljene centroide, u najgorem slučaju kriterijska funkcija je iznosila 58838.1

Najmanja vrijednost kriterijske funkcije je **27979.45**, koja se dobije ili na sreću korištenjem nasumično postaljenim centroidima ili pametnijim pristupom uvijek (uz sitne oscilacije zbog nedeterminističkog odabira pozicije centroida u oba slučaja)

Slika rezultata grupiranja (crno označeni krivo grupirani primjeri):



c)

K	1	2	3	4	5	6	7	8	9	10
J(K)	338085.1	195935.5	128794.5	57576.1	27979.4	26004.0	24266.2	23239.2	21096.8	18552.3
K*	338089.1	195943.5	128806.5	57592.1	27999.4	26028.0	24294.2	23271.2	21132.8	18592.3

Nažalost, iz aikikeovog kriterija za k-means algoritam dan u petoj bilješci:

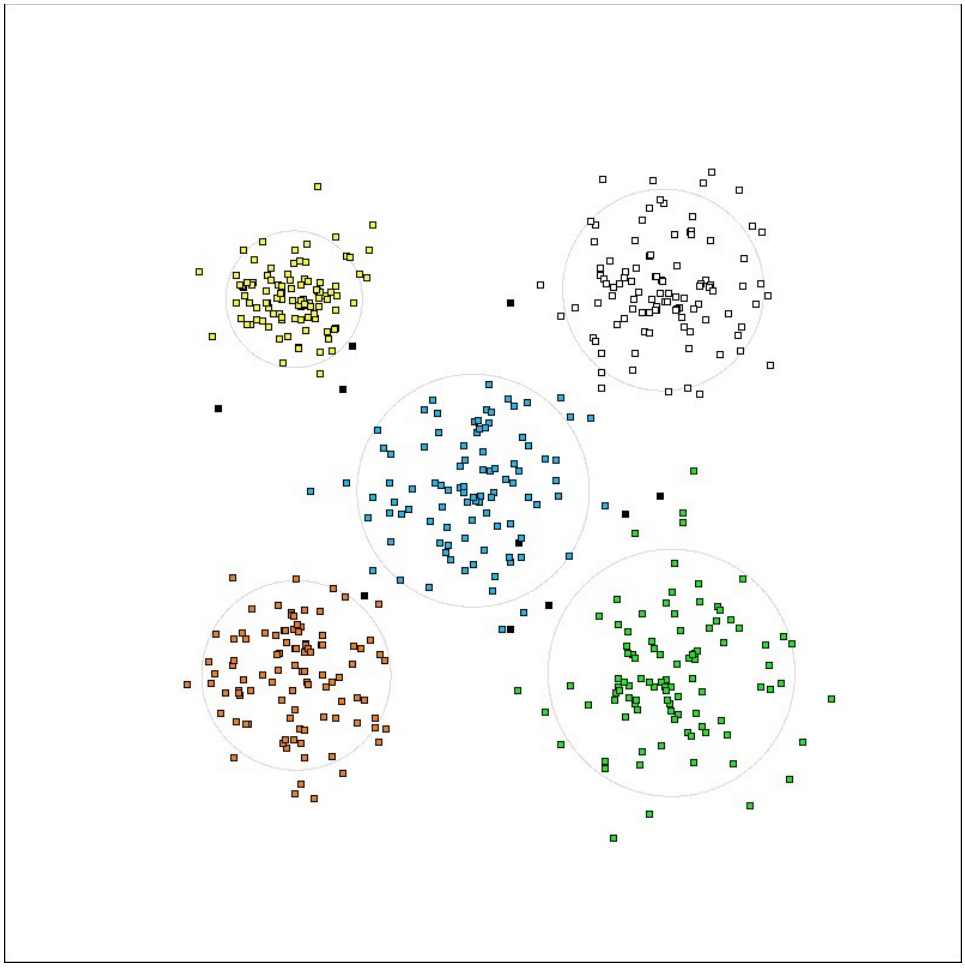
$$K^* = \underset{K}{\operatorname{argmin}} \left(J(K) + 2nK \right)$$

jednostavno ne možemo ništa zaključiti u ovom primjeru jer kao što se može vidjeti iz tablice, K* monotono opada. 2nK je jednostavno premali korekcijski faktor da utječe na kriterijsku funkciju kmeans algoritma. Ili bi taj dio kriterija trebao biti pomnožen za nekim faktorom koji bi značajnije utjecao na strukturno ograničenje ili jednostavno aikikeov kriterij ne funkcionira dobro kod primjera sa niskom dimenzionalnošću.

Idealno, što pretpostavljam da je bio cilj ovog zadatka bi trebalo biti tako da za određeni broj grupa K* prestane opadati i počne rasti. U toj točki “koljena” bi trebao biti (po tom kriteriju) prirodan broj grupa koji postoji u primjerima.

d) kod za ovaj zadatak dan je u zad3/em.py.

U slici ispod se vidi vizualni prikaz rezultata EM algoritma za K = 5. Koristio sam rezultate iz kmeans algoritma kako bih postavio centre distribucija što je rezultiralo u prilično točnom grupiranju. Algoritam je konvergirao u 5 iteracija.



Eksperimentirao sam sa nasumično postavljenim centrima i kako je bilo za očekivati, algoritam je pronašao lokalne optimume. Možda bi umjesto gradijentnog spusta bilo bolje koristiti genetske ili slične “pametnije” optimizacijske algoritme.

Uglavnom, kada sam postavio centre nasumično, često se dogodilo da grupa postavljena u sredini prostora naraste i istisne sve ostale grupe. Modificirao sam malo algoritam tako da ako se dogodi da se neka distribucija smanji na neznačajni dio, da se izbaci. Pa tako uz K = 20 algoritam češće konvergira i izbacuje loše postavljene grupe. Tako npr od K = 120 dobijem 5 grupa u 90% slučajeva, 5 grupa koje su dobro postavljeni, slično kao i kod gornje slike.

U prosjeku algoritam sa nasumično postavljenim centrima distribucija konvergira između 20 i 40 iteracija.

Log izglednost nakon izvođena algoritma sa “pametno” postavljenim centrima distribucija iznosi -2568.52.

e) Kod za izračun randovog indexa se nalazi u zad3/kmeans.py odnosno zad3/em.py, na dnu svake datoteke.

Randov index za k-means algoritam: **0.986**

Randov index za EM algoritam: **0.984**

Dobili smo vrlo slične randove indexe, što je moguće i zaključiti po slikama za svaki algoritam. Razlog tome, rekao bih, je taj što su ta 2 algoritma u suštini vrlo slična. Možemo čak reći da je EM poopćenje k-means algoritma jer radi isto što i kmeans uz dodatak soft granica i mogućnost kompleksnijih modela poput gaussa.

U ovom primjeru, generirani podaci su vizualno dosta odvojeni/grupirani, možda kada bi razlike između grupa bile manje, EM algoritam bi imao veću uspiješnost u odnosu na k-means.