Grafovi za prepoznavanje i pretragu sličnih entiteta

1 Uvod

Potreba za pronalaženjem sličnih entiteta u velikim skupovima podataka je prisutna u mnogim oblastima, poput preporučivanja proizvoda, pretraživanja teksta, ili pretraživanja sličnih slika (npr. kada korisnik želi da pronađe slike koje su slične onoj koju je uneo).

Kako bi se efikasno izvršila pretraga nad velikom količinom podataka, tokom godina su razvijane različite metode i korišćene različite strukture podataka. Ovde će biti predstavljeni HNSW grafovi i KNN pretraga nad njima. Biće prikazana pretraga nad skupom slika, sa ciljem prepoznavanja osoba na osnovu njihovih slika lica.

2 Predstavljanje podataka

Sliku lica je prvo potrebno predstaviti u prostoru koji je niže dimenzije, a u kom je moguće zadržati sve informacije koje su neophodne za prepoznavanje. Na primer, moguće ih je predstaviti kao vektore realnih brojeva. Za te potrebe se koriste neuronske mreže koje su obučene da izdvajaju bitne karakteristike slike (*eng. feature embeddings*). Neki od javno dostupnih modela za izdvajanje bitnih karakteristika lica su: SphereFace, CosFace, FaceNet, ArcFace, VGGFace, DeepFace.

Za potrebe ovog primera je uzeta varijanta FaceNet-a koja konvertuje sliku u 512-dimenzioni vektor, $x_i \in \mathbb{R}^{512}$.

3 KNN pretraga

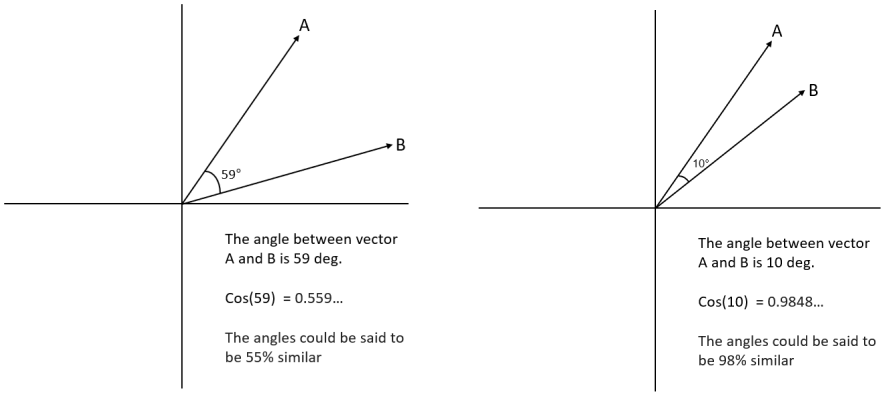
Algoritmi za pretragu koji se najčešće koriste se oslanjaju na algoritam pretrage K najbližih suseda (KNNS - *K-Nearest Neighbor Search*). KNN pretraga podrazumeva da je definisana funkcija na osnovu koje može da se meri rastojanje između elemenata koji se pretražuju.

Za određivanje distance između vektorskih reprezentacija dve slike lica, najčešće se koristi kosinusna distanca. Kosinusna distanca između dva vektora se računa kao kosinus ugla između njih. Definisana je kao:

$$
\begin{aligned}
cos\_distance &= 1 - cos\_similarity \\
&= 1 - cos(\theta) \\
&= 1 - \frac{A \cdot B}{||A||||B||} \\
&= 1 - \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^n} \sqrt{\sum_{i=1}^{n} B_i^n}}
\end{aligned}
.
$$

gde su A i B vektori koji odgovaraju slikama koje se porede, a θ ugao između njih.

Geometrijski prikaz ovoga, u dvodimenzionom slučaju, se može videti na slici 1:



Slika 1: Levo: ugao između vektora je veći, što znači da su oni međusobno dalji i da su slike kojima odgovaraju manje slične. Desno: ugao između vektora je manji, što znači da su oni međusobno bliži i da su slike kojima odgovaraju sličnije.

Kosinusna distanca je uvek u intervalu $[0, 2]$, a što je manja, to su slike sličnije. Ukoliko su slike identične, kosinusna distanca je 0, a ukoliko su slike potpuno različite, kosinusna distanca je 2.

Alternativno, može se koristiti i L_2 norma.

Naivni pristup KNN pretrage se zasniva na tome da se za svaki element iz skupa podataka izračuna rastojanje od svih ostalih elemenata, i da se zatim izabere K elemenata koji su najbliži datom elementu. Nažalost, složenost ovog pristupa raste linearno sa porastom broja elemenata u skupu podataka, čineći ga neupotrebljivim za realne primene. Zbog toga se koriste različite strukture

podataka koje omogućavaju bržu pretragu, kao i aproksimacije pretrage.

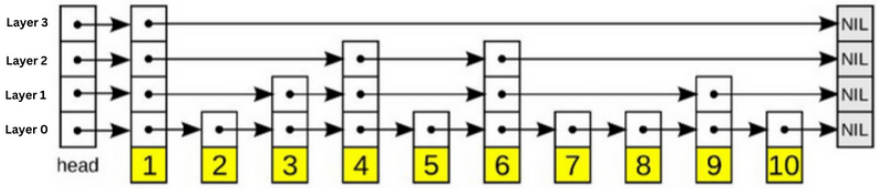
4 Aproksimativna KNN pretraga pomoću HNSW grafova

HNSW (*Hierarchical Navigable Small World*) graf predstavlja *state-of-the-art* strukturu podataka za aproksimativnu KNN pretragu. HNSW graf je struktura podataka koja se sastoji iz više nivoa.

HNSW uzima koncept pretrage od *skip* listi.

4.1 *Skip* lista

Skip lista je struktura podataka koja omogućava brzu pretragu, a sastoji se od više nivoa. Viši nivoi sadrže manje elemenata, između kojih su uspostavljene duže konekcije. Kako se spuštamo na niže nivoe, broj elemenata raste, a konekcije postaju kraće. Najniži nivo sadrži sve elmente originalne liste. Primer *skip* liste se može videti na slici 2:



Slika 2: Primer *skip* liste

Pretraga za nekim elementom k počinje od najvišeg nivoa. Kada nađemo element koji je veći od k, vraćamo se na prethodni manji element, spuštamo se na niži nivo, i nastavljamo pretragu od tog elementa. Ovaj postupak ponavljamo sve dok se ne pronađe traženi element.

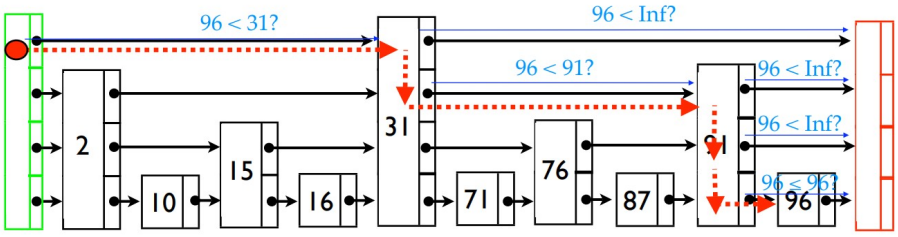
Pseudokod za pretragu izgleda ovako:

```
1: If k = key: done;
2: k < next key: go down a level;
3: k ≥ next key: go right;
```

Ovime se postiže da složenost pretrage bude O(log n), umesto O(n) kakav je slučaj sa običnim listama, s tim što se prostorna složenost poveća sa O(n) na O(n log n) . *Skip* liste su primenljive samo nad sortiranim listama.

Primer jedne *skip* liste i pretrage za brojem 96 je prikazan na slici 3:

Dodavanje novih elmenata se određuje probabilistički. Za svaki novi element prvo moramo da odredimo nivo na koji ćemo ga dodati. Verovatnoća izbora

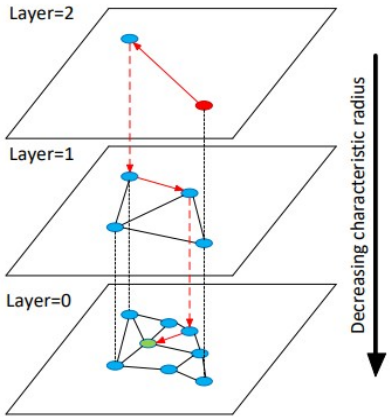


Slika 3: Primer pretrage *skip* liste

najvišeg nivoa je uvek najmanja, dok se verovatnoća izbora svakog narednog nivoa povećava. Opšte pravilo je da će se element koji se nalazi na nekom nivou l, pojaviti na nivou iznad njega (l+*1* nivou) sa nekom predefinisanom verovatnoćom p. A takođe važi i sledeće: Ako je element dodat na nivo l, znači da će biti dodat i na sve nivoe ispod njega (l-*1*, l-*2*, itd).

4.2 HNSW graf

Sličan koncept se primenjuje i pri kreiranju i pretrazi HNSW grafa. On se takođe sastoji iz više nivoa, s tim što se na svakom nivou umesto liste nalazi graf. Primer HNSW grafa se može videti na slici 4:



Slika 4: Primer HNSW grafa

Čvorovi grafa predstavljaju *feature* vektore koji odgovaraju slikama lica. Grane kojima su čvorovi povezani odgovaraju kosinusnim distancama između tih vektora.

4.2.1 Pretraga HNSW grafa

Pretraga se odvija po sledećem principu:

1. Pretraga počinje od slučajno odabranog čvora na najvišem nivou. Najviši nivo ima najmanje čvorova i najduže veze između njih, dok svaki naredni sloj ima sve više čvorova i sve kraće veze.

2. Najviši sloj se pretražuje sve dok se ne pronađe lokalni minimum – čvor (vektor) koji ima najmanju distancu od traženog čvora (vektora). Odnosno, dok se ne pronađe traženi broj najbližih čvorova. Pretraga sloja se vrši heuristički.

   Prilikom pretrage se čuva dinamička lista W koja sadrži ef najbližih suseda. Lista se u svakom koraku ažurira na osnovu evaluacije suseda elemenata liste koji su prethodno dodati. Kada se lista dostigne maksimalnu veličinu od ef elemenata, ako se naidje na element koji je bliži traženom čvoru od najdaljeg elementa liste, taj najdalji element će biti zamenjen njime. Kada se evaluiraju svi susedi svakog elementa liste, pretraga sloja se završava.

   Prednost ovakvog kriterijuma zaustavljanja je u tome što se neće evaluirati svi čvorovi grafa, već će se odmah odbacivati svi oni koji se od traženog čvora nalaze dalje od najdaljeg elementa liste W.

3. Nakon što se pronađe lokalni minimum, pretraga se nastavlja na narednom (nižem) sloju, od čvora koji je predstavljao lokalni minimum na prethodnom sloju.

4. Postupak se ponavlja sve dok se ne dođe do najnižeg sloja, kada se vraća K suseda najbližih traženom čvoru.

Ako je maksimalan broj konekcija po čvoru na svakom od slojeva konstantan, složenost pretrage je O(log n), gde je n broj čvorova u grafu.

Procedura pretrage je opisana u algoritmima 5 i 2.

Pre nego što krenemo u detaljniju analizu procesa kreiranja i pretrage grafa, definišimo neke od osnovnih parametara koji se koriste:

- ef_{search} - određuje broj čvorova koji će se pretraživati na svakom nivou. Što je ovaj broj veći, pretraga će biti preciznija, ali sporija. (Oznaka ef dolazi od pojma *exploratory factor*)

- $ef_{construction}$ - određuje broj čvorova koji će se pretraživati prilikom kreiranja grafa. Što je ovaj broj veći, graf koji se kreira će biti precizniji, ali će kreiranje trajati duže. Idealno, ova vrednost treba da bude dovoljno velika da bi *recall* KNN pretrage bio što bliži 1.

- m_L - maksimalan broj konekcija po čvoru na svakom od slojeva. Što je ovaj broj veći, pretraga će biti preciznija, ali sporija i graf će zauzimati više memorije.

4.2.2 Kreiranje HNSW grafa

Insertovanje elemenata u HNSW graf se odvija po sledećem principu:

1. Za svaki element koji se insertuje se prvo odredi maksimalni sloj l na kom će se nalaziti. To se radi probabilistički – l se bira iz eksponencijalno opadajuće raspodele (normalizovane m_L parametrom).

$$l = \lfloor -ln(unif(0..1)) \cdot m_L \rfloor$$

2. Zatim sledi pretraga za najbližim susedima elementa q koji se insertuje.

3. Prva faza pretrage ide od najvišeg sloja ka sloju l. Ovime se dobija inicijalni element ep koji je najbliži čvoru q.

4. U sledećoj fazi pretrage se polazi od sloja l (ako je on ispod L) i ide ka nižim slojevima, pri čemu pretraga počinje od prethodno pronađenog elementa ep. (Ako se desi da je l iznad L, čvor q se insertuje kao polazni čvor hnsw-a.)

5. Ovime se u svakom sloju pronalazi $ef = ef_{construction}$ najbližih suseda čvoru q.

6. Zatim se kreiraju grane od q ka pronađenim susedima u trenutnom sloju (pri čemu se vodi računa o maksimalnom dozvoljenom broju konekcija koje q može da ima).

7. Pretraga u narednom sloju se nastavlja počevši od najbližih suseda pronađenih u prethodnom nivou.

Proces kreiranja HNSW grafa je prikazan algoritmima 1 i 4.

Algoritam 1 INSERT(*hnsw, q, M, Mmax, efConstruction,* m_L)

Opis: Insertovanje novog elementa u HNSW graf

Ulaz: *hnsw* – graf, q – novi element, M – broj uspostavljenih konekcija, M_{max} – maksimalan broj konekcija svakog elementa po nivou grafa, $efConstruction$ – veličina dinamičke liste kandidata, m_L – faktor normalizacije koji se koristi pri generisanju nivoa grafa;

Izlaz: *hnsw* graf koji sadrži element q;

$W \leftarrow \emptyset$ // *lista trenutno nađenih najbližih suseda*
$ep \leftarrow$ slučajno odabrana početna tačka pretrage; // enter point
$L \leftarrow$ nivo čvora ep // *najviši sloj grafa*
$l \leftarrow \lfloor -ln(unif(0..1)) \cdot m_L \rfloor$ // *nivo na koji će biti insertovan novi element*
for $l_c \leftarrow L...l+1$ **do**
  $W \leftarrow SEARCH\_LAYER(q, ep, ef = 1, l_c)$ // *pohlepna pretraga*
  $ep \leftarrow$ element iz W koji je najbliži elementu q
end for
for $l_c \leftarrow min(L, l)...0$ **do**
  $W \leftarrow SEARCH\_LAYER(q, ep, efConstruction, l_c)$
  $neighbours \leftarrow SELECT\_NEIGHBOURS(q, W, M, l_c)$ // alg. 3 ili 4
  Kreirati grane između čvora q i čvorova iz liste $neighbours$ na nivou l_c
  for each $e \in neighbours$ **do** // *ukoliko je potrebno, smanjiti broj grana*
    eConn $\leftarrow$ neighbourhood(e) na nivou l_c
    if then$|eConn| > M_{max}$ // *smanjiti broj konekcija čvora e (ako je* $l_c = 0$ *onda je* $M_{max} = M_{max0}$*)*
      $eNewConn \leftarrow SELECT\_NEIGHBOURS(e, eConn, M_{max}, l_c)$
      Zameniti element $neighbourhood(e)$ na nivou l_c sa $eNewConn$
    end if
  end for
  $ep \leftarrow W$
end for
if $l > L$ **then**
  postavi q kao polaznu tačku za *hnsw*
end if

Algoritam 2 SEARCH_LAYER(q, ep, ef, l_c)

Opis: Algoritam pretrage datog sloja grafa za najbližim susedima datog elementa q

Ulaz: q – element čije najbliže susede tražimo, ep – ulazni čvorovi od kojih će pretraga krenuti, ef – broj najbližih suseda koje treba vratiti, l_c – nivo grafa na kom se vrši pretraga;

Izlaz: ef najbližih suseda elementu q

```
1: v ← ep // skup posećenih elemenata
2: C ← ep // skup kandidata
3: W ← ep // dinamička lista pronađenih najbližih suseda
4: while |C| > 0 do
5:    c ← element iz C koji je najbliži elementu q
6:    f ← element iz W koji je najdalji elementu q
7:    if distance(c, q) > distance(f, q) then
8:       // Svi elementi iz W su evaluirani.
9:       // Ako je najbliži kandidat dalji od najdaljeg kandidata
10:      // znaci da vise nece biti poboljsanja i
11:      // možemo da prekinemo pretragu.
12:      break
13:   end if
14:   for each e ∈ neighourhood(c) na nivou l_c do
15:      // Treba evaluirati svakog suseda kandidata c
16:      // i na osnovu toga ažurirati C i W
17:      if e ∉ v then // ako taj sused nije vec posecen
18:         v ← v ∪ e
19:         f ← element iz W koji je najdalji elementu q
20:         if distance(e, q) > distance(f, q) or |W| < ef then
21:            // Ako je novi kandidat bliži od najdanljeg već uzetog kandidata
22:            // (ili ako nisamo sakupili dovoljan broj kandidata),
23:            // dodaćemo ga u skup kandidata
24:            C ← C ∪ e
25:            W ← W ∪ e
26:            if |W| > ef then
27:               // Ako smo dodavanjem novog kandidata prešli
28:               // definisani maksimalni broj, uklonićemo najdaljeg kandidata
29:               Ukloniti element iz W koji je najdalji od q
30:            end if
31:         end if
32:      end if
33:   end for
34: end while
35: return W
```

Algoritam 3 SELECT_NEIGHBOURS_SIMPLE(*q, C, M*)

Opis: Jednostavan algoritam za pronalazak najblizih suseda

Ulaz: q – element čije najbliže susede tražimo, C – lista kandidata, M – broj suseda koje treba vratiti

Izlaz: M elemenata iz C koji su najblizi q

Algoritam 4 SELECT_NEIGHBOURS_HEURISTIC(*q, C, M,* l_c*, extendCandidates, keepPrunedConnections*)

Opis: Algoritam za pronalazak najblizih suseda zasnovan na heuristici koja uzima u obzir distance izmedju kandidata kako bi kreirala konekcije u raznovrsnim pravcima

Ulaz: q – element čije najbliže susede tražimo, C – lista kandidata, M – broj suseda koje treba vratiti, *extendCandidates* – indikator koji govori da li treba prosiriti listu kandidata, *keepPrunedConnections* – indikator koji govori da li treba dodati elmente koji su vec bili odbaceni

Izlaz: M elemenata iz C koji su najblizi q, odabranih heuristikom

1: $R \leftarrow \emptyset$
2: $W \leftarrow C$ // *red koji ce sadrzati kandidate*
3: **if** extendCandidates **then**
4: // *prosiri skup kandidata dodavanjem njihovih suseda*
5: **for each** $e \in C$ **do**
6: **for each** $e_{adj} \in neighbourhood(e)$ na nivou l_c **do**
7: **if** $e_{adj} \notin W$ **then**
8: $W \leftarrow W \cup e_{adj}$
9: **end if**
10: **end for**
11: **end for**
12: **end if**
13: $W_d \leftarrow \emptyset$ // *red za odbacene kandidate*
14: **while** $|W| > 0$ and $|R| < M$ **do**
15: $e \leftarrow$ element iz W koji je najblizi q
16: **if** e je blize q u odnosu na sve elmente iz R **then**
17: $R \leftarrow R \cup e$
18: **else**
19: $W_d \leftarrow W_d \cup e$
20: **end if**
21: **end while**
22: **if** keepPrunedConnections **then**
23: // *Dodati neke (najblize) kandidate koji su bili odbaceni*
24: **while** $|W_d| > 0$ and $|R| < M$ **do**
25: $R \leftarrow R \cup$ elment iz W_d koji je najblizi q
26: **end while**
27: **end if**

Algoritam 5 KNN_SEARCH(*hnsw, q, K, ef*)

Opis: KNN pretraga HNSW grafa

Ulaz: *hnsw* – graf, q – element čije najbliže susede tražimo, K – broj najbližih suseda koje treba vratiti, ef – veličina dinamičke liste kandidata

Izlaz: K elemenata koji su najbliži q

1: $W \leftarrow \emptyset$
2: $ep \leftarrow$ početna tačka pretrage
3: $L \leftarrow$ nivo na kom je ep *// najviši nivo za hnsw*
4: **for** $l_c \leftarrow L...1$ **do**
5: *// Prvo radimo za jednim najblizim susedom krecuci se kroz sve slojeve*
6: $W \leftarrow SEARCH\_LAYER(q, ep, ef = 1, l_c)$
7: $ep \leftarrow$ element iz W koji je najbliži q
8: **end for**
9: *// Ponovo radimo pretragu, sada pocevsi od prethodno pronadjenog nablizeg cvora, i trazimo ef nablizih cvorova*
10: $W \leftarrow SEARCH\_LAYER(q, ep, ef, l_c = 0)$
11: **return** K elemenata iz W koji su nabliži q