Κ23α - Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Χειμερινό Εξάμηνο 2024 2025

Άσκηση 2 – Παράδοση: Δευτέρα 9 Δεκεμβρίου 2024

Αναζήτηση εγγύτερου γείτονα (Nearest Neighbor Search)

Στο πρόβλημα αναζήτησης εγγύτερων γείτονων, μας δίνεται ένα σύνολο δεδομένων P με n σημεία σε κάποιο μετρικό χώρο.. Με δεδομένο ένα ερώτημα xq∈Rd και k∈N, θα θέλαμε να επιστρέψουμε τους k πλησιέστερους γείτονες του xq από το P. Για να βρούμε ακριβώς τους k πλησιέστερους γείτονες, η καλύτερη δυνατή μέθοδος είναι από μια γραμμική σάρωση του συνόλου δεδομένων λόγω του "dimensionality curse". Επομένως, στην πράξη, οι αλγόριθμοι σχεδιάζονται για την κατά προσέγγιση αναζήτηση εγγύτερων γειτόνων (Approximate Nearest Neighbor Search - ANNS), δηλαδή για την αποτελεσματική ανάκτηση ενός συνόλου L από k υποψηφίους για μεγιστοποίηση του recall@k=|G∩L|k, όπου G είναι το σύνολο αληθείας βάσης των των k πλησιέστερων γειτόνων στο P.

Προσεγγιστική Αναζήτηση Εγγύτερων Γειτόνων με Φίλτρα - Filtered ANNS

Σε μια επέκταση της πληροφορίας που φέρει κάθε σημείο δεδομένων (γνωστός και ως διάνυσμα) x ∈ P, έχουμε και ένα συσχετισμένο σύνολο ετικετών Fx⊆F, όπου το F είναι ένα πεπερασμένο σύνολο ετικετών. Ένα ερώτημα στο ευρετήριο τώρα αποτελείται από το διάνυσμα x_q, τον αριθμό στόχο των πλησιέστερων γειτόνων k και ένα φίλτρο ετικέτας f∈F. Το ευρετήριο ANNS χρειάζεται για να βρεθούν οι πλησιέστεροι γείτονες του x_q από το σύνολο $P_f = \{x \in P : f \in F_x\}$, δηλαδή τα σημεία του ευρετηρίου που έχουν την ετικέτα f που σχετίζεται με αυτά. Το ευρετήριο θα πρέπει να μεγιστοποιεί το recall@k, αλλά σε σχέση με το ground truth που υπολογίζεται σε σχέση με το σύνολο P_f, αντί για το P. Ορίζουμε επίσης την ειδικότητα (specificity) της f να είναι |Pf|/|P|: το κλάσμα των σημείων στο ευρετήριοπου έχουν την ετικέτα f που σχετίζεται με αυτά.

Filtered Vamana

Τα ευρετήρια ANNS που βασίζονται σε γράφους κατασκευάζονται έτσι ώστε η GreedySearch να συγκλίνει γρήγορα στους εγγύτερους γείτονες ενός διανύσματος ερωτήματος x_q. Αρχικά περιγράφεται μια φυσική προσαρμογή της GreedySearchς για φιλτραρισμένα ερωτήματα (FilteredGreedySearch) και

μια διαδικασία κατασκευής ευρετηρίου (FilteredVamana) που επιτρέπει στην αναζήτηση να συγκλίνει στη σωστή απάντηση με σχετικά λίγες συγκρίσεις απόστασης.

Algorithm 1: FilteredGreedySearch(S, x_q, k, L, F_q)

Data: Graph G with initial nodes S, query vector x_q, search list size $\mathcal{L}$, and query filter(s) F_q.
Result: Result set $\mathcal{L}$ containing k approximate nearest neighbors, and a set $\mathcal{V}$ containing all visited nodes.
begin
1 Initialize sets $\mathcal{L} \leftarrow \emptyset$ and $\mathcal{V} \leftarrow \emptyset$.
for $s \in S$ **do**
if $F_s \cap F_x \neq \emptyset$ **then**
$\mathcal{L} \leftarrow \mathcal{L} \cup \{s\}$
while $\mathcal{L} \setminus \mathcal{V} \neq \emptyset$ **do**
2 Let $p^* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} \|x_p - x_q\|$
3 $\mathcal{V} \leftarrow \mathcal{V} \cup \{p^*\}$
4 Let $N'_{\text{out}}(p^*) \leftarrow \{p' \in N_{\text{out}}(p^*) : F_{p'} \cap F_q \neq \emptyset, p' \notin \mathcal{V}\}$
5 $\mathcal{L} \leftarrow \mathcal{L} \cup N'_{\text{out}}(p^*)$
if $|\mathcal{L}| > L$ **then**
6 Update $\mathcal{L}$ with the closest L nodes to x_q.
return [k NNs from $\mathcal{L}$; $\mathcal{V}$]

Με δεδομένο ένα ερώτημα x_q και ένα σύνολο ετικετών F_q, θέλουμε να βρούμε τους k κατά προσέγγιση πλησιέστερους γείτονες του x_q, όπου κάθε σημείο της εξόδου μοιράζεται τουλάχιστον μία ετικέτα με το F_q. Η διαδικασία αναζήτησης περιλαμβάνει επίσης ένα σύνολο S κόμβων έναρξης. Για ένα ερώτημα με σύνολο ετικετών F_q, το σύνολο S είναι τυπικά {s(f):f∈Fq}, όπου s(f) είναι ο καθορισμένος κόμβος έναρξης για την ετικέτα f ∈ F που υπολογίζεται κατά την κατασκευή του ευρετηρίου.
Ο αλγόριθμος διατηρεί μια ουρά προτεραιότητας $\mathcal{L}$ μεγέθους το πολύ L (όπου k ≤ L). Σε κάθε επανάληψη, αναζητά τον πλησιέστερο γείτονα p* που δεν έχουμε επισκεφτεί του xq στο $\mathcal{L}$. Στη συνέχεια προσθέτει το p* στο σύνολο κόμβων $\mathcal{V}$. Στη συνέχεια, προσθέτουμε μόνο εκείνους τους εξερχόμενους γείτονες του p* που έχουν τουλάχιστον μία ετικέτα στο Fq στη λίστα $\mathcal{L}$. Τέλος, εάν $|\mathcal{L}| > L$, περικόπτουμε το L για να περιέχει τα $\mathcal{L}$ που βρίσκονται πλησιέστερα στο x_q. Η αναζήτηση τερματίζεται όταν έχουν επισκεφθεί όλους τους κόμβους στο $\mathcal{L}$. Η έξοδος αποτελείται από τους k πλησιέστερους

γείτονες του x από το $\mathcal{L}$, καθώς και από το σύνολο των κόμβων $\mathcal{V}$ που επισκέφθηκαν το οποίο είναι χρήσιμο για την κατασκευή ευρετηρίου (αλλά όχι σε ερωτήματα χρήστη).

Κατασκευή ευρετηρίου

Επιλογή σημείου έναρξης. Απαιτούμε τους κόμβους έναρξης για κάθε φίλτρο να πληρούν δύο κριτήρια: (α) το σημείο έναρξης s για ένα ερώτημα με ένα μόνο φίλτρο f πρέπει να συσχετίζεται με αυτό το φίλτρο, δηλ. f ∈ Fs και (β) κανένα σημείο στο P δεν πρέπει να είναι το σημείο εκκίνησης για πάρα πολλές ετικέτες φίλτρων. Το φορτίο των ερωτημάτων δρομολόγησης με διαφορετικά φίλτρα θα πρέπει να μοιράζεται σε πολλά σημεία, έτσι ώστε να μπορούμε να δημιουργήσουμε ένα γράφημα μικρού οριοθετημένου βαθμού που να καλύπτει όλες τις ετικέτες φίλτρων. Πράγματι, εάν ένα μόνο σημείο χρησίμευε ως το σημείο εκκίνησης πολλών ετικετών, μπορεί να υπάρχουν πολύ λίγες γειτονικές κορυφές με μια συγκεκριμένη ετικέτα από το σημείο εκκίνησης, οδηγώντας σε κακή αναζήτηση. Αυτό το επιτυγχάνουμε χρησιμοποιώντας έναν απλό τυχαιοποιημένο αλγόριθμο εξισορρόπησης φορτίου που περιγράφεται στον αλγόριθμο εύρεσης medoid.

Algorithm 2: FindMedoid(P, τ)

Data: Dataset P with associated filters for all the points, threshold τ.
Result: Map M mapping filters to start nodes.
begin
1 Initialize M be an empty map, and T to an zero map; *//T is intended as a counter*
foreach $f \in \mathcal{F}$, *the set of all filters* **do**
2 Let P_f denote the ids of all points matching filter f
3 Let $R_f \leftarrow \tau$ randomly sampled data point ids from P_f
4 $p^* \leftarrow \arg\min_{p \in R_f} T[p]$
5 update $M[f] \leftarrow p^*$ and $T[p^*] \leftarrow T[p^*] + 1$
return M

Προσθετική κατασκευή γράφου. Η κατασκευή γράφου FilteredVamana είναι ένας προσθετικός αλγόριθμος. Αρχικά προσδιορίζουμε τον κόμβο έναρξης s(·) για κάθε ετικέτα φίλτρου και αρχικοποιούμε το G σε ένα κενό γράφημα. Στη συνέχεια, για κάθε σημείο δεδομένων p ∈ P, με συσχετισμένα φίλτρα/ετικέτες f ∈ Fp, εκτελούμε το FilteredGreedySearch(S_{F_p}, x_p, L, L, F_p), με σημεία εκκίνησης $S_{F_p} = \{s(f) : f \in F_p\}$. Αυτό επιστρέφει ένα σύνολο $\mathcal{V}_{F_p}$ κορυφών που έχουν επισκεφτεί στην

εξερεύνηση αναζήτησης. Όλοι οι κόμβοι που έχει επισκεφτεί ο αλγόριθμος έχουν κάποια ετικέτα $f \in F_p$
.

Algorithm 3: FilteredRobustPrune($p, \mathcal{V}, \alpha, R$)

Data: Graph G, point $p \in P$, candidate set $\mathcal{V}$, distance threshold $\alpha \geq 1$, max outdegree bound R.
Result: G is modified by setting at most R out-neighbors for p.
begin

1 $\mathcal{V} \leftarrow \mathcal{V} \cup N_{\text{out}}(p) \setminus \{p\}$
2 $N_{\text{out}}(p) \leftarrow \emptyset$
while $\mathcal{V} \neq \emptyset$ **do**
3 $p^* \leftarrow \arg\min_{p' \in \mathcal{V}} d(p, p')$
4 $N_{\text{out}}(p) \leftarrow N_{\text{out}}(p) \cup \{p^*\}$
if $|N_{out}(p)| = R$ **then**
5 **break**
for $p' \in \mathcal{V}$ **do**
if $F_{p'} \cap F_p \not\subset F_{p^*}$ **then**
6 **continue**
if $\alpha \cdot d(p^*, p') \leq d(p, p')$ **then**
7 Remove p' from $\mathcal{V}$.

Στη συνέχεια, "κλαδεύουμε" το υποψήφιο σύνολο $\mathcal{V}_{F_p}$ με μια κλήση στη διαδικασία κλαδέματος με γνώση φίλτρου FiteredRobustPrune με παραμέτρους (x,V,α,R). Αυτό διασφαλίζει ότι ο κόμβος γράφου που αντιστοιχεί στο x έχει το πολύ R εξερχόμενουςγείτονες, ενώ επίσης εξαλείφονται οι περιττές ακμές σε κοντινά σημεία. Η διαδικασία κλαδέματος βασίζεται στην ακόλουθη αρχή:

Για οποιαδήποτε τριάδα κορυφών a, b, c και σταθεράς α ≥ 1, η κατευθυνόμενη ακμή (a, c) μπορεί να κλαδευτεί (να απομακρυνθεί από το γράφο) εάν

- υπάρχει η ακμή (a, b)
- το σημείο x_b είναι σημαντικά πιο κοντά στο x_c από ότι το x_a στο x_c, δηλαδή, $|x_b - x_c| \leq (1/\alpha)|x_a - x_c|$, και

- Το F_b περιέχει όλες τις κοινές ετικέτες φίλτρων F_a και F_c, δηλαδή $F_a \cap F_c \subseteq F_b$.

Τέλος, ακμές εισερχόμενες από το y στο x για όλα τα $y \in N_{out}(x)$ και ελέγχουμε ξανά εάν ο βαθμός οποιουδήποτε τέτοιου y υπερβαίνει το R, οπότε και εκτελούμε τη διαδικασία FilteredRobustPrune στο y.

Algorithm 4: FilteredVamana Indexing Algorithm

Data: Database P with n points where i-th point has coords x_i, parameters α, L, R.
Result: Directed graph G over P with out-degree $\leq R$.
begin

Initialize G to an empty graph
Let s denote the medoid of P
Let $\mathrm{st}(f)$ denote the start node for filter label f for every $f \in F$
Let σ be a random permutation of $[n]$
Let F_x be the label-set for every $x \in P$
foreach $i \in [n]$ **do**
  Let $S_{F_{x_{\sigma(i)}}} = \{\mathrm{st}(f) : f \in F_{x_{\sigma(i)}}\}$
  Let $[\emptyset; \mathcal{V}_{F_{x_{\sigma(i)}}}] \leftarrow$ `FilteredGreedySearch`$(S_{F_{x_{\sigma(i)}}}, x_{\sigma(i)}, 0, L, F_{x_{\sigma(i)}})$
  $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}_{F_{x_{\sigma(i)}}}$
  Run `FilteredRobustPrune`$(\sigma(i), \mathcal{V}_{F_{x_{\sigma(i)}}}, \alpha, R)$ to update out-neighbors of $\sigma(i)$.
  foreach $j \in N_{\mathrm{out}}(\sigma(i))$ **do**
    Update $N_{\mathrm{out}}(j) \leftarrow N_{\mathrm{out}}(j) \cup \{\sigma(i)\}$
    if $|N_{\mathrm{out}}(j)| > R$ **then**
      Run `FilteredRobustPrune`$(j, N_{\mathrm{out}}(j), \alpha, R)$ to update out-neighbors of j.

Algorithm 5: StitchedVamana Indexing Algorithm

Data: Database P with n points where i-th point has coords x_i,
Database F of labels, parameters $\alpha, L_{\text{small}}, R_{\text{small}}, R_{\text{stitched}}$.
Result: Directed graph G over P with out-degree $\leq R_{\text{stitched}}$.
begin

Initialize $G = (V, E)$ to an empty graph
Let $F_x \subseteq F$ be the label-set for every $x \in P$
Let $P_f \subseteq P$ be the set of points with label $f \in F$.
foreach $f \in F$ **do**
    Let $G_f \leftarrow \text{Vamana}(P_f, \alpha, R_{\text{small}}, L_{\text{small}})$
foreach $v \in V$ **do**
    FilteredRobustPrune $(v, N_{\text{out}}(v), \alpha, R_{\text{stitched}})$

Stitched Vamana

Ένας εναλλακτικός αλγόριθμος για τη δημιουργία ευρετηρίου μπορεί να χρησιμοποιηθεί μόνο όταν το σύνολο σημείων είναι γνωστό εκ των προτέρων ονομάζεται Stitched Vamana. Για κάθε f $\in$ F, κατασκευάζεται ένα ευρετήριο-γράφος πάνω από τα σημεία P_f με ετικέτα f χρησιμοποιώντας τον αλγόριθμο Vamana με παραμέτρους L_{small} και R_{small}. Αυτές οι παράμετροι είναι μικρότερες από αυτές στον FilteredVamana για την ταχύτερη κατασκευή τουο ευρετηρίου. Στη συνέχεια, δεδομένου ότι οι κορυφές μπορεί δυνητικά να ανήκουν σε πολλαπλούς δείκτες G_f (καθώς ένα σημείο p $\in$ P μπορεί να ανήκει σε πολλαπλά P_f), «ράβουμε» τα γραφήματα G_f μαζί στο γράφημα G, του οποίου οι ακμές είναι η ένωση των συνόλων των ακμών του καθενός G_f. Ο G θα μπορούσε να είναι μεγάλου βαθμού. Μειώνουμε τον μέγιστο εξερχόμενο βαθμό του σε R_{stitched} χρησιμοποιώντας τη διαδικασία FilteredRobustPrune. Ο γράφος που προκύπτει είναι συμβατός με τη διαδικασία FilteredGreedySearch.

Παράδοση εργασίας

Η εργασία είναι ομαδική, **2 ή 3 ατόμων**.
Γλώσσα υλοποίησης: C / C++
Περιβάλλον υλοποίησης: Linux (gcc >= 9.4+).

Παραδοτέα: Η παράδοση της εργασίας θα γίνει με βάση το τελευταιο commit πριν την προθεσμία υποβολής στο git repository σας. **Η χρήση git είναι υποχρεωτική**.
Στο αρχείο README.md θα αναφέρονται τα εξής:

- Ονοματεπώνυμα και ΑΜ των μελών της ομάδας
- Αναφορά στο ποιο μέλος της ομάδας ασχολήθηκε με ποιο αντικείμενο

Επιπλέον, εκτός από τον πηγαίο κώδικα, θα παραδώσετε μια σύντομη αναφορά, με τις σχεδιαστικές σας επιλογές καθώς και να εφαρμόσετε ελέγχους ως προς την ορθότητα του λογισμικού με τη χρήση ανάλογων βιβλιοθηκών (Software testing). Η ορθότητα τυχόν μεταβολών θα ελέγχεται με αυτοματοποιημένο τρόπο σε κάθε commit/push μέσω github actions.

Προδιαγραφές κώδικα: σύμφωνα με τον προγραμματιστικό διαγωνισμό SIGMOD 2024
https://transactional.blog/sigmod-contest/2024 (περιέχει reference implementation, datasets)

Αναφορές

1. Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In Proceedings of the ACM Web Conference 2023 (WWW '23). Association for Computing Machinery, New York, NY, USA, 3406–3416. https://doi.org/10.1145/3543507.3583552