2η Προγραμματιστική Εργασία

Κ23γ: Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα

Χειμερινό εξάμηνο 2025-26

Αναζήτηση διανυσμάτων με Νευρωνικό LSH (Neural LSH) σε Python

Η άσκηση θα υλοποιηθεί σε σύστημα Linux με χρήση **Python 3.10+** και των βιβλιοθηκών **PyTorch**, **NumPy**, **kahip** και των παραδοτέων της 1ης εργασίας. Θα υποβληθεί στις Εργασίες του e-class το αργότερο την **Παρασκευή 5/12 στις 23.59**.

Περιγραφή της εργασίας

Σε αυτή την εργασία θα υλοποιήσετε τον αλγόριθμο **Neural LSH**, ο οποίος μαθαίνει μια βελτιστοποιημένη διαμέριση του χώρου για προσεγγιστική αναζήτηση. Η υλοποίηση θα διαχωριστεί σε δύο κύρια προγράμματα: ένα για την **κατασκευή** του ευρετηρίου (`nlsh_build.py`) και ένα για την **αναζήτηση** (`nlsh_search.py`).

Ο στόχος είναι να υλοποιηθεί το πλαίσιο που περιγράφεται στις διαφάνειες:

1. **Κατασκευή Γράφου k-NN** (Βήμα 1): Κατασκευή του γράφου k-πλησιέστερων γειτόνων $G = (P, E)$ του συνόλου δεδομένων P.
2. **Προετοιμασία Γράφου**: Ο γράφος θα μετατραπεί σε μη κατευθυνόμενο και θα του ανατεθούν βάρη (π.χ., 2 για αμοιβαίους γείτονες, 1 για μονόπλευρους).
3. **Ισοκατανεμημένη Διαμέριση (KaHIP)** (Βήμα 2): Χρήση της βιβλιοθήκης `kahip` για τον υπολογισμό μιας ισοκατανεμημένης διαμέρισης του G σε m μέρη. Αυτό παράγει τις ετικέτες $\pi(p) \in \{0, \ldots, m-1\}$ για κάθε $p \in P$.
4. **Εκπαίδευση Ταξινομητή (PyTorch)** (Βήμα 3): Εκπαίδευση ενός ταξινομητή $M : \mathbb{R}^d \to \mathbb{R}^m$ (π.χ., MLP) που να προβλέπει την ετικέτα $\pi(p)$ από το διάνυσμα p.

Το πρόγραμμα `nlsh_build.py` θα εκτελέσει τα παραπάνω βήματα και θα αποθηκεύσει σε αρχεία το εκπαιδευμένο μοντέλο M (π.χ., `model.pth`) και τη δομή του ευρετηρίου (π.χ., ένα "inverted file" που αντιστοιχίζει κάθε ετικέτα $r \in \{0, \ldots, m-1\}$ στα σημεία p_i που ανήκουν σε αυτήν: $\{r \to \{p_i : \pi(p_i) = r\}\}$).

Το πρόγραμμα `nlsh_search.py` θα φορτώνει το αποθηκευμένο ευρετήριο και, για κάθε ερώτημα q, θα εκτελεί τα εξής:

1. **Πρόβλεψη**: Θα υπολογίζει την έξοδο του μοντέλου $M(q) = \text{softmax}(f_\theta(q))$.
2. **Επιλογή Μερών (Multi-Probe)**: Θα εντοπίζει τα T μέρη (bins) με την υψηλότερη πιθανότητα.
3. **Συλλογή Υποψηφίων**: Θα συλλέγει όλα τα σημεία p_i που ανήκουν σε αυτά τα T υποσύνολα (χρησιμοποιώντας το "inverted file").
4. **Ακριβής Αναζήτηση**: Θα εκτελεί εξαντλητική αναζήτηση (με Ευκλείδεια μετρική) μόνο στο σύνολο των υποψηφίων για να βρει τους N πλησιέστερους γείτονες.

Μετά την υλοποίηση, θα εκτελεστούν πειράματα για τη βέλτιστη επιλογή των υπερπαραμέτρων (αριθμός στρωμάτων, αριθμός κόμβων ανά στρώμα, μέγεθος δέσμης [batch], αριθμός περιόδων εκπαίδευσης [epochs], αριθμός μερών διαμέρισης [m], παράμετρος k για την κατασκευή του γράφου των πλησιέστερων γειτόνων, κ.λπ.). Στη συνέχεια, θα πραγματοποιηθεί **πειραματική συγκριτική μελέτη** της απόδοσης του Neural LSH -με τη βέλτιστη επιλογή υπερπαραμέτρων- σε σχέση με τις μεθόδους της 1ης εργασίας (LSH, Hypercube, IVFFlat, IVFPQ).

ΕΙΣΟΔΟΣ

1. Σύνολο Δεδομένων & Αναζήτησης

Θα χρησιμοποιηθούν τα ίδια σύνολα δεδομένων (MNIST, SIFT) και αναζήτησης (query sets) με την 1η εργασία. Θα πρέπει να υλοποιήσετε σε Python τον κώδικα ανάγνωσης (parser) για τα δυαδικά αρχεία.

2. Παράμετροι Αλγορίθμων

Α. `nlsh_build.py` (Κατασκευή Ευρετηρίου)

Το πρόγραμμα αυτό δέχεται ως είσοδο το σύνολο δεδομένων και παράγει τα αρχεία του ευρετηρίου.

- `-d <input file>`: (Υποχρεωτικό) Το αρχείο του συνόλου δεδομένων (π.χ., `input.dat`).
- `-i <index path>`: (Υποχρεωτικό) Το path (π.χ., `nlsh_index.bin`) όπου θα αποθηκευτούν τα αρχεία του ευρετηρίου (μοντέλο και inverted file).
- `-type <sift|mnist>`: (Υποχρεωτικό) Ο τύπος των δεδομένων.
- `--knn <int>`: Ο αριθμός γειτόνων k για την κατασκευή του k-NN γράφου (Default: 10).
- `-m <int>`: Ο αριθμός των μερών (blocks/parts) m για το KaHIP (Default: 100).
- `--imbalance <float>`: Το ποσοστό ανισορροπίας ϵ για το KaHIP (Default: 0.03).
- `--kahip_mode <int>`: Η ρύθμιση του KaHIP (0=FAST, 1=ECO, 2=STRONG) (Default: 2).
- `--layers <int>`: Αριθμός στρωμάτων MLP (Default: 3).

- `--nodes <int>`: Αριθμός κόμβων στρώματος (Default: 64).
- `--epochs <int>`: Περίοδοι εκπαίδευσης του MLP (Default: 10).
- `--batch_size <int>`: Μέγεθος δέσμης (Default: 128).
- `--lr <float>`: Ρυθμός εκμάθησης (Default: 0.001).
- `--seed <int>`: Φύτρο ψευδοτυχαιών (Default: 1).

Β. `nlsh_search.py` (Αναζήτηση)

Το πρόγραμμα αυτό δέχεται ως είσοδο το ευρετήριο, τα ερωτήματα και το αρχικό σύνολο δεδομένων (απαραίτητο για τον υπολογισμό των πραγματικών αποστάσεων) και παράγει το αρχείο εξόδου.

- `-d <input file>`: (Υποχρεωτικό) Το αρχείο του συνόλου δεδομένων.
- `-q <query file>`: (Υποχρεωτικό) Το αρχείο ερωτημάτων (π.χ., `query.dat`).
- `-i <index path>`: (Υποχρεωτικό) Το path του ευρετηρίου που δημιουργήθηκε από το `nlsh_build.py`.
- `-o <output file>`: (Υποχρεωτικό) Το αρχείο εξόδου.
- `-type <sift|mnist>`: (Υποχρεωτικό) Ο τύπος των δεδομένων.
- `-N <int>`: Αριθμός πλησιέστερων γειτόνων (Default: 1).
- `-R <double>`: Ακτίνα αναζήτησης (Default: 2000 για MNIST, 2800 για SIFT).
- `-T <int>`: Αριθμός μερών (bins) προς έλεγχο (multi-probe) (Default: 5).
- `-range <true|false>`: Εάν θα εκτελεστεί range search (Default: true).

3. Εκτέλεση

Φάση 1: Κατασκευή Ευρετηρίου

```
python nlsh_build.py -d input.dat -i nlsh_index -type sift --knn 15 -m 100 --epochs 10
```

Φάση 2: Αναζήτηση

```
nlsh_search.py -d input.dat -q query.dat -i nlsh_index -o output.txt -type sift -N 10 -T 5
```

ΕΞΟΔΟΣ

Το αρχείο εξόδου (`-o`) του `nlsh_search.py` θα ακολουθεί **ακριβώς το ίδιο πρότυπο** με αυτό της 1ης εργασίας, ώστε να είναι δυνατή η άμεση σύγκριση των μετρικών.

Πρότυπο Αρχείου Εξόδου:

```
<METHOD NAME> [Neural LSH]
Query: image_id_in_query_set
Nearest neighbor-1: image_id_in_data_set
distanceApproximate: <double>
distanceTrue: <double>
...
Nearest neighbor-N: image_id_in_data_set
distanceApproximate: <double>
distanceTrue: <double>
R-near neighbors:
image_id_A
...
image_id_Z
Average AF: <double>
Recall@N: <double>
QPS: <double>
tApproximateAverage: <double>
tTrueAverage: <double>
```

Σημείωση: Το `tTrueAverage` και το `distanceTrue` απαιτούν την εκτέλεση εξαντλητικής αναζήτησης σε ολόκληρο το σύνολο P για κάθε ερώτημα, όπως και στην 1η εργασία.

Επιπρόσθετες απαιτήσεις

1. **Οργάνωση Κώδικα**: Τα δύο προγράμματα (`nlsh_build.py`, `nlsh_search.py`) πρέπει να είναι καλά οργανωμένα, κάνοντας χρήση ξεχωριστών modules (π.χ., `dataset_parser.py`, `models.py`, `graph_utils.py`).

2. **Βιβλιοθήκες**: Απαιτείται η χρήση **PyTorch** για το MLP, **NumPy** για διανύσματα, η χρήση των παραδοτέων της πρώτης εργασίας για την αποδοτική κατασκευή του k-NN γράφου, και της βιβλιοθήκης **`kahip`** (μέσω `pip install kahip`) για τη διαμέριση.

3. **Μορφή Γράφου (KaHIP)**: Για τη χρήση του `kahip.kaffpa`, ο γράφος πρέπει να μετατραπεί στη μορφή **CSR** (Compressed Sparse Row), όπως περιγράφεται στις διαφάνειες, παρέχοντας τις λίστες `vwgt`, `xadj`, `adjncy`, και `adjcwgt`. Ιδιαίτερη προσοχή πρέπει να δοθεί στη συμμετρικοποίηση και τα βάρη (1 ή 2) του γράφου k-NN.

4. **Αρχείο `readme.md`**: Το `readme.md` πρέπει να περιλαμβάνει: α) τίτλο/περιγραφή, β) κατάλογο αρχείων Python και περιγραφή τους, γ) οδηγίες εγκατάστασης εξαρτήσεων (`pip install -r requirements.txt`), και δ) λεπτομερείς οδηγίες χρήσης/εκτέλεσης και για τα δύο σενάρια (`build` και `search`).

5. **Αναφορά**: Η τεκμηρίωση της πειραματικής συγκριτικής μελέτης (που θα περιλαμβάνει αναλυτική τεκμηρίωση για την επιλογή των υπερπαραμέτρων και θα συγκρίνει **όλες** τις μεθόδους από την 1η και 2η εργασία) αποτελεί ξεχωριστό παραδοτέο (markdown ή PDF).

6. **Git**: Η υλοποίηση θα πρέπει να γίνει με χρήση Git.