PROJECT

Πολυδιάστατες Δομές Δεδομένων

ΔΗΜΗΤΡΗΣ ΚΑΤΣΑΝΟΣ ΑΜ: 1093384

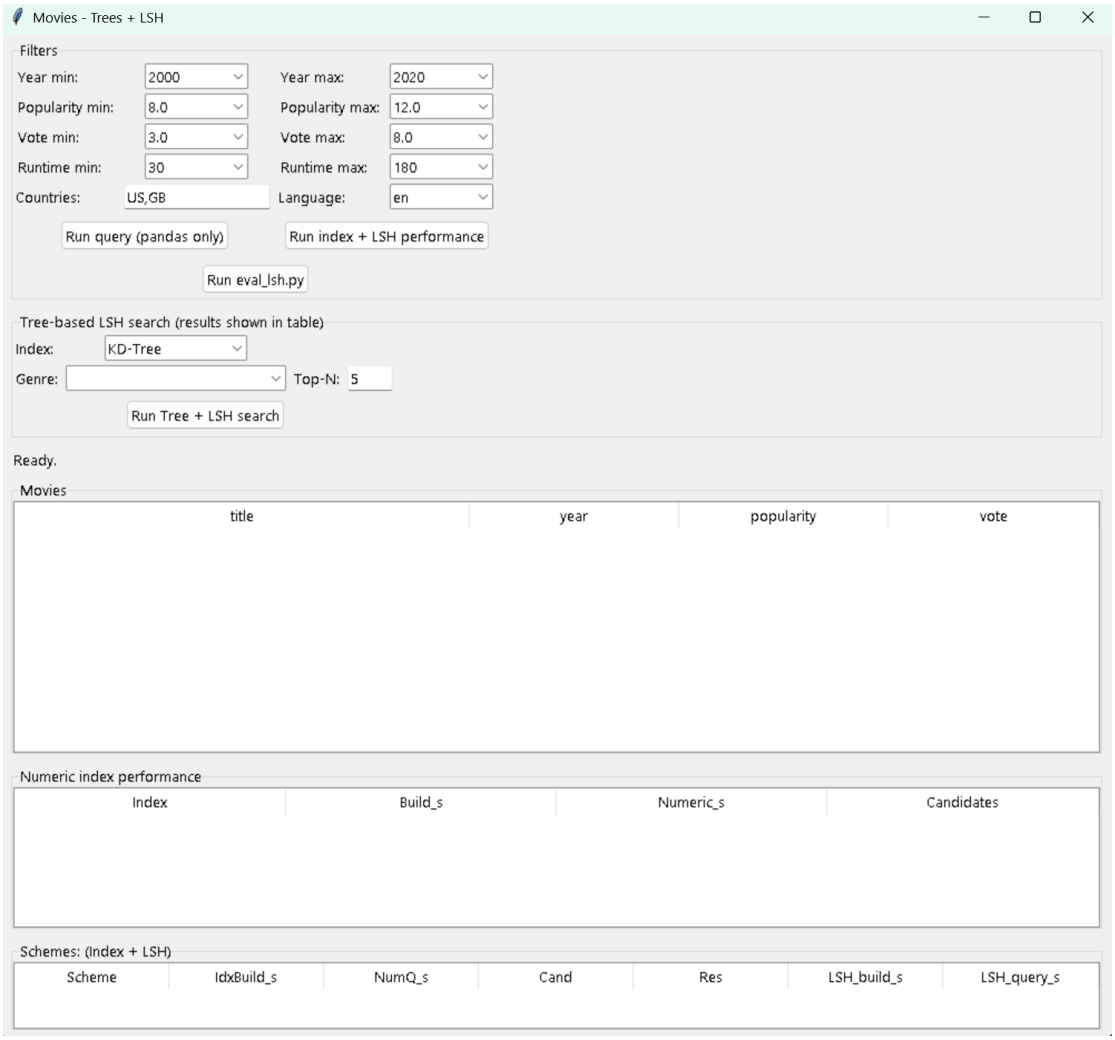
ΑΝΔΡΕΑΣ ΖΑΦΕΙΡΟΠΟΥΛΟΣ ΑΜ: 1093361

ΘΑΛΗΣ-ΧΡΥΣΟΣΤΟΜΟΣ ΤΣΙΩΝΑΣ ΑΜ: 1097467

ΙΩΑΝΝΗΣ ΓΡΗΓΟΡΟΠΟΥΛΟΣ ΑΜ: 1097444

Ασχοληθήκαμε με το Project 1 στο οποίο υλοποιήσαμε ένα σύστημα αποδοτικής αναζήτησης ταινιών, το οποίο υποστηρίζει πολυδιάστατα αριθμητικά ερωτήματα και αναζήτηση ομοιότητας σε κατηγορικά χαρακτηριστικά. Στόχος μας ήταν η μείωση του χρόνου αναζήτησης μέσω κατάλληλων δομών ευρετηρίασης (Indexes) και τεχνικών hashing, σε σύγκριση με απλή σειριακή αναζήτηση, η οποία θα ήταν πολύ αργή λόγο του μεγάλου όγκου δεδομένων. Επίσης δημιουργήσαμε και ένα GUI για την καλύτερη αλληλεπίδραση μας με το Project.

To GUI



Dataset και Προεπεξεργασία

Το Dataset που κατεβάσαμε από το link της αναφοράς περιλαμβάνει έναν μεγάλο όγκο από πληροφορίες για ταινίες, όπως έτος, popularity, μέσο όρο βαθμολογίας, διάρκεια, χώρες παραγωγής, γλώσσες παραγωγής, genres κ.α. Η φόρτωση και προεπεξεργασία των δεδομένων υλοποιείται στο αρχείο main.py μέσω των συναρτήσεων load_dataset() και preprocess_dataset(). Σε αυτό το στάδιο γίνεται καθαρισμός των δεδομένων, μετατροπή τύπων και εξαγωγή δομών που διευκολύνουν τις επόμενες φάσεις (π.χ. μετατροπή genres σε σύνολα). Στην συνέχεια το βασικό dataframe δημιουργείται με την build_base_pool() το οποίο χρησιμοποιείται τόσο από το main.py όσο και από το GUI (gui_tk.py).

Δομές Ευρετηρίασης (Indexes)

Για την επιτάχυνση των numeric range quaries υλοποιήθηκαν οι τέσσερις δομές ευρετηρίασης που μας ζητήθηκαν και από την εκφώνηση. Συγκεκριμένα KD-Tree (kd_tree.py), Quad_tree (quad_tree.py), Range-Tree (range_tree.py) και R-Tree (r_tree.py). Κάθε δομή χρησιμοποιείται για να φιλτράρει γρήγορα τα δεδομένα βάσει αριθμητικών κριτηρίων και να επιστρέψει ένα σύνολο υποψήφιων αποτελεσμάτων (candidates).

Παρότι το ερώτημα είναι πολυδιάστατο (έως 5 αριθμητικές διαστάσεις), κάθε δομή δεδομένων κάνει index σε διαφορετικό αριθμό διαστάσεων. Το KD-Tree υποστηρίζει queries σε περισσότερες διαστάσεις, ενώ το Quad-Tree και το R-Tree χρησιμοποιούνται σε 2 διαστάσεις και το Range-Tree σε 1 διάσταση. Τα υπόλοιπα φίλτρα εφαρμόζονται στη συνέχεια πάνω στα candidates, ώστε το τελικό αποτέλεσμα να παραμένει σωστό και η σύγκριση των σχημάτων Index + LSH να είναι δίκαιη.

Candidates και Φιλτράρισμα

Στο GUI, το φιλτράρισμα εκτελείται αρχικά με pandas, ώστε να παραχθεί το βασικό αποτέλεσμα και να υποστηριχθεί η επιλογή genres. Στην συνέχεια, τα indexes χρησιμοποιούνται για αριθμητικό φιλτράρισμα και παραγωγή candidates, με σκοπό τη σύγκριση της απόδοσης τους.

Αναζήτηση ομοιότητας με LSH

Για την αναζήτηση ομοιότητας βάση genres χρησιμοποιείται LSH, υλοποιημένο στο lsh_text.py. Η μέθοδος βασίζεται σε MinHash signatures και επιτρέπει την αποδοτική εύρεση παρόμοιων ταινιών χωρίς πλήρη σύγκριση όλων των στοιχείων.

Συνδυαστικά Σχήματα και Αξιολόγηση

Το σύστημα αξιολογείται μέσω συνδυαστικών σχημάτων της μορφής Index + LSH. Η πειραματική αξιολόγηση υλοποιείται με την συνάρτηση evaluate_indexes() στο main.py και παρουσιάζει χρόνους build, χρόνους query, πλήθος candidates και τελικά αποτελέσματα.

kNN

Επιπλέον υλοποιήθηκαν kNN queries στο KD-Tree στο αρχείο kd_tree.py. Τα kNN queries αξιολογούνται μέσω του main.py, συγκρίνοντας τα αποτελέσματα και τον χρόνο εκτέλεσης με μια απλή προσέγγιση που υπολογίζει τις αποστάσεις από όλα τα σημεία του dataset (brute-force)

```python
    # kNN
    def knn_query(
        self,
        query_point: Sequence[float],
        k: int = 5,
        return_distances: bool = False,
    ) -> List[int] | List[Tuple[int, float]]:
        """Επιστρέφει τους k κοντινότερους γείτονες του query_point (Euclidean).
AV  return_distances=True επιστρέφει (index, distance), αλλιώς μόνο indices."""

        if self.root is None or self.k == 0 or k <= 0:
            return []

        if len(query_point) != self.k:
            raise ValueError("query_point must have length equal to k dimensions")

        k = min(k, self.n_points)

        # max-heap me ( -dist_sq, index ) gia na kratame toys k kalyterous
        best: List[Tuple[float, int]] = []
        self._knn_search(self.root, query_point, k, best)

        # taksinomisi apo pio kontino se mio makrino
        best_sorted = sorted(((-d2, idx) for d2, idx in best), key=lambda x: x[0])

        if return_distances:
            return [(idx, dist ** 0.5) for dist, idx in best_sorted]
        return [idx for dist, idx in best_sorted]

    def _knn_search(
        self,
        node: Optional[KDNode],
        query_point: Sequence[float],
        k: int,
        best: List[Tuple[float, int]],
    ) -> None:
        "Αναδρομική αναζήτηση kNN με pruning."
        if node is None:
            return
```

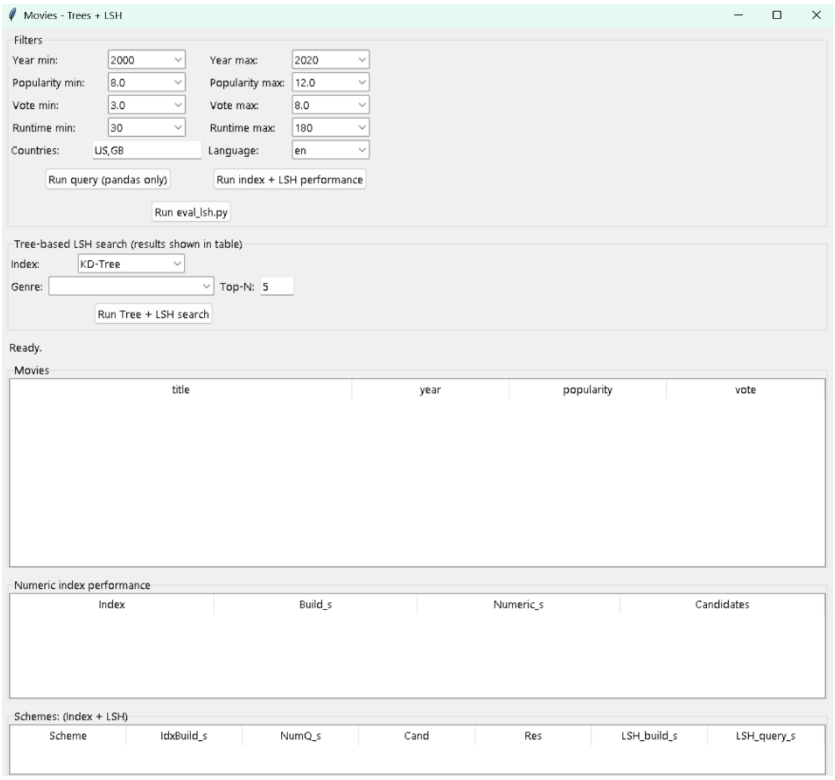
```python
        # apostasi tou node.point apo query
        d2 = 0.0
        p = node.point
        for dim in range(self.k):
            diff = p[dim] - query_point[dim]
            d2 += diff * diff

        # enimerosi heap
        if len(best) < k:
            heapq.heappush(best, (-d2, node.index))
        else:
            # an brikame kalytero apo ton xeirotero, antikatastasi
            worst_neg_d2, _ = best[0]
            if -d2 > worst_neg_d2:
                heapq.heapreplace(best, (-d2, node.index))

        axis = node.axis
        diff_axis = query_point[axis] - p[axis]

        # prwta pame sto pio kontino ypodentro
        near = node.left if diff_axis < 0 else node.right
        far = node.right if diff_axis < 0 else node.left
        self._knn_search(near, query_point, k, best)

        # pruning
        if len(best) < k:
            self._knn_search(far, query_point, k, best)
        else:
            worst_d2 = -best[0][0]
            if diff_axis * diff_axis <= worst_d2:
                self._knn_search(far, query_point, k, best)
```

GUI

Το GUI επιτρέπει την εκτέλεση baseline queries, index-based queries και πειραματικής αξιολόγησης. Χρησιμοποιείται για επίδειξη και σύγκριση απόδοσης των μεθόδων, χωρίς να επηρεάζει την βασική λογική του συστήματος.

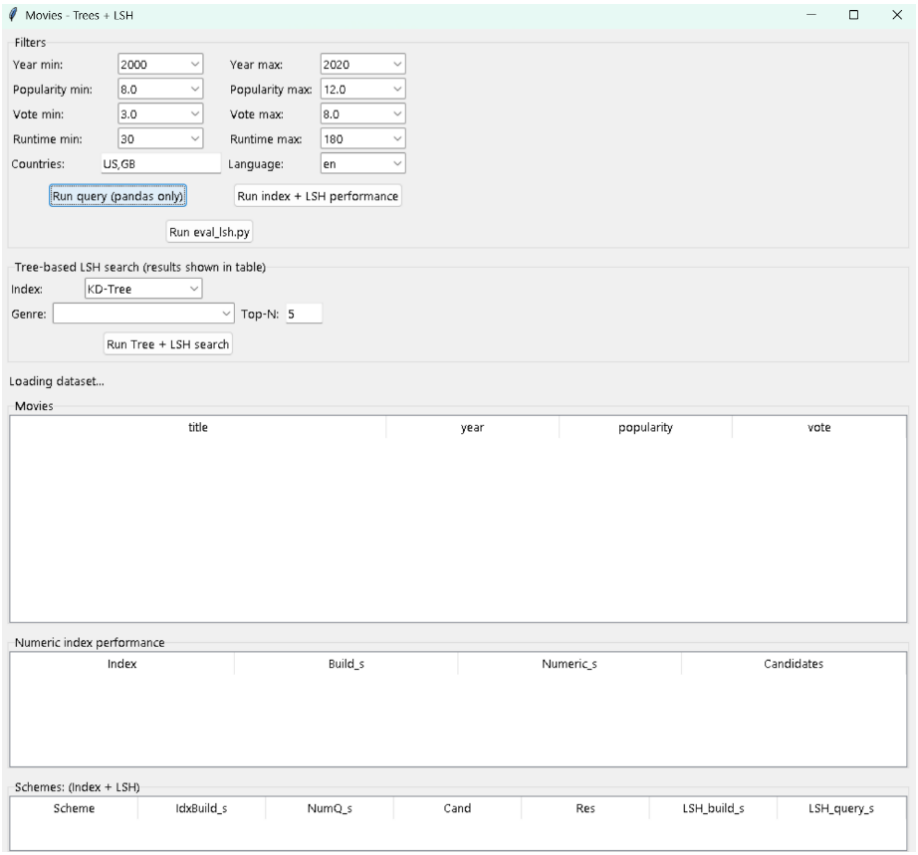
Ακολουθεί από κάτω ένα παράδειγμα με την λειτουργικότητα του GUI:

1.Αρχική Κατάσταση GUI



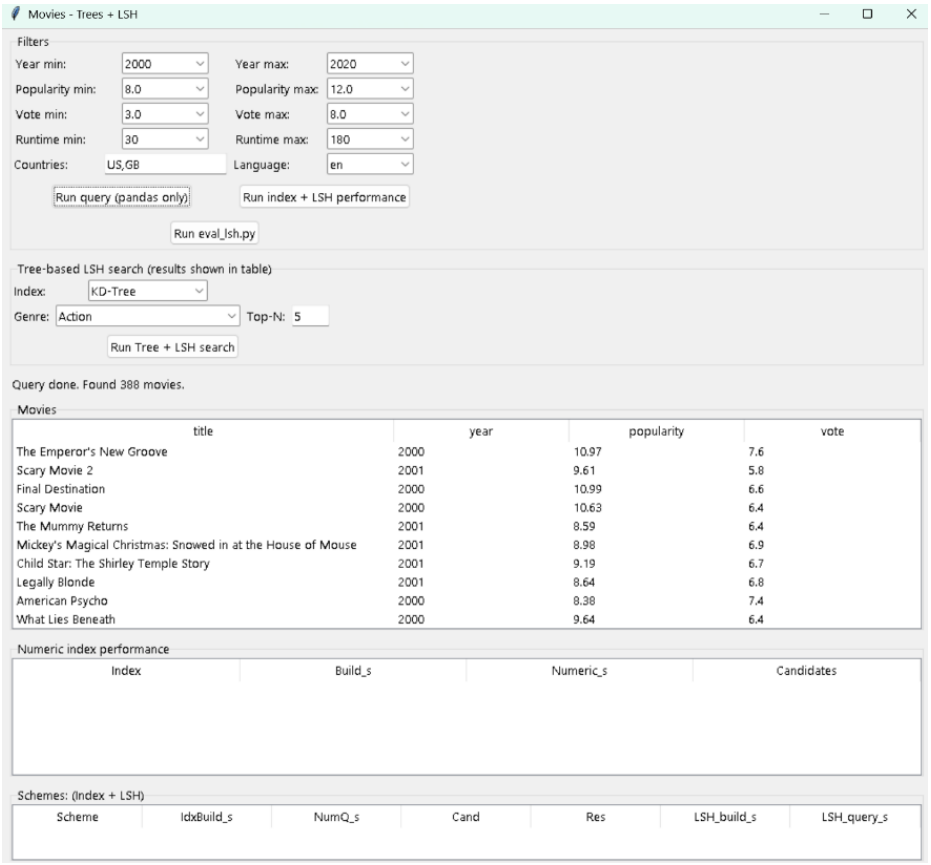
Ορίζουμε τα αριθμητικά φίλτρα του ερωτήματος (έτος, popularity, βαθμολογία, διάρκεια, χώρα και γλώσσα) πριν την εκτέλεση της αναζήτησης

2. Εκτέλεση pandas-only query



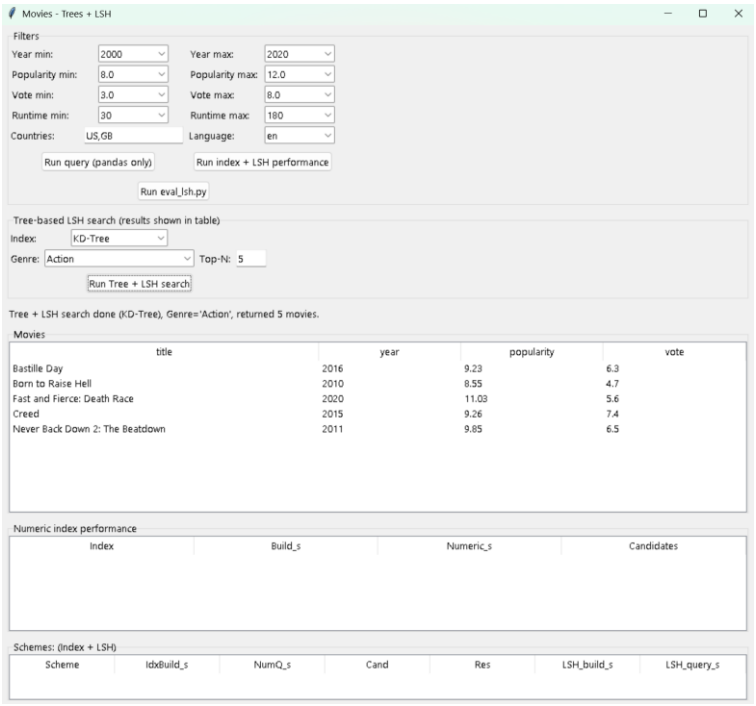
Εκτελούμε αναζήτηση με χρήση pandas, ώστε να εφαρμοστούν όλα τα φίλτρα και να παραχθεί το βασικό αποτέλεσμα (ground truth).

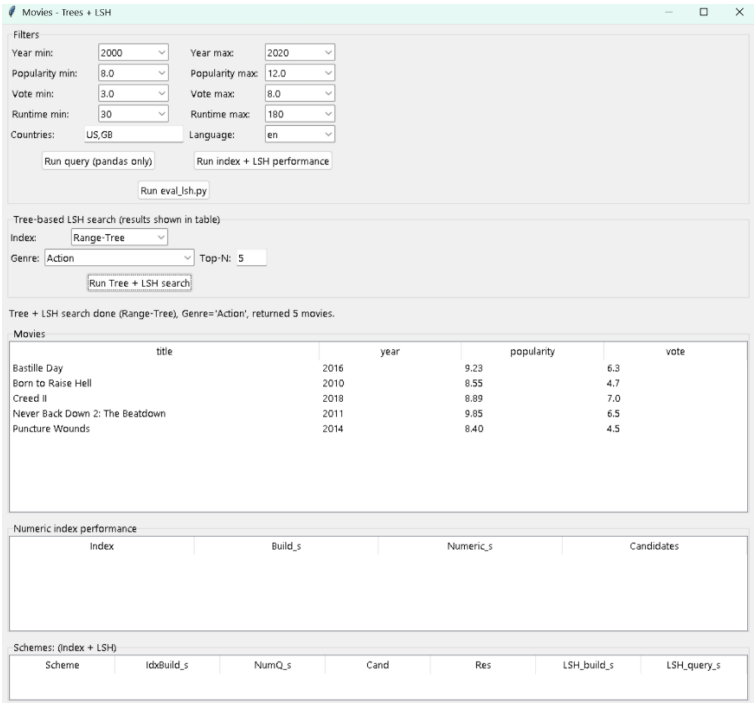
3.Αποτελέσματα pandas-only



Εμφανίζουμε τα αποτελέσματα της baseline αναζήτησης χωρίς χρήση index, τα οποία χρησιμοποιούμε ως σημείο αναφοράς και για την επιλογή genre.

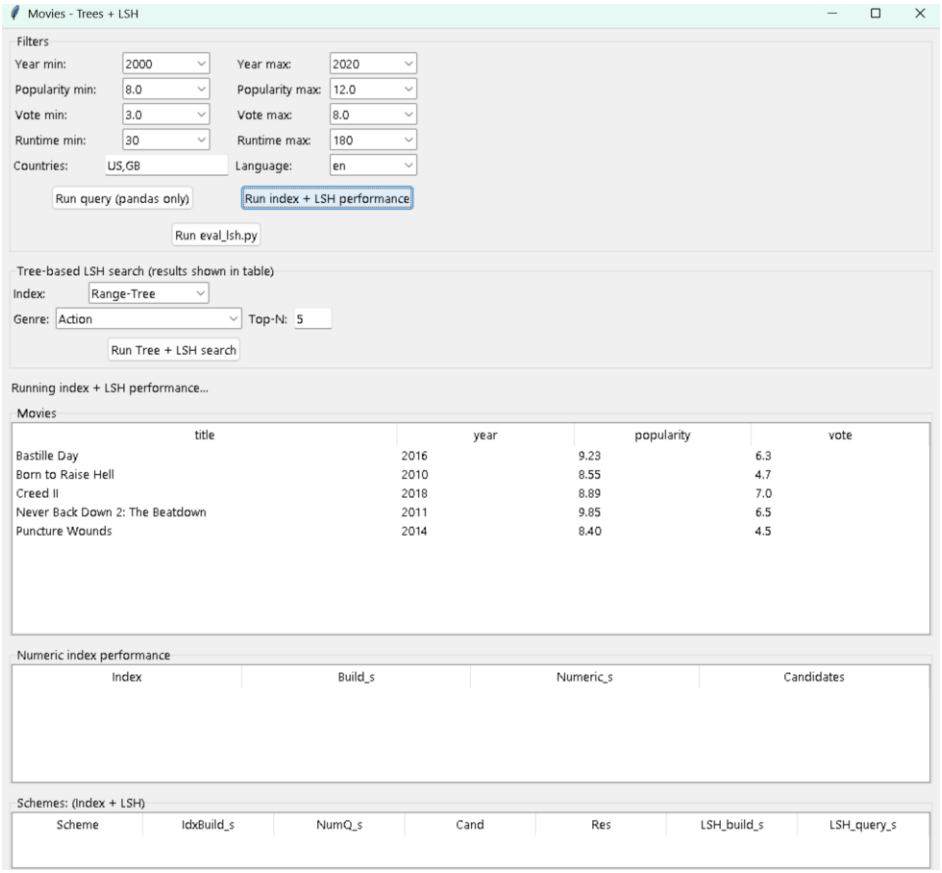
4. Tree + LSH search





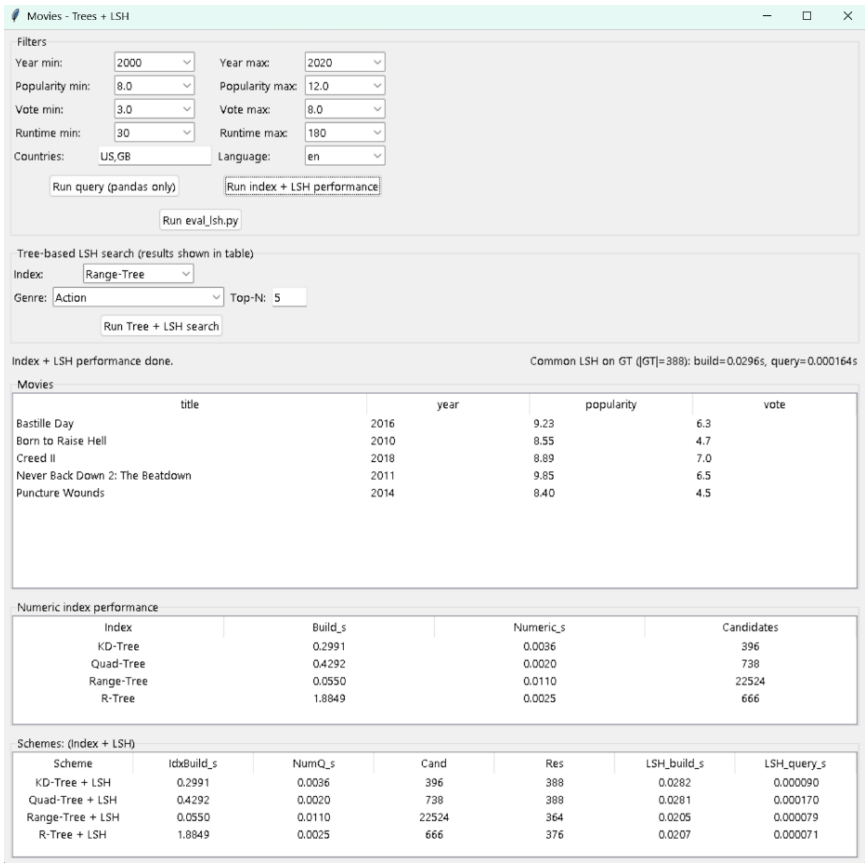
Εκτελούμε αναζήτηση χρησιμοποιώντας διαφορετικά είδη Tree για το αριθμητικό φιλτράρισμα και LSH για αναζήτηση ομοιότητας με βάση το επιλεγμένο genre. Στα δυο αυτά screenshots δείχνουμε ότι με 2 διαφορετικά αλλά στο ίδιο genre, επιστρέφει διαφορετικά αποτελέσματα μεταξύ τους. Κάτι το οποίο είναι λογικό, καθώς κάθε index επιστρέφει διαφορετικό σύνολο υποψήφιων αποτελεσμάτων πάνω στο οποίο εφαρμόζεται το LSH.

5.Εκτέλεση Index + LSH performance



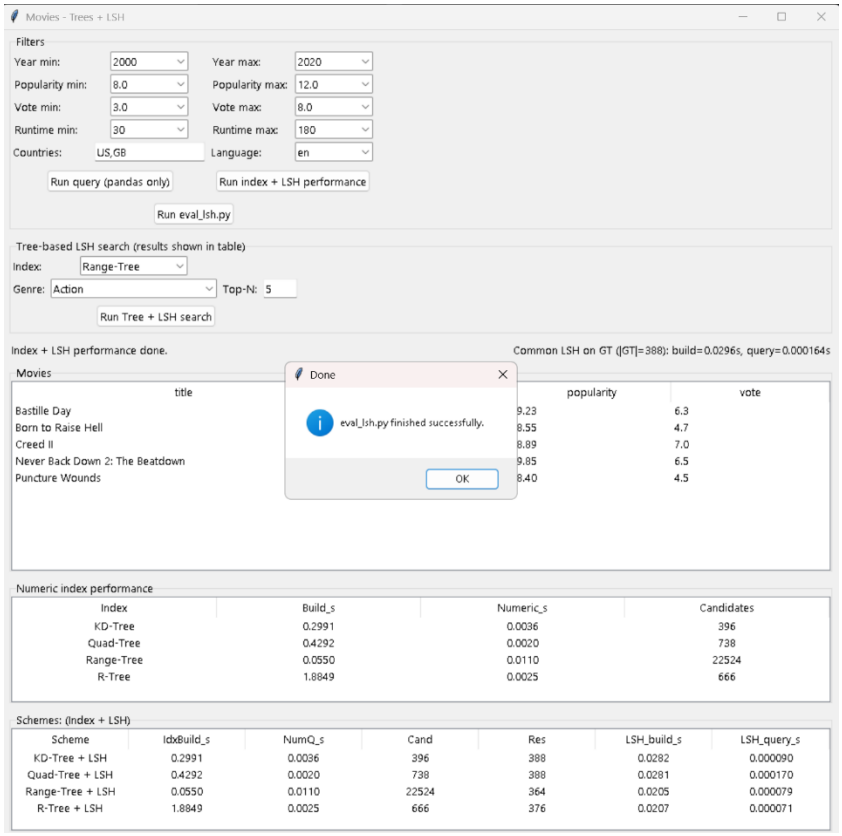
Εκτελούμε την διαδικασία αξιολόγησης της απόδοσης για όλα τα indexes και τα συνδυαστικά σχήματα Index + LSH.

6. Numeric Index Performance Table και Schemes: (Index + LSH) Table



Παρουσιάζουμε συγκριτικά την απόδοση των Indexes και των συνδυαστικών σχημάτων Index + LSH, ως προς τον χρόνο κατασκευής, τον χρόνο εκτέλεσης των αριθμητικών queries, τον αριθμό των υποψήφιων αποτελεσμάτων και τους χρόνους του LSH.

7. Εκτέλεση αξιολόγησης LSH (eval_lsh.py)



Τέλος με το κουμπί ”Run eval_lsh.py” εκτελούμε το συγκεκριμένο αρχείο, το οποίο εκτελεί ανεξάρτητη αξιολόγηση της λειτουργικότητάς του LSH και επιβεβαιώνει την σωστή εκτέλεση του.