kd-tree를 이용한 k-NN 탐색 알고리즘

Input: kd-tree에 저장된 점들 $d_1, d_2, \ldots, d_N$, 질의 점 q, 탐색할 이웃의 개수 k

Output: q의 k-최근접 이웃

$pq \leftarrow \{(\text{root}, d(\text{root}, q))\}$;

while *pq is not empty* **do**

  $(e, \text{dist}) \leftarrow pq.pop()$;

  if *e가 내부 노드이면* **then**

    foreach *자식 c of e* **do**

      Compute $d(c, q)$;

      $pq.push((c, d(c, q)))$;

    end

  else

    Report e as a nearest neighbor;

    if *보고된 이웃의 개수가 k에 도달하면* **then**

      break;

    end

  end

end

관련 질문들 (kd-tree)

- **질문 1:** kd-tree의 구조적 특징과 k-NN 탐색에서 어떻게 활용되는지 설명하세요.
- **질문 2:** 우선순위 큐를 이용하는 방식이 왜 효율적인지, 다른 탐색 방법과 비교하여 논의해보세요.
- **질문 3:** 이 알고리즘에서 내부 노드와 리프 노드를 구분하는 이유는 무엇인가요?
- **질문 4:** k-NN 탐색의 종료 조건과 탐색 중단 기준을 어떻게 설정할 수 있는지 설명해보세요.
- **질문 5:** 데이터의 차원이 증가할 경우 kd-tree 기반 k-NN 탐색에서 발생할 수 있는 문제점은 무엇이며, 이를 해결하기 위한 방법은 무엇인가요?

질문에 대한 답변 (kd-tree)

- **답 1:** kd-tree는 공간을 재귀적으로 분할하는 구조를 가지며, 각 분할 영역은 특정 차원에 대해 정렬된 데이터를 포함합니다. 이 구조 덕분에 질의 점과 가까운 영역만 탐색할 수 있어 k-NN 탐색에서 효율적으로 사용됩니다.
- **답 2:** 우선순위 큐를 사용하면 현재까지 탐색한 노드 중 질의 점과의 거리가 가장 짧은 순서대로 처리할 수 있습니다. 이는 불필요한 영역을 건너뛰고, 가장 가능성이 높은 근접점을 우선적으로 확인할 수 있게 해주어 다른 방법보다 효율적입니다.
- **답 3:** 내부 노드는 추가 자식 노드를 포함하고 있어 탐색의 범위를 축소하는 역할을 합니다. 반면, 리프 노드는 실제 데이터 포인트이므로, 이들을 대상으로 최종 k-NN 후보를 결정하게 됩니다.
- **답 4:** 보통 k개의 근접점을 찾은 후, 남은 노드들의 최소 거리가 현재 k번째 근접점의 거리보다 크다면 탐색을 중단할 수 있습니다. 이 조건은 추가적으로 더 가까운 점이 없음을 보장하는 역할을 합니다.
- **답 5:** 데이터 차원이 증가하면 '차원의 저주'로 인해 kd-tree의 분할 효율이 떨어져 탐색 성능이 저하될 수 있습니다. 이를 완화하기 위해 차원 축소 기법(PCA, t-SNE 등)을 사용하거나, ball tree, cover tree 등의 다른 자료구조를 고려할 수 있습니다.

Cosine Similarity 기반 LSH 개요

- **LSH (Locality Sensitive Hashing):** 유사한 데이터가 동일한 해시 버킷에 들어가도록 설계된 해싱 기법으로, 대용량 데이터셋에서 근사 최근접 이웃 검색을 빠르게 수행할 수 있습니다.
- **Cosine Similarity:** 두 벡터의 내적을 벡터 크기의 곱으로 나누어 계산하며, 벡터 간의 방향(각도) 유사도를 측정합니다.
- **작동 원리:**
  1. 여러 개의 무작위 하이퍼플레인을 생성합니다.
  2. 각 하이퍼플레인을 기준으로 벡터가 어느 쪽에 위치하는지(예: 0 또는 1)로 이진 해시 값을 생성합니다.
  3. 여러 해시 함수를 조합하여 최종 해시 키를 구성하고, 유사한 벡터들은 동일하거나 유사한 해시 키를 가지게 됩니다.
- **장점:** 정확한 계산 대신 근사 결과를 빠르게 도출할 수 있어, 대규모 데이터셋에서 효율적으로 유사 데이터를 검색할 수 있습니다.

Cosine Similarity 기반 LSH 관련 질문

- **질문 1:** Cosine similarity의 정의와, 두 벡터 간 유사도를 어떻게 측정하는지 설명하세요.
- **질문 2:** LSH에서 무작위 하이퍼플레인을 사용하는 이유와 그 역할에 대해 논의하세요.
- **질문 3:** Cosine similarity 기반 LSH에서 해시 충돌이 발생하는 이유와, 이를 줄이기 위한 방법에는 어떤 것들이 있는지 설명하세요.
- **질문 4:** 대용량 텍스트 데이터셋에서 LSH를 활용한 근사 최근접 이웃 검색의 장단점을 비교해보세요.

Cosine Similarity 기반 LSH 질문에 대한 답변

- **답 1:** Cosine similarity는 두 벡터의 방향성을 비교하는 지표로, 두 벡터의 내적을 각 벡터의 크기의 곱으로 나눈 값으로 계산됩니다.
- **답 2:** LSH에서 무작위 하이퍼플레인은 벡터 공간을 여러 부분으로 균등하게 분할하는 역할을 합니다. 이를 통해 유사한 벡터들이 같은 해시 버킷에 들어갈 확률이 높아지며, 전체 데이터셋을 순회하지 않고도 근사적으로 유사한 데이터를 빠르게 찾을 수 있습니다.
- **답 3:** Cosine similarity 기반 LSH에서는 벡터들이 비슷한 방향을 가질 경우 동일한 해시 값을 가질 가능성이 있어 해시 충돌이 발생할 수 있습니다. 이를 줄이기 위해 여러 개의 해시 함수를 조합하여 다중 해시 테이블을 구성하거나, 해시 함수의 수를 늘리는 방법이 사용됩니다.
- **답 4:** LSH를 활용한 근사 최근접 이웃 검색은 대용량 텍스트 데이터셋에서 빠른 검색과 낮은 계산 비용이라는 장점을 가집니다. 그러나 근사적인 결과를 제공하기 때문에 정확도가 떨어질 수 있으며, 최적의 성능을 위해 해시 함수의 수, 테이블의 수 등 파라미터 튜닝이 요구된다는 단점이 있습니다.