**Nombre del Algoritmo:** (1+ε)-approximate nearest neighbor search

**Descripción del Problema Computacional para la cual el algoritmo es una solución.**

**Delimitación del Problema:**

El algoritmo (1+ε)-approximate nearest neighbor search (ANN) busca resolver el problema de encontrar el punto más cercano (o aproximadamente cercano) a una consulta en un espacio de alta Dimensionalidad, con un margen de error permitido 𝜖. Este problema surge en situaciones donde la búsqueda exacta es computacionalmente costosa debido al tamaño del conjunto de datos o la Dimensionalidad del espacio.

**Comprensión del Problema:**

El problema consiste en:

1. Búsqueda de Vecindad: Para un punto de consulta 𝑞, encontrar un punto 𝑝 del conjunto 𝑆 tal que la distancia 𝑑 (𝑝, 𝑞) esté dentro de un factor de (1+𝜖) de la distancia al vecino exacto más cercano.
2. Aproximación Controlada: Ofrecer un balance entre precisión y eficiencia.

**Clasificación del Problema:**

El problema pertenece al área de:

* Búsqueda Aproximada en estructuras de datos.
* Optimización Subóptima en algoritmos geométricos.
* Reducción de Complejidad para espacios de alta Dimensionalidad.

**Viabilidad Computacional:**

La viabilidad se garantiza al aceptar una aproximación (𝜖>0), lo que permite reducir el tiempo de cómputo en comparación con algoritmos exactos. Se utilizan estructuras como Locality-Sensitive Hashing (LSH) o árboles KD optimizados para aproximaciones.

**Contexto del Problema:**

El algoritmo es útil en aplicaciones donde:

* La dimensionalidad del espacio es alta (problema de la "maldición de la dimensionalidad").
* Se requiere alta velocidad en búsquedas con tolerancia a errores, como en:
  + Recuperación de información.
  + Reconocimiento de patrones.
  + Procesamiento de señales.

**Análisis de Datos y Estructura del Problema:**

* Entrada: Un conjunto 𝑆 de puntos en un espacio métrico (𝑋, 𝑑).}
* Consulta: Un punto 𝑞 en el mismo espacio.
* Salida: Un punto 𝑝 ∈ 𝑆 tal que 𝑑(𝑝, 𝑞) ≤ (1+𝜖) ⋅ 𝑑(𝑝𝑒𝑥𝑎𝑐𝑡𝑜,𝑞), donde 𝑝𝑒𝑥𝑎𝑐𝑡𝑜 es el vecino más cercano exacto.

**Evaluación de Complejidad:**

El enfoque exacto tiene una complejidad 𝑂(𝑛) por consulta para 𝑛 puntos, mientras que los métodos aproximados pueden reducirla a 𝑂 (log 𝑛) o sublineal dependiendo de la estructura de datos utilizada. Sin embargo, esto se logra a costa de introducir una probabilidad de error o un factor de aproximación.

**Análisis del Contexto del Problema para la cual el algoritmo es una solución**.

**Contexto general del Problema:**

En escenarios con grandes bases de datos y espacios de alta dimensionalidad, buscar vecinos exactos puede ser ineficiente. Por lo tanto, se adopta una aproximación controlada para acelerar la búsqueda y mantener una precisión aceptable. Ejemplos:

* Recuperación de imágenes similares en bases de datos visuales.
* Identificación de palabras similares en grandes colecciones de texto.

**Identificación de los conjuntos y estructuras de datos:**

1. Conjuntos de datos:

* Una colección de puntos 𝑆 = {𝑝1, 𝑝2 ,.., 𝑝𝑛} en un espacio métrico.

1. Consulta:

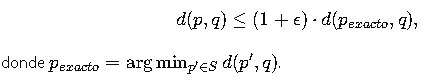
* Un punto 𝑞 en el mismo espacio métrico.

1. Estructuras utilizadas:

* Árboles de partición (por ejemplo, árboles KD).
* Locality-Sensitive Hashing (LSH) para espacios no euclidianos.

**Formalización del problema:**

Dado un conjunto 𝑆 y una métrica 𝑑 (𝑥, 𝑦) para un punto de consulta 𝑞 encontrar un punto 𝑝 ∈𝑆 que cumpla:



**Análisis de la Dinámica del Problema:**

El problema involucra dos fases principales:

1. Preprocesamiento: Construcción de la estructura de datos (LSH, árboles) para organizar los puntos 𝑆.
2. Consulta: Uso de la estructura para encontrar rápidamente un vecino aproximado.

**Evaluación de la Complejidad del Contexto:**

* Preprocesamiento: Depende de la estructura elegida. Para LSH, suele ser 𝑂 (𝑛⋅log 𝑛).
* Consulta: Se logra en 𝑂(log𝑛) o sublineal en estructuras eficientes, sacrificando exactitud.

**Identificación de los casos límites y excepciones:**

**Casos límite:**

* Alta dimensionalidad (𝑑≫100: Puede requerir estructuras específicas.
* Datos uniformemente distribuidos: Los métodos aproximados pueden ser menos efectivos.

**Excepciones:**

* Tolerancia a errores baja (𝜖→0): El problema se aproxima al caso exacto.
* Métricas no euclidianas complejas.

**Interacción con otros sistemas:**

* Aplicaciones en aprendizaje automático: Identificar vecinos en algoritmos como k-NN.
* Bases de datos: Mejorar tiempos de consulta para datos espaciales.
* Procesamiento multimedia: Encontrar coincidencias aproximadas en señales.

**Especificación Formal:**

**Definición de las entradas y salidas:**

* Entradas:
  + Conjunto de puntos 𝑆 ⊂ 𝑋.
  + Punto de consulta 𝑞 ∈𝑋.
  + Métrica 𝑑(𝑥,𝑦) para medir la distancia.
  + Factor de aproximación 𝜖>0.
* Salida:
  + Punto 𝑝 ∈ 𝑆 tal que 𝑑(𝑝,𝑞)≤(1+𝜖)⋅𝑑(𝑝𝑒𝑥𝑎𝑐𝑡𝑜,𝑞)

**Definición formal del problema mediante funciones:**

* Definición de Vecino Exacto:
* Criterio del Vecino Aproximado:

**Relaciones y Restricciones:**

* Relaciones:
  + El algoritmo depende de la métrica 𝑑(𝑥,𝑦).
  + Factor de aproximación ϵ controla el error permitido.
* Restricciones:
  + ϵ>0 para garantizar una búsqueda significativa.
  + Estructuras de datos adecuadas según 𝑑(𝑥,𝑦).

**Invariantes y propiedades:**

* Propiedad de aproximación: La solución siempre está dentro de un factor (1+𝜖) de la solución exacta.
* Robustez: Funciona en espacios de alta dimensionalidad.

**Control de Flujo:**

1. Preprocesamiento:

* Construcción de la estructura (por ejemplo, LSH).
* Configuración del parámetro ϵ.

1. Consulta:

* Evaluar 𝑞 utilizando la estructura.
* Encontrar 𝑝𝑎𝑝𝑝𝑟𝑜𝑥.

1. Verificación:

* Validar que 𝑑 (𝑝𝑎𝑝𝑝𝑟𝑜𝑥, 𝑞) ≤ (1+𝜖) ⋅ 𝑑(𝑝𝑒𝑥𝑎𝑐𝑡𝑜,𝑞).

**Pseudocodigo del Algoritmo:**

Algoritmo BusquedaAproxVecinoCercano

// Entradas:

// datos\_entrenamiento: Matriz[n][d], puntos en d dimensiones.

// punto\_consulta: Vector[d], punto objetivo.

// epsilon: Tolerancia de aproximación.

// k: Número de vecinos más cercanos requeridos.

Funcion VecinosAproximados(datos\_entrenamiento, punto\_consulta, epsilon, k)

Definir vecinos como Lista // Lista de vecinos con sus distancias

Definir r\_eps como Real // Rango inicial

// Calcular el rango basado en la tolerancia epsilon

r\_eps <- CalcularRadioInicial(datos\_entrenamiento, punto\_consulta, epsilon)

// Buscar puntos dentro del rango (1+epsilon) \* r\_eps

Para i <- 1 Hasta tamaño(datos\_entrenamiento)

distancia <- DistanciaEuclidiana(datos\_entrenamiento[i], punto\_consulta)

Si distancia <= (1 + epsilon) \* r\_eps Entonces

Añadir (datos\_entrenamiento[i], distancia) a vecinos

FinSi

FinPara

// Ordenar los vecinos por distancia

OrdenarLista(vecinos, "distancia")

// Devolver los k vecinos más cercanos

Retornar SubLista(vecinos, 1, k)

FinFuncion

Funcion CalcularRadioInicial(datos\_entrenamiento, punto\_consulta, epsilon)

// Selección de un rango inicial aproximado

r\_eps <- Infinito

Para i <- 1 Hasta tamaño(datos\_entrenamiento)

distancia <- DistanciaEuclidiana(datos\_entrenamiento[i], punto\_consulta)

Si distancia < r\_eps Entonces

r\_eps <- distancia

FinSi

FinPara

Retornar r\_eps

FinFuncion

Funcion DistanciaEuclidiana(punto1, punto2)

Definir suma como Real <- 0

Para j <- 1 Hasta tamaño(punto1)

suma <- suma + (punto1[j] - punto2[j])^2

FinPara

Retornar sqrt(suma)

FinFuncion

// Proceso principal

Escribir "Iniciando búsqueda aproximada de vecinos más cercanos..."

Definir datos\_entrenamiento como Matriz[n][d] // Cargar puntos

Definir punto\_consulta como Vector[d] // Punto de consulta

Definir epsilon como Real // Tolerancia

Definir k como Entero // Número de vecinos requeridos

vecinos <- VecinosAproximados(datos\_entrenamiento, punto\_consulta, epsilon, k)

// Mostrar resultados

Para i <- 1 Hasta tamaño(vecinos)

Escribir "Vecino ", i, ": ", vecinos[i][1], " - Distancia: ", vecinos[i][2]

FinPara

FinAlgoritmo

**Casos de usos Posible:**

* Búsqueda en Espacios de Alta Dimensionalidad:
  + Encontrar documentos similares en una base de datos de textos representados como vectores TF-IDF.
* Procesamiento de Imágenes:
  + Comparar imágenes en una base de datos usando vectores de características generados por redes neuronales.
* Recomendación de Productos:
  + Identificar productos similares a los que ha visto o comprado un usuario, usando vectores de características.
* Reconocimiento de Voz:
  + Emparejar grabaciones de audio con patrones almacenados en una base de datos.
* Detección de Anomalías:
  + Identificar puntos en un conjunto de datos que no tengan vecinos cercanos, señalando anomalías.