Penerapan Struktur KD-Tree dalam Optimasi Algoritma K-Nearest Neighbor pada Sistem Machine Learning

Christian Immanuel - 13525116
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: christian.immanuel77@gmail.com, 13525116@std.stei.itb.ac.id

***Abstrak*—Makalah ini membahas penerapan struktur KD-Tree dalam mengoptimalkan algoritma K-Nearest Neighbor pada sistem Machine Learning. Pembahasan meliputi konsep dasar KD-Tree, proses pembentukan struktur pohon, mekanisme pencarian tetangga terdekat, serta implementasinya pada algoritma K-Nearest Neighbor. Namun, K-Nearest Neighbor memiliki kelemahan pada proses pencarian tetangga terdekat yang memerlukan perhitungan jarak terhadap seluruh data, sehingga menjadi lambat ketika ukuran data semakin besar. Salah satu pendekatan untuk mengatasi masalah tersebut adalah penggunaan struktur data KD-Tree. KD-Tree membagi ruang data secara rekursif sehingga pencarian kandidat tetangga dapat dipersempit dan waktu komputasi menjadi lebih efisien.**

Kata kunci—KD-Tree, K-Nearest Neighbor, machine learning, klasifikasi, optimasi pencarian

I. PENDAHULUAN

Perkembangan teknologi kecerdasan buatan (Artificial Intelligence) dan Machine Learning telah mendorong penggunaan berbagai algoritma untuk menyelesaikan masalah klasifikasi, prediksi, dan pengenalan pola. Salah satu algoritma yang banyak digunakan adalah algoritma K-Nearest Neighbor (KNN), yang bekerja dengan mengklasifikasikan suatu data berdasarkan sejumlah tetangga terdekat dari data tersebut. Algoritma K-Nearest Neighbor (KNN) banyak digunakan dalam klasifikasi dan prediksi karena sederhana dan mudah diimplementasikan. Algoritma ini banyak diimplementasikan pada bidang-bidang seperti pengenalan wajah, klasifikasi gambar, diagnosis penyakit, serta sistem rekomendasi.

Meskipun memiliki konsep yang sederhana dan tingkat akurasi yang cukup baik, algoritma K-Nearest Neighbor memiliki kelemahan dalam hal efisiensi pencarian tetangga terdekat. Umumnya proses pencarian dilakukan dengan membandingkan data uji terhadap seluruh data pelatihan (brute force search), sehingga waktu komputasi akan semakin meningkat dengan bertambahnya jumlah data

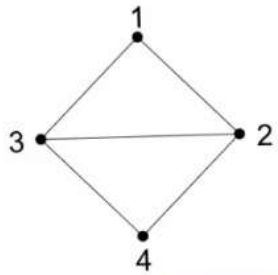
Salah satu struktur data yang dapat digunakan untuk meningkatkan efisiensi pencarian tetangga terdekat adalah KD-Tree (K-Dimensional Tree). KD-Tree merupakan struktur pohon biner yang digunakan untuk menyimpan dan mengorganisasi data yang berdimensi banyak. Dengan pembagian ruang yang terorganisasi, KD-Tree membuat proses pencarian tetangga terdekat dilakukan dengan lebih efisien karena KD-Tree dapat memangkas jumlah kandidat yang perlu diperiksa saat melakukan pencarian tetangga sehingga lebih hemat waktu dibandingkan metode pencarian secara linear.

Makalah ini akan membahas penerapan struktur KD-Tree dalam mengoptimalkan algoritma K-Nearest Neighbor pada sistem *Machine Learning*. Pembahasan akan mencakup konsep dasar KD-Tree, proses pembentukan struktur pohon, mekanisme pencarian tetangga terdekat, serta analisis efisiensi yang diperoleh dibandingkan dengan metode pencarian secara *brute force*.

II. DASAR TEORI

A. Graf

Graf merupakan struktur dasar yang digunakan untuk merepresentasikan hubungan antara objek-objek diskrit. Secara matematis, graf didefinisikan sebagai G = (V, E), dengan V adalah himpunan tidak kosong dari simpul (vertices), dan E sebagai himpunan sisi (edge) yang menghubungkan pasangan simpul.



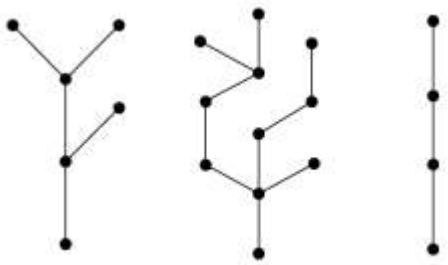
Gambar 2.1 Contoh Graf
Sumber : LMS-SPADA INDONESIA

B. Pohon

Pohon (tree) adalah graf tak berarah yang terhubung (connected) dan tidak mengandung sirkuit (cycle).

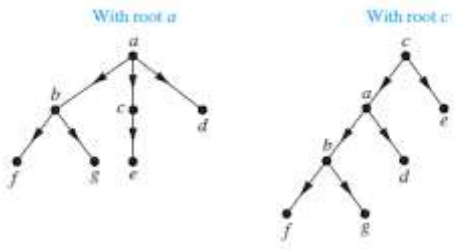
Misalkan G = (V, E) adalah graf tak-berarah sederhana dan jumlah simpulnya n. Maka, semua pernyataan di bawah ini adalah ekivalen:

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki m = n – 1 buah sisi.
4. G tidak mengandung sirkuit dan memiliki m = n – 1 buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.



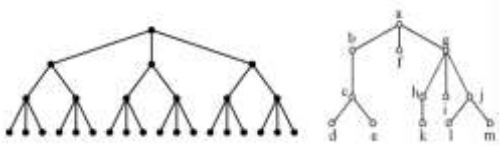
Gambar 2.2 Pohon
Sumber: Rinaldi Munir Pohon Bagian 1 2026

Pohon berakar (rooted tree) adalah pohon yang satu buah simpulnya berperan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah yang mengalir dari akarnya menuju simpul-simpul di bawahnya.



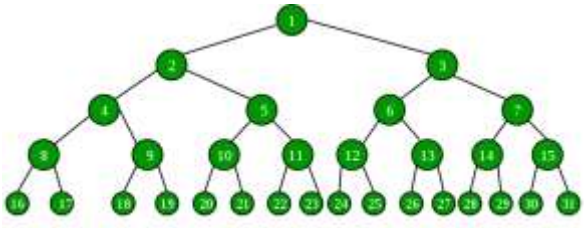
Gambar 2.3 Pohon Berakar
Sumber: Pengantar Pohon Telkom University

Pohon n-ary adalah Pohon berakar yang setiap simpul cabangnya mempunyai paling banyak n buah anak



Gambar 2.4 Pohon 3-ary
Sumber: Rinaldi Munir Pohon Bagian 2 2026

Pohon biner adalah struktur pohon di mana setiap simpul (node) memiliki paling banyak dua anak, yaitu anak kiri (left child) dan anak kanan (right child). Struktur ini banyak diimplementasikan dalam berbagai algoritma pemrograman. KD-Tree merupakan salah satu bentuk khusus dari pohon biner yang digunakan untuk menyimpan data multidimensi.

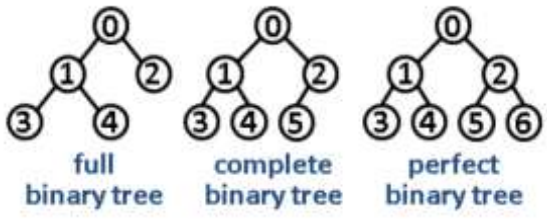


Gambar 2.5 Pohon Biner
Sumber: geeksforgeeks

Full binary tree: setiap simpul, kecuali simpul pada aras daun, memiliki tepat dua anak

Complete binary tree: setiap aras, mungkin kecuali pada aras terakhir, terisi lengkap, dan semua simpul dipadatkan ke bagian kiri jika memungkinkan

Perfect Binary Tree adalah jenis pohon biner yang setiap simpul internalnya memiliki tepat dua anak serta semua simpul daun berada pada level atau kedalaman yang sama.



Gambar 2.6 Jenis-jenis Pohon Biner
Sumber: Rinaldi Munir Pohon Bagian 2 2026

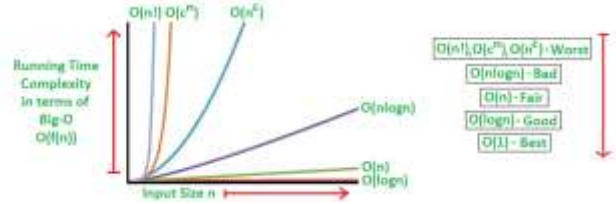
C. Kompleksitas Algoritma

Kompleksitas algoritma merupakan ukuran yang digunakan untuk menganalisis efisiensi suatu algoritma berdasarkan jumlah sumber daya komputasi yang dibutuhkan. Pada umumnya, sumber daya yang dianalisis adalah waktu eksekusi (time complexity) dan penggunaan memori (space complexity). Dalam makalah ini, lebih difokuskan pada pembahasan kompleksitas waktu karena tujuan utama penerapan KD-Tree adalah mempercepat proses pencarian tetangga terdekat pada algoritma K-Nearest Neighbor.

Notasi Big-O digunakan untuk menyatakan batas atas (upper bound) dari kebutuhan waktu suatu algoritma. Menurut definisi formal, fungsi T(n) dikatakan memiliki kompleksitas O(f(n)) apabila terdapat konstanta positif C dan n_0 sehingga

$$T(n) \leq C \cdot f(n) \text{ untuk setiap } n \geq n_0$$

f(n) adalah batas lebih atas (upper bound) dari T(n) untuk n yang besar.



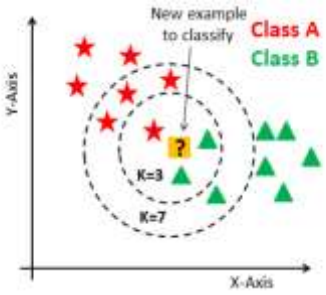
Grafik 2.7 Big-O Notation

Sumber: geeksforgeeks

D. *K-Nearest Neighbor (KNN)*

K-Nearest Neighbor (KNN) merupakan salah satu algoritma yang dipakai dalam supervised learning yang digunakan untuk menyelesaikan permasalahan klasifikasi. Algoritma ini bekerja dengan menentukan kelas suatu data berdasarkan sejumlah K data yang memiliki jarak paling dekat terhadap data tersebut. Dalam kasus klasifikasi, kelas ditentukan menggunakan prinsip majority voting, sedangkan pada regresi nilai prediksi diperoleh dari rata-rata nilai tetangga terdekat.

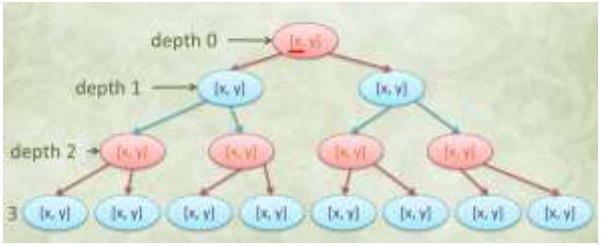
Dalam algoritma KNN, setiap data direpresentasikan sebagai sebuah titik pada ruang berdimensi K. Misalkan suatu data memiliki n atribut: $X = (x_1, x_2, x_3, \ldots, x_n)$. Maka data tersebut dapat dianggap sebagai titik pada ruang berdimensi K.



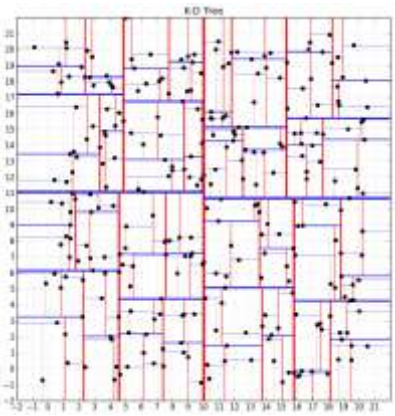
Gambar 2.8 K-Nearest Neighbor
Sumber: geeksforgeeks

E. *Pohon KD*

KD-Tree (K-dimensional tree) adalah struktur pohon pencarian biner yang dirancang untuk data berdimensi banyak. KD-Tree membagi ruang berdimensi K ke dalam beberapa wilayah (kotak-kotak berdimensi K) secara rekursif dengan membagi dataset menurut nilai median di tiap simpul. Setiap simpul pohon berhubungan dengan satu hyperplane yang memotong ruang pada dimensi tertentu. Sebagai contoh, untuk data 2 dimensi, simpul akar mungkin membagi data berdasarkan median menurut sumbu x, kemudian anak-anaknya membagi berdasarkan median sumbu y, dan anaknya membagi lagi berdasarkan median sumbu x dan seterusnya secara berulang.



Gambar 2.9 Pohon KD
Sumber: Stable Sort KD-Tree Nearest Neighbor Data Structure



Grafik 2.10 Visualisasi Pohon KD
Sumber: Salzi Blog Kd-tree and Nearest neighbor (NN) search (2D case)

III. METODE PENELITIAN

A. *Pencarian Tetangga Terdekat dengan Algoritma KNN Konvensional dalam Sistem Machine Learning*

Pada sistem Machine Learning berbasis supervised learning, K-Nearest Neighbor merupakan salah satu algoritma yang paling sederhana dan banyak digunakan untuk permasalahan klasifikasi. K-Nearest Neighbor merupakan algoritma klasifikasi yang menentukan kelas suatu data berdasarkan sejumlah tetangga terdekat dari data tersebut. Untuk memperoleh tetangga terdekat, algoritma harus menghitung jarak antara data uji dengan seluruh data latih yang tersedia.

Langkah utama dalam algoritma KNN adalah menghitung jarak antara data uji dan data latih. Semakin kecil jarak dua titik, semakin tinggi tingkat kemiripannya.

Salah satu metode yang paling umum digunakan adalah Euclidean Distance.

Misalkan terdapat dua titik:

$$P = (p_1, p_2, \ldots, p_n)$$

$$Q = (q_1, q_2, \ldots, q_n)$$

maka jarak Euclidean antara kedua titik tersebut didefinisikan sebagai:

$$d(P,Q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Untuk data dua dimensi, rumusnya menjadi:

$$d(x,y) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

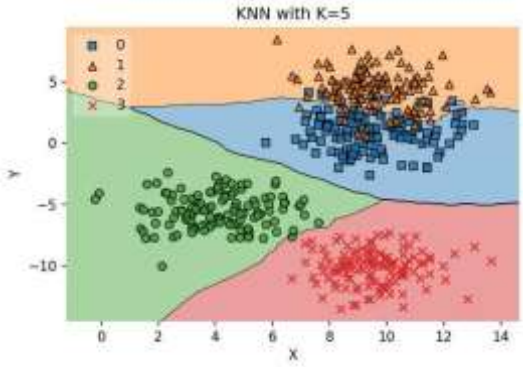
Nilai jarak inilah yang digunakan untuk menentukan tetangga terdekat dalam algoritma KNN.

Secara umum, proses klasifikasi menggunakan KNN dilakukan melalui langkah-langkah berikut:

1. Menentukan nilai K.

2. Menghitung jarak antara data uji dengan seluruh data latih.
3. Mengurutkan hasil perhitungan jarak dari yang terkecil.
4. Memilih K data dengan jarak terdekat.
5. Menentukan kelas berdasarkan mayoritas kelas dari *K*tetangga tersebut.

Misalkan terdapat suatu data uji X yang memiliki label kelas A, A, dan B. Karena majoritas adalah kelas A maka X diklasifikasikan ke dalam kelas A.



Grafik 3.1 Visualisasi KNN
Sumber: geeksforgeeks

Misalkan tersedia data latih machine learning berikut:

X	Y	Kelas
2	3	A
5	4	A
9	6	B
4	7	A
8	1	B
7	2	B

Lalu terdapat data uji: Q = (3,3) yang belum diketahui kelasnya.

Misalkan ditentukan nilai K = 3, maka program mengambil 3 titik terdekat berdasarkan perhitungan yaitu titik (2,3), (5,4), dan (4,7). Karena ketiga titik ini adalah kelas A maka Titik Q adalah kelas A.

Maka dengan Algoritma KNN konvensional, program akan bruteforce melakukan perhitungan kepada semua titik yang ada pada data set

Titik	Kelas	Jarak
(2,3)	A	1
(5,4)	A	2.24
(4,7)	A	4.12
(7,2)	B	4.12
(8,1)	B	5.39
(9,6)	B	6.71

Dengan algoritma KNN secara konvensional harus menghitung jarak antara titik data uji dengan setiap data latih tersebut. Metode brute force ini mampu menghasilkan klasifikasi yang akurat, biaya komputasinya meningkat secara linier terhadap jumlah data yang digunakan T(n) = O(n). Hal inilah yang menjadi kelemahan utama KNN karena harus menghitung jarak antara data uji dengan seluruh data latih yang dapat menjadi tidak efisien dengan bertambahnya data.

```python
def euclidean_distance(a, b):
    return sum((x - y) ** 2 for x, y in zip(a, b))

def knn_bruteforce(X_train, y_train, target, k):
    distances = []
    for point, label in zip(X_train, y_train):
        d = euclidean_distance(target, point)
        distances.append((d, label))

    distances.sort(key=lambda x: x[0])
    k_nearest = distances[:k]
    votes = {}

    for _, label in k_nearest:
        votes[label] = votes.get(label, 0) + 1
    return max(votes, key=votes.get)
```

Gambar 3.2 Kode KNN dengan Metode Bruteforce

B. Penerapan KD-Tree pada Algortima KNN dalam Sistem Machine Learning

KD-Tree diterapkan sebagai struktur indeks yang mengorganisasi data latih pada sistem machine learning ke dalam bentuk pohon biner multidimensi. Pada algoritma K-Nearest Neighbor (KNN), struktur ini memungkinkan proses klasifikasi dilakukan dengan lebih efisien dalam pencarian tetangga terdekat.

Tahapan penerapannya adalah sebagai berikut:

1. Seluruh data latih dimasukkan ke dalam KD-Tree.
2. Pohon dibangun dengan membagi data berdasarkan median pada dimensi tertentu.

   Kita urutkan datanya terlebih dahulu berdasarkan nilai x : (2,3), (4,7), (5,4), (7,2), (8,1), (9,6)

   Sehingga didapatkan median (7,2), inilah yang menjadi akar dari pohon KD.

   Kita juga mengetahui data di bawah median yaitu (2,3), (4,7), (5,4) dan data di atas median yaitu (8,1), (9,6).

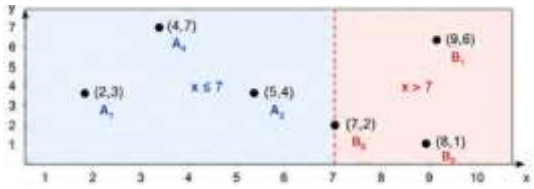
   Disini kita urutkan data dari kedua kelompok data itu sekarang berdasarkan sumbu y

   y Kelompok 1: (2,3), (5,4), (4,7)

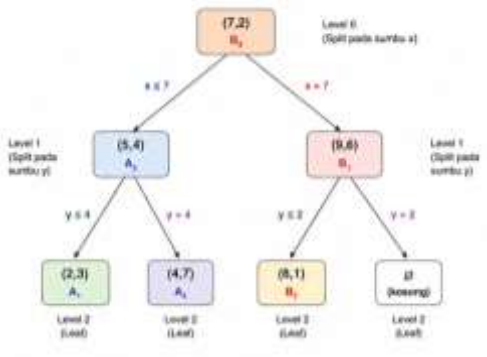
   y Kelompok 2 : (8,1), (9,6)

Kita ambil median dari kedua kelompok data yaitu : (5,4) dan (9,6)

Lalu kita ulangi proses ini secara rekursif.



Grafik 3.3 Visualisasi Pembagian Wilayah



Gambar 3.4 Pohon KD-Tree yang terbentuk

3. Saat data uji diberikan, Algoritma melakukan pencarian dengan cabang yang sesuai dengan titik yang diuji. Pencarian akan dimulai dengan membandingkan dan menghitung jarak antara data uji: Q = (3,3) dengan akar (7,2)

$$d\big((3,3),(7,2)\big) = \sqrt{(3-7)^2 + (3-2)^2} \;=\; 4.12$$

Nilai jarak ini akan disimpan sebagai jarak maksimum kandidat. Lalu karena 3 < 7 (membandingkan sesama sumbu x) maka algoritma bergerak ke subtree bagian kiri.

Program kemudian membandingkan titik Q dengan (5, 4). Pada setiap simpul yang dikunjungi, dihitung jarak terhadap uji titik.

$$d\big((3,3),(5,4)\big) = \sqrt{(3-5)^2 + (3-4)^2} \;=\; 2.23$$

Karena jarak 2.23 lebih kecil dibandingkan dengan jarak maksimum sementara. Maka jarak maksimum tetap. Lalu karena 3 < 4 maka bergerak ke subtree bagian kiri lagi (membandingkan sesama sumbu y).

4. Saat titik uji sampai di daun maka algoritma akan menghitung jarak antar kedua titik.

$$d\big((3,3),(2,3)\big) = \sqrt{(3-2)^2 + (3-3)^2} \;=\; 1$$

Karena jarak yang didapatkan lebih kecil lagi, maka jarak maksimum tetap sama.

Karena K = 3 maka kandidat KNN adalah (2,3), (5,4), (7,2). Apabila terdapat titik yang lebih dekat terhadap Q dibandingkan dengan titik (7,2) terhadap Q maka kandidat titik (7,2) disingkirkan serta jarak maksimum kandidat diperbarui lagi

5. Algoritma juga harus mengecek apakah jarak maksimum yang disimpan sudah kurang dari jarak sumbu y titik uji dengan pemisah (5,4) karena terdapat beberapa kasus dimana titik yang terdekat tidak berada di cabang yang dilalui.

   Jarak sumbu y titik q dengan pemisah : |3 – 4| = 1

   Karena Jarak terdekat maksimum yang disimpan (4.12) ≤ Jarak sumbu y titik Q dengan pemisah (1) maka Algoritma melakukan backtracking.

6. Algoritma melakukan backtracking lagi dan mengecek jarak Q dengan (4,7) untuk mencari kandidat jarak yang lebih dekat dengan jarak kandidat maksimum itu.

$$d\big((3,3),(4,7)\big) = \sqrt{(4-2)^2 + (7-3)^2} \;=\; 4.12$$

   Karena 4.12 = 4.12 maka dapat dipilih titik (4,7) ataupun titik (7,2) sebagai kandidat K tetangga terdekat dengan jarak yang paling jauh.

7. Algoritma lalu menentukan K tetangga terdekat, setelah proses menghitung jarak terdekat selesai. Program mengambil 3 kandidat jarak minimum dan menentukan kelas titik Q.

Titik	Kelas	Jarak
(2,3)	A	1
(5,4)	A	2.24
(4,7)	A	4.12

Disini Program menemukan bahwa Q diklasifikasikan ke dalam kelas A

```
class KDNode:
    def __init__(self, point, label, axis):
        self.point = point
        self.label = label
        self.axis = axis
        self.left = None
        self.right = None

def distance_squared(a, b):
    return sum((x - y) ** 2 for x, y in zip(a, b))

def build_kdtree(points, labels, depth=0):
    if not points:
        return None

    dim = len(points[0])
    axis = depth % dim

    data = list(zip(points, labels))
    data.sort(key=lambda item: item[0][axis])

    mid = len(data) // 2

    node = KDNode(data[mid][0], data[mid][1], axis)

    left_data = data[:mid]
    right_data = data[mid + 1:]

    node.left = build_kdtree([p for p, l in left_data], [l for p, l in left_data], depth + 1)
    node.right = build_kdtree([p for p, l in right_data], [l for p, l in right_data], depth + 1)
    return node
```

Gambar 3.5 Kode KNN dengan Metode Pohon KD

```python
def knn_search(node, target, k, neighbors):
    if node is None:
        return
    d2 = distance_squared(target, node.point)
    neighbors.append((d2, node.label))
    neighbors.sort(key=lambda x: x[0])

    if len(neighbors) > k:
        neighbors.pop()

    axis = node.axis
    diff = target[axis] - node.point[axis]

    if diff < 0:
        near = node.left
        far = node.right
    else:
        near = node.right
        far = node.left

    knn_search(near, target, k, neighbors)

    if len(neighbors) < k or diff * diff <= neighbors[-1][0]:
        knn_search(far, target, k, neighbors)


def classify(root, target, k):
    neighbors = []
    knn_search(root, target, k, neighbors)

    votes = {}

    for _, label in neighbors:
        votes[label] = votes.get(label, 0) + 1

    return max(votes, key=votes.get)
```

Gambar 3.6 Kode KNN dengan Metode Pohon KD

IV. Analisis

A. *Perbandingan KNN Brute Force antara KNN dengan KD-Tree Seiring Bertambahnya Data Latih Machine Learning*

Proses pencarian tetangga terdekat secara brute force membutuhkan perhitungan jarak terhadap seluruh data pelatihan, sehingga biaya komputasi meningkat seiring bertambahnya ukuran dataset. Untuk mengatasi permasalahan tersebut, struktur KD-Tree digunakan sebagai metode untuk space partitioning yang memungkinkan proses pencarian nearest neighbor dilakukan secara lebih efisien. Dengan pendekatan ini, jumlah titik yang perlu diperiksa oleh komputer menjadi jauh lebih sedikit dibandingkan metode brute force sehingga bisa mempercepat proses machine learning.

Jumlah Data Latih	Jumlah Data Uji	Akurasi	Waktu Brute Force KNN	Waktu Pohon KD dengan KNN	Rata-rata Node yang Dikunjungi Pohon KD
1050	450	100%	0,022585	0,045779	24,49
2100	900	99.8%	0,085261	0,099409	26,52
6.300	2.700	99.7%	0,632462	0,315177	28,18
10.500	4.500	99.8%	1,669638	0,532595	29,31
21.000	9.000	99.8%	7,801986	1,186643	30,41

Tabel 4.1 Perbandingan Efisiensi KNN Brute Force dan KNN dengan KD-Tree

Berdasarkan hasil pengujian, algoritma KNN berbasis KD-Tree belum memberikan keuntungan yang signifikan pada dataset berukuran kecil karena pembangunan struktur pohon masih tergolong rendah. Namun, ketika jumlah data latih meningkat, perbedaan performa menjadi semakin terlihat. Pada dataset dengan 21.000 data latih, metode brute force memerlukan waktu 7,802 detik, sedangkan KD-Tree hanya membutuhkan 1,187 detik termasuk proses pembangunan pohon.

Selain itu, KD-Tree juga mampu mengurangi jumlah titik yang perlu diperiksa secara signifikan. Pada dataset terbesar, brute force harus menghitung jarak terhadap seluruh 21.000 titik, sedangkan KD-Tree hanya mengunjungi rata-rata 30,41 simpul. Hal ini menunjukkan bahwa mekanisme pruning pada KD-Tree berhasil menghilangkan hampir seluruh dari dataset pencarian tanpa mengurangi akurasi klasifikasi yang dihasilkan. Hal ini membuktikan bahwa KD-Tree merupakan struktur data yang efektif untuk mengoptimalkan algoritma K-Nearest Neighbor pada dataset berukuran besar.

B. *Perbandingan KNN Brute Force antara KNN dengan KD-Tree Seiring Bertambahnya Dimensi Data Machine Learning*

Dimensi	Akurasi	Waktu Brute Force KNN	Waktu Pohon KD dengan KNN	Rata-rata Node yang Dikunjungi Pohon KD
2D	99.8%	7,801986	1,186643	30,41
3D	99.9%	7,552473	2,332559	63,71
4D	100%	7,783823	4,607037	142,04
5D	100%	8,723997	9,688802	312,21
6D	100%	8.187276	17,466107	632.22

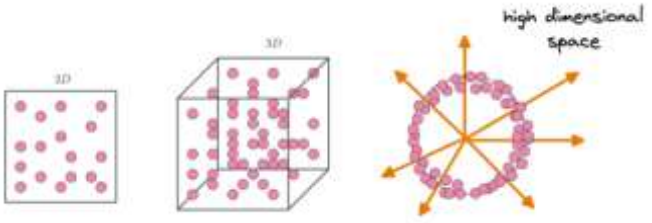
Tabel 4.2 Pengaruh Jumlah Dimensi terhadap Efisiensi KD-Tree dengan 21.000 Data Latih dan K = 3

Hasil pengujian menunjukkan bahwa peningkatan jumlah dimensi menyebabkan performa KD-Tree mengalami penurunan secara bertahap Pada dataset dua dimensi, KD-Tree hanya mengunjungi rata-rata 30,41 simpul dari total 21.000 data latih dan mampu menyelesaikan proses pencarian dalam waktu 1,187 detik. Hal ini menunjukkan bahwa mekanisme pruning bekerja sangat efektif pada dimensi itu.

Ketika jumlah dimensi meningkat menjadi tiga dan empat dimensi, jumlah simpul yang harus diperiksa meningkat menjadi 63,71 dan 142,04 simpul. Akibatnya waktu pencarian juga meningkat menjadi 2,333 detik dan 4,607 detik. Disini KD-Tree masih lebih cepat dibandingkan metode brute force. Namun pada lima dimensi, rata-rata simpul yang dikunjungi meningkat hingga 312,21 simpul. Waktu total KD-Tree mencapai 9,689 detik, bahkan sedikit lambat dibandingkan brute force yang membutuhkan 8,724 detik. Pada saat enam dimensi, perbedaan

metode brute force dan KD-Tree semakin terlihat, dengan rata-rata node yang dikunjungi menjadi 632,22 dan metode KD-Tree menghabiskan 17,466107 detik dimana itu adalah waktu yang jauh lebih lama dibandingkan dengan metode bruteforce.

Hal ini menunjukkan bahwa semakin tinggi dimensi data, semakin sulit KD-Tree melakukan pruning karena banyak wilayah ruang yang masih berpotensi mengandung tetangga terdekat. Akibatnya jumlah simpul yang harus dikunjungi meningkat dan keuntungan penggunaan KD-Tree berkurang. Hal inilah yang disebut dengan Curse of Dimensionality, yaitu kondisi ketika semakin bertambahnya dimensi data dapat menyebabkan struktur dan algoritma yang bergantung pada pembagian ruang, seperti KD-Tree, kehilangan efisiensinya. Hal ini terjadi karena data yang cenderung menjadi lebih tersebar karena banyaknya dimensi sehingga metode dan performa KD-Tree menjadi lebih buruk daripada metode Brute Force.



Gambar 4.3 Curse of Dimensionality

Sumber: Daily Dose of Data Science

V. KESIMPULAN

K-Nearest Neighbor (KNN) merupakan salah satu algoritma klasifikasi dalam machine learning yang sederhana dan memiliki tingkat akurasi yang baik, namun proses pencarian tetangga terdekat menggunakan metode brute force memerlukan biaya komputasi yang tinggi karena harus membandingkan data uji dengan seluruh data latih. Seiring bertambahnya jumlah data, waktu pencarian juga meningkat sehingga efisiensi sistem menurun.

Berdasarkan hasil pengujian, penggunaan struktur KD-Tree mampu mengoptimalkan proses pencarian tetangga terdekat pada algoritma KNN. Melalui metode pembagian ruang dan pruning, KD-Tree dapat mengurangi jumlah titik yang diperiksa sehingga waktu klasifikasi menjadi lebih cepat tanpa mengurangi akurasi.

Meskipun demikian, efektivitas KD-Tree menurun pada data berdimensi tinggi akibat fenomena Curse of Dimensionality. Oleh karena itu, KD-Tree lebih cocok digunakan pada dataset berukuran besar dengan dimensi rendah hingga menengah. Penerapan KD-Tree dapat digunakan sebagai solusi yang efektif untuk meningkatkan efisiensi algoritma KNN dan mendukung pengembangan sistem machine learning yang lebih cepat.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya. Penulis juga menyampaikan penghargaan kepada Prof. Dr. Ir. Rinaldi, M.T. sebagai dosen mata kuliah Matematika Diskrit yang telah memberikan kesempatan serta bimbingan dalam mengerjakan makalah ini. Penulis berharap makalah ini dapat memberikan manfaat dan menambah wawasan bagi pembaca mengenai implementasi algoritma K-Nearest Neighbor (KNN) dan KD-Tree.

REFERENSI

[1] R. Munir, "Graf (Bagian 1)," Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Indonesia, 2026.

[2] R. Munir, "Pohon (Bagian 1)," Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Indonesia, 2026.

[3] R. Munir, "Pohon (Bagian 2)," Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Indonesia, 2026.

[4] R. Munir, "Kompleksitas Algoritma (Bagian 2)," Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Indonesia, 2026.

[5] GeeksforGeeks, "K-Nearest Neighbor (KNN) Algorithm," GeeksforGeeks, 23 Desember 2025. [Daring]. Tersedia di: https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/ [Diakses: 13 Juni 2026].

[6] GeeksforGeeks, "KD Trees in C++," GeeksforGeeks, 23 Desember 2025. [Daring]. Tersedia di: https://www.geeksforgeeks.org/cpp/kd-trees-in-cpp/ [Diakses: 13 Juni 2026].

[7] V. R. Tiwari, "Developments in KD Tree and KNN Searches," Jai Hind College, Churchgate, Mumbai, India, 2023.

[8] GeeksforGeeks, "How to Visualize KNN in Python" GeeksforGeeks, 23 Juli 2025. [Daring]. Tersedia di: https://www.geeksforgeeks.org/machine-learning/how-to-visualize-knn-in-python/

[Diakses: 15 Juni 2026].

[9] Stable Sort, "KD-Tree Nearest Neighbor Data Structure," YouTube, 9 Oktober 2020 [Daring]. Tersedia di: https://www.youtube.com/watch?v=Glp7THUpGow.

[Diakses: Jun. 15, 2026].

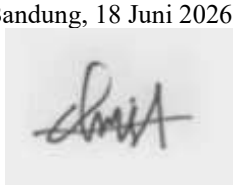
[10] DataMListic, "The Curse of Dimensionality," YouTube, 19 Maret 2025 [Daring]. Tersedia di: https://www.youtube.com/watch?v=9Tf-_mJhOkU

[Diakses: Jun. 17, 2026].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Juni 2026



Christian Immanuel
13525116