Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Convex Hull

Bryan Amirul Husna / 13520146 / K02

A. Algoritma *Divide and Conquer* untuk Pencarian *Convex Hull*

1. Urutkan titik-titik berdasarkan koordinat x nya menaik (langkah prekondisi)
2. Masukkan titik berkoordinat x terkecil dan titik berkoordinat x terbesar ke daftar titik solusi
3. Buat garis antara titik berkoordinat x terkecil (titik kiri) dengan terbesar (titik kanan).
4. Kelompokkan titik-titik lain sebagai di atas/kiri garis atau di bawah/kanan garis (titik yang terletak tepat pada garis diabaikan). Lakukan langkah 5-6 pada **kelompok titik di atas garis**.
5. Pada kelompok titik, jika hanya tersisa satu titik, masukkan titik ini ke daftar titik solusi (kasus basis).
6. Jika tidak, cari titik dengan jarak terjauh dari garis titik kiri-kanan. Masukkan titik ini ke daftar titik solusi. Tarik garis antara titik kiri - titik terjauh (garis kiri) dan titik terjauh - titik kanan (garis kanan), kemudian kelompokkan menjadi titik-titik yang berada di atas garis kiri dan di atas garis kanan. Titik-titik yang berada di bawah garis diabaikan. Lakukan kembali langkah 5-6 pada tiap kelompok titik.
7. Lakukan juga langkah 5-6 pada kelompok **titik yang berada di bawah garis** pada langkah 4. Akan tetapi pada langkah 6, jika sebelumnya diambil titik yang ada di atas garis, kali ini diambil titik yang ada di bawah garis tiap langkah.

Algoritma ini memiliki efisiensi rata-rata linear (n), dengan efisiensi terburuk kuadratik mirip quicksort (n^2) [Levitin, Anany. 2012. *Introduction to the Design and Analysis of Algorithms, Third Edition*. Harlow: Pearson Education Limited]. Akan tetapi, jika pengurutan pada prekondisi diperhitungkan, algoritma akan memiliki efisiensi rata-rata linearitmik (n logn) jika diasumsikan menggunakan pengurutan quicksort.

B. *Source Code*

- **bryanhull.py**, berisi algoritma utama pencarian *Convex Hull*

src > bryanhull.py > ...

```python
# Convex Hull
# Bryan Amirul Husna / 13520146 / K02
from line import Line
from utils import *

# Class untuk menemukan convex hull
# Hasil langsung dihitung ketika constructor dipanggil, disimpan di atribut simplices
class ConvexHull:
    # vertices adalah list of point (x, y)
    #vertices = []
    #convexhullpoints = []
    #simplices = []
    def __init__(self, vertices):
        self.vertices = []
        self.convexhullpoints = []
        self.simplices = []
        i = 0
        for vertex in vertices:
            self.vertices.append(vertex.copy() + [i])
            i += 1
        self.convexHullInitial()
```

```python
    # convexHullInitial untuk menangani pemanggilan pertama
    def convexHullInitial(self):
        # Mengecek kasus khusus
        if(len(self.vertices) == 0):
            return
        elif(len(self.vertices) == 1):
            self.convexhullpoints.append(self.vertices[0])
            return

        self.vertices.sort(key=lambda k: [k[0], k[1]], reverse=False)   # Mengurutkan berdasarkan x menaik
        upIdxArr = []
        downIdxArr = []
        n = len(self.vertices)

        # Titik paling kiri (0) dan kanan (n-1) termasuk dalam convex hull
        self.convexhullpoints.append(self.vertices[0])
        self.convexhullpoints.append(self.vertices[n-1])

        # Menambahkan titik-titik ke atas atau bawah
        for i in range(1, n-1):
            det = calculateDeterminan(self.vertices[0], self.vertices[n-1], self.vertices[i])
            if(det > 0):
                upIdxArr.append(i)
            elif(det < 0):
                downIdxArr.append(i)
            # Jika det == 0, tidak diolah karena jelas bukan titik convex hull

        # Penentuan titik convex hull bagian atas
        if(len(upIdxArr) == 0): # Jika bagian atas kosong, masukkan [0, n-1] sebagai salah satu jalur
            self.simplices.append([0, n-1])
        else:   # Jika tidak panggil rekursif
            self.convexHull(upIdxArr, 0, n-1)

        # Penentuan titik convex hull bagian bawah
        # array bagian bawah dibalik karena bagian bawah dapat dihitung memakai kode yang sama dengan bagian atas, tetapi dibalik
        if(len(downIdxArr) == 0):
            self.simplices.append([n-1, 0])
        else:
            downIdxArr.reverse()
            self.convexHull(downIdxArr, n-1, 0)

        for i in range(len(self.simplices)):
            self.simplices[i] = [self.vertices[self.simplices[i][0]][2], self.vertices[self.simplices[i][1]][2]]

    def convexHull(self, idxArr, leftPointIdx, rightPointIdx):
        if(len(idxArr) == 0):
            return
        if(len(idxArr) == 1):  # Kalau tersisa satu titik, titik tersebut masuk convex hull
            self.convexhullpoints.append(self.vertices[idxArr[0]])
            self.simplices.append([leftPointIdx, idxArr[0]])
            self.simplices.append([idxArr[0], rightPointIdx])
        elif(len(idxArr) > 1):
            leftPoint = self.vertices[leftPointIdx]
            rightPoint = self.vertices[rightPointIdx]
            # Menentukan titik terjauh
            line = Line()
            line.setTwoPoints(leftPoint, rightPoint)    # Membuat garis antara titik kiri dan kanan, untuk pengukuran jarak
            t = self.vertices[idxArr[0]]  # titik terjauh saat ini
            tidx = 0
            tdist = line.pointToLineDistance(t)
            currdist = tdist
            n = len(idxArr)
            for i in range(1, n):
                currdist = line.pointToLineDistance(self.vertices[idxArr[i]])
                if(currdist > tdist):
                    tdist = currdist
                    tidx = i
            t = self.vertices[idxArr[tidx]]
            self.convexhullpoints.append(t)    # Titik terjauh masuk convex hull
```

```
            t = self.vertices[idxArr[tidx]]
            self.convexhullpoints.append(t)    # Titik terjauh masuk convex hull

            # Pembuatan daftar titik selanjutnya yang akan direkursi
            leftArr = []
            for i in range(tidx):
                det = calculateDeterminan(leftPoint, t, self.vertices[idxArr[i]])
                if(det > 0):
                    leftArr.append(idxArr[i])
            rightArr = []
            for i in range(tidx+1, n):
                det = calculateDeterminan(t, rightPoint, self.vertices[idxArr[i]])
                if(det > 0):
                    rightArr.append(idxArr[i])

            # Solve subproblem
            if(len(leftArr) == 0):
                self.simplices.append([leftPointIdx, idxArr[tidx]])
            else:
                self.convexHull(leftArr, leftPointIdx, idxArr[tidx])

            if(len(rightArr) == 0):
                self.simplices.append([idxArr[tidx], rightPointIdx])
            else:
                self.convexHull(rightArr, idxArr[tidx], rightPointIdx)
```

- **bryanhull_test.py**, berisi tes-tes pengujian untuk pencarian *convex hull*

```
from bryanhull import ConvexHull

# Test for Up Side
t1 = [[0, 0], [2, 2], [4, 0]]
#tr1 = ConvexHull(t1)
#print(tr1.simplices)

t2 = [[0, 0], [2, 2], [4, 0], [2, 3]]
#tr2 = ConvexHull(t2)
#print(tr2.simplices)

t3 = [[0, 0], [2, 2], [4, 0], [2, 3], [0.41, 1.6], [3.61, 1.6]]
#tr3 = ConvexHull(t3)
#print(tr3.simplices)

t4 = [[0, 0], [2, 2], [4, 0], [2, 3], [0.41, 1.6], [3.61, 1.6], [2.85, 1.66]]
#tr4 = ConvexHull(t4)
#print(tr4.simplices)

# Test for Up and Down Side
t5 = [[0, 0], [4, 0], [2, 2], [2, 1], [2, -1], [2, -2]]
#tr5 = ConvexHull(t5)
#print(tr5.simplices)

t6 = [[0, 0], [4, 0], [2, 2], [2, 1], [2, -1], [2, -2], [4, -1], [0.5, -1]]
#tr6 = ConvexHull(t6)
#print(tr6.simplices)

t7 = [[0, 0], [2, 2], [4, 0], [2, 3], [0.41, 1.6], [3.61, 1.6], [2.85, 1.66], [2, -1], [2, -2], [4, -1], [0.5, -1]]
#tr7 = ConvexHull(t7)
#print(tr7.convexhullpoints)
#print(tr7.simplices)
```

```
t7 = [[0, 0], [2, 2], [4, 0], [2, 3], [0.41, 1.6], [3.61, 1.6], [2.85, 1.66], [2, -1], [2, -2], [4, -1], [0.5, -1]]
#tr7 = ConvexHull(t7)
#print(tr7.convexhullpoints)
#print(tr7.simplices)

t8 = [[5.1, 3.5], [4.9, 3.0], [4.7, 3.2], [4.6, 3.1], [5.0, 3.6], [5.4, 3.9], [4.6, 3.4], [5.0, 3.4], [4.4, 2.9], [4.9, 3.1
tr8 = ConvexHull(t8)
print(tr8.convexhullpoints)
print(tr8.simplices)
```

- **line.py**, berisi kelas Line untuk mempermudah perhitungan jarak dari titik ke garis-dua-titik

```python
# Class Line merepresentasikan garis ax + by + c = 0
# gradien m = -a/b
class Line:
    isTwoPoint = False  # True jika garis ini direpresentasikan oleh dua titik p1 dan p2, False jika direpresentasikan a, b, c
    # Line mempunyai variabel a, b, c
    # ATAU
    # p1 dan p2
    # Selain a, b, c, p1, dan p2, juga menghitung gradien m

    # Karena Python tidak mendukung banyak konstruktor, setelah konstruksi panggil newLine(...)
    def __init__(self):
        pass

    # Setter dengan masukan parameter a, b, dan c langsung
    def setabc(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c
        self.m = -a / b
        self.isTwoPoint = False

    # Setter dengan membuat garis dari dua titik
    def setTwoPoints(self, p1, p2):
        # Pada representasi dua titik,
        # a = m, b = -1, c = y1 - mx1
        self.p1 = p1
        self.p2 = p2
        self.m = (p2[1] - p1[1]) / (p2[0] - p1[0])
        self.a = self.m
        self.b = -1
        self.c = p1[1] - p1[0]*(p2[1] - p1[1]) / (p2[0] - p1[0])    # c = y1 - mx1
        self.isTwoPoint = True

    # Mereturn jarak titik p0 ke garis ini
    def pointToLineDistance(self, p0):
        if(not self.isTwoPoint):
            return abs(self.a*p0[0] + self.b*p0[1] + self.c) / (self.a**2 + self.b**2)**0.5
        else:
            # Jika direpresentasikan dua titik,
            # a = m, b = -1, c = y1 - mx1; tetapi untuk menjaga kepresisian dihitung langsung tidak dengan m yang telah dihitu
            if(self.p2[0] - self.p1[0] == 0):   # Jika garis vertikal, jaraknya abs(x1 - x0)
                return abs(self.p2[0] - p0[0])
            return abs(p0[0]*(self.p2[1] - self.p1[1])/(self.p2[0] - self.p1[0]) - p0[1] + self.c) / (1 + (self.p2[1] - self.p
```

- **utils.py**, berisi fungsi perhitungan determinan

```python
# Menghitung determinan
# pl dan pr adalah titik acuan, p adalah target
def calculateDeterminan(pl, pr, p):
    return (pl[0]*pr[1] + p[0]*pl[1] + pr[0]*p[1] - p[0]*pr[1] - pr[0]*pl[1] - pl[0]*p[1])

```

- **main.ipynb**, berisi demonstrasi penggunaan bryanhull pada visualisasi data

```python
# Demonstrasi menggunakan hasil implementasi sendiri, modul bryanhull class ConvexHull
# Untuk Bonus menerima sebarang dataset ada di cell terakhir
# Bryan Amirul Husna / 13520146 / K02
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
data = datasets.load_iris()
#create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

# Visualisasi hasil ConvexHull
import matplotlib.pyplot as plt
from bryanhull import ConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
```

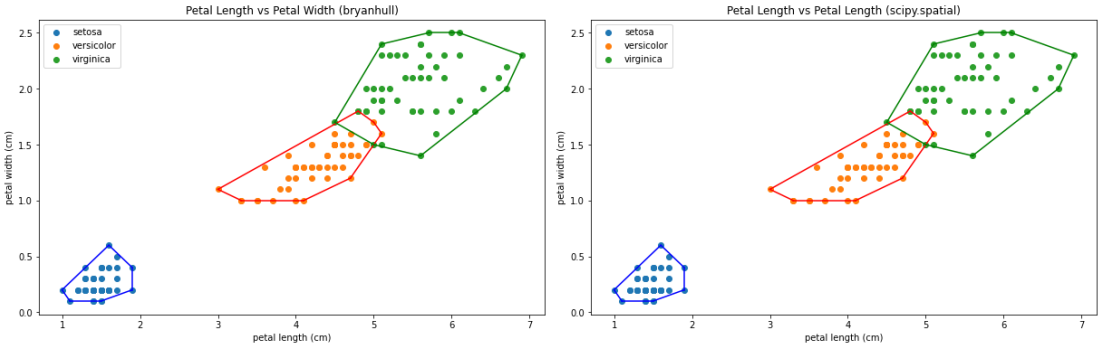
```
# Petal Length vs Petal Width
plt.title('Petal Length vs Petal Length')
plt.xlabel(data.feature_names[2])
plt.ylabel(data.feature_names[3])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[2,3]].values
    bucketlst = bucket.tolist()     # Dari numpy array perlu diubah ke Python List karena implementasi sendiri menggunakan List Python
    hull = ConvexHull(bucketlst)    # Pemanggilan kalkulasi ConvexHull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    for simplex in hull.simplices:
        plt.plot(bucket[simplex, 0], bucket[simplex, 1], colors[i])
        plt.legend()
plt.show()

# Sepal Length vs Sepal Width
plt.title('Sepal Length vs Sepal Width')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[0,1]].values

    bucketlst = bucket.tolist()     # Dari numpy array perlu diubah ke Python List karena implementasi sendiri menggunakan List Python
    hull = ConvexHull(bucketlst)    # Pemanggilan kalkulasi ConvexHull

    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    for simplex in hull.simplices:
        plt.plot(bucket[simplex, 0], bucket[simplex, 1], colors[i])
        plt.legend()
plt.show()
```

C. *Screenshot Input Output*

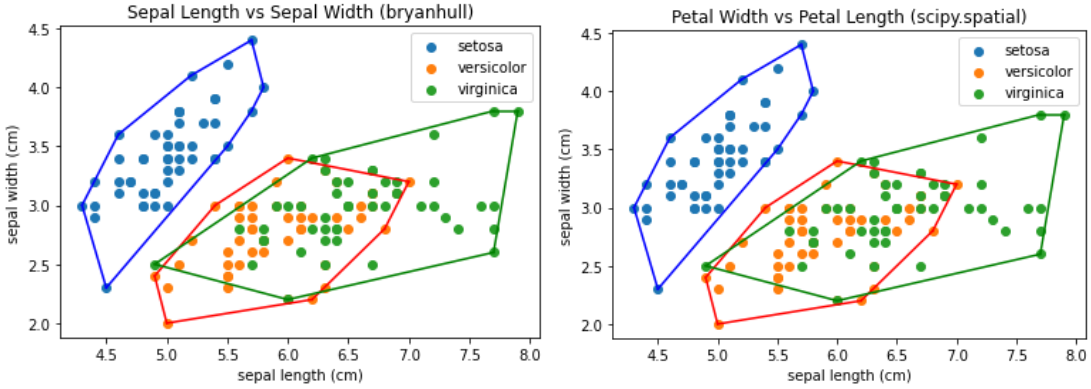
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Gambar C.1 Dataset iris



Gambar C.2 Hasil *output* dataset iris petal length vs petal width. **Kiri** menggunakan implementasi sendiri (bryanhull), **kanan** menggunakan pustaka scipy.spatial

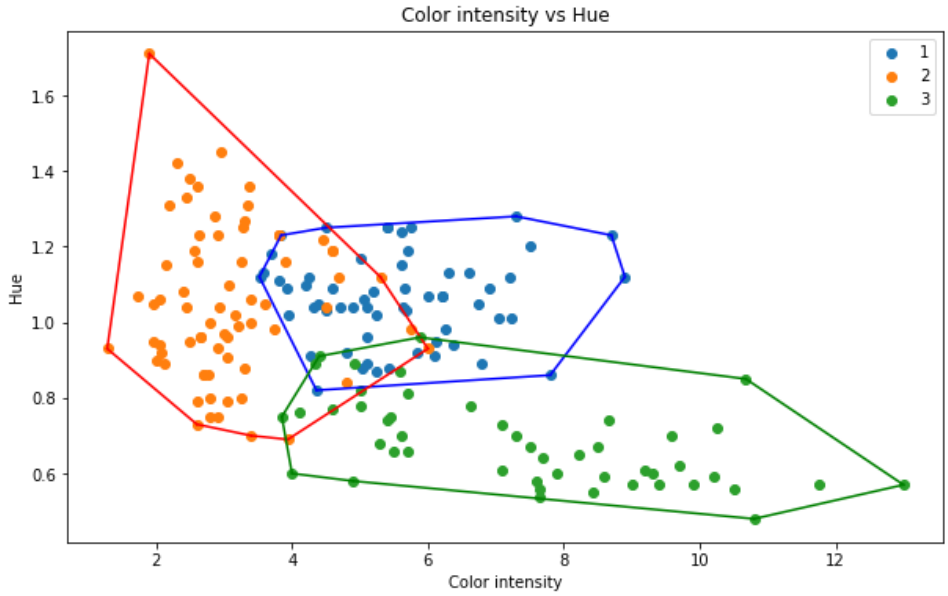


Gambar C.3 Hasil *output* dataset iris sepal length vs sepal width. **Kiri** menggunakan implementasi sendiri (bryanhull), **kanan** menggunakan pustaka scipy.spatial

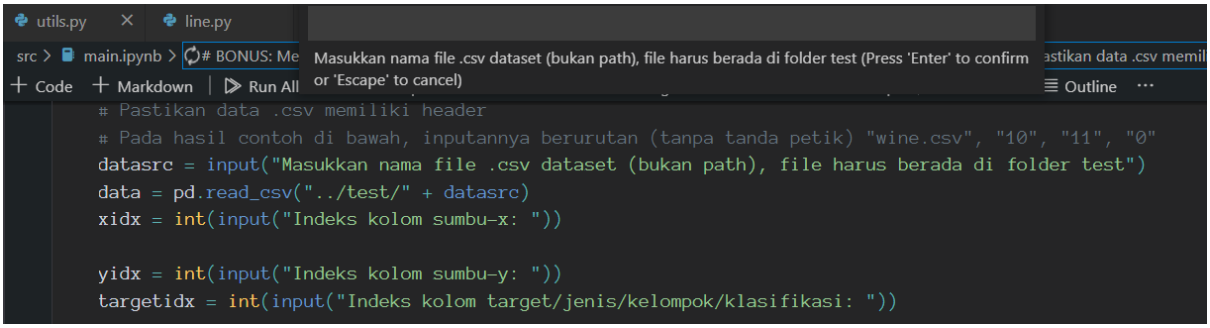
	Class	Alcohol	Malic acid	Ash	Alcalinity of Ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows × 14 columns

Gambar C.4 Dataset wine



Gambar C.4 (Bonus) menguji dengan dataset lainnya. Dataset wine Color intensity vs Hue menggunakan implementasi sendiri (bryanhull)



Gambar C.5 Menu untuk menerima input dataset lain

D. Alamat Drive Kode Program

https://github.com/bryanahusna/Stima-Convex-Hull

E. Checklist

Poin	Ya	Tidak
1. Pustaka *myConvexhull* berhasil dibuat dan tidak ada kesalahan	√	
2. *Convex hull* yang dihasilkan sudah benar	√	
3. Pustaka *myConvexHull* dapat digunakan untuk menampilkan *convex hull* setiap label dengan warna yang berbeda.	√	
4. **Bonus**: program dapat menerima input dan menuliskan output untuk dataset lainnya	√	