IF2211 – Strategi Algoritma

Tugas Kecil 2

Implementasi Convex Hull untuk Visualisasi Tes *Linear Separability Dataset* dengan Algoritma *Divide and Conquer*

Oleh:

13520118 Mohamad Daffa Argakoesoemah

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2022

1. Algoritma *Divide & Conquer* dalam Penentuan *Convex Hull*

Algoritma *Divide & Conquer* adalah salah satu algoritma yang banyak dimanfaatkan untuk berbagai keperluan. Algoritma ini terdiri dari beberapa tahap, yaitu *divide*, *conquer* (*solve*), *combine*.

Salah satu penerapan algoritma *divide & conquer*, yaitu dalam menentukan *convex hull* dari kumpulan titik. Himpunan titik pada bidang planar disebut *convex* jika untuk sembarang dua titik pada bidang tersebut (misal p dan q), seluruh segmen garis yang berakhir di p dan q berada pada himpunan tersebut. Untuk titik yang lebih dari tiga dalam himpunan S dan tidak terletak pada satu garis, maka *convex hull* berupa poligon *convex* dengan sisi berupa garis yang menghubungkan beberapa titik pada S. Langkah-langkahnya sebagai berikut:

1. Misalnya S adalah *array* yang elemennya adalah titik-titik (absis dan ordinat) yang ingin ditentukan *convex hull*-nya. S diurutkan menaik berdasarkan nilai absis yang menaik. Jika ada nilai absis yang sama, diurutkan berdasarkan nilai ordinat yang menaik.
2. Setelah S sudah teurut, dapatkan elemen pertama S (p_1) dan elemen terakhirnya (p_2) sebagai dua titik ekstrem yang akan membentuk *convex hull* untuk titik-titik pada S.
3. *Divide*: garis yang menghubungkan p_1 dan p_2 membagi S menjadi dua *array*, yaitu S1 dan S2. S1 adalah elemen S yang berada di atas atau kiri garis p_1p_2, sedangkan S2 adalah elemen S yang berada di atas atau kiri garis p_1p_2. Pemeriksaan lokasi titik untuk menentukan apakah sebuah titik masuk ke dalam S1 atau S2 menggunakan penentuan determinan seperti pada Gambar 1. Titik (x_3, y_3) berada di sebelah kiri garis yang dibentuk oleh (x_1,y_1) dan (x_2,y_2) jika nilai determinan positif. Sebaliknya, determinan bernilai negatif jika (x_3, y_3) berada di kanannya. Determinan bernilai nol jika (x_3, y_3) tepat berada di garis tersebut dan titik tersebut diabaikan dari pemeriksaan lebih lanjut.

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

Gambar 1: Penentuan determinan

4. *Conquer*: S1 dapat membentuk *convex hull* bagian atas, sedangkan S2 dapat membentuk bagian bawahnya. Untuk masing-masing S1 dan S2, selanjutnya terdapat dua kemungkinan yang sama: Misalnya kemungkinan pada S1:
   a. Jika tidak ditemukan titik pada S1, p_1 dan p_2 menjadi titik-titik pembentuk *convex hull* sehingga dapat dimasukkan ke dalam *array* solusi, misalnya T. Kemungkinan ini adalah basis dalam prosedur *convex hull* sehingga prosedur selesai jika memasuki kemungkinan ini.
   b. Jika masih terdapat titik pada S1, dipilih sebuah titik yang memiliki jarak terjauh dari garis p_1p_2 (misalnya p_3). Semua titik yang berada di dalam segitiga $p_1p_2p_3$ diabaikan dari pemeriksaan lebih lanjut karena tidak akan membentuk *convex hull*.
5. Jika belum mencapai basis atau mendapatkan kemungkinan bagian 4b, bagi S1 ke dalam dua *array*, yaitu *array* $S_{1,1}$ yang elemennya adalah kumpulan titik yang berada di sebelah kiri garis p_1p_3 dan *array* $S_{1,2}$ yang elemennya adalah kumpulan titik yang berada di sebelah kanan garis p_1p_3. Lakukan kembali langkah 4 (rekurens).
6. Langkah 4 dan 5 dilakukan juga untuk bagian S2.
7. *Combine*: Setelah mencapai bagian 4a atau basis, gabungkan *array* solusi dari S1 dan S2 sehingga didapatkan kumpulan lengkap titik-titik yang membentuk *convex hull*.

2. Kode Program

Berikut adalah *screenshot* algoritma pencarian *convex hull* serta algoritma untuk melakukan *plotting* menjadi grafik:

```python
def checkPointPosition(p1, p2, p3):
    '''
        p1, p2, p3 adalah matriks dgn 2 elemen (elemen pertama absis, elemen kedua oordinat)
        Mengembalikan 1 jika titik p3 berada di sebelah
        kiri atau atas garis yg dibentuk oleh p1 dan p2,
        -1 jika di bawahnya, 0 jika tepat pada garis
    '''

    det = (p3[1] - p1[1]) * (p2[0] - p1[0]) - (p2[1] - p1[1]) * (p3[0] - p1[0])
    if (det > 0):
        return 1
    elif (det < 0):
        return -1
    else:
        return 0

def distanceBetweenLineAndPoint(p1, p2, p3):
    '''
        p1, p2, p3 adalah matriks dgn 2 elemen (elemen pertama absis, elemen kedua oordinat)
        Mengembalikan nilai yg sebanding dgn jarak antara titik p3
        dgn garis yg dibentuk oleh titik p1 dan p2. Pembilang dlm
        rumus asli perhitungan ini diabaikan sehingga yg dihitung hanya
        penyebutnya.
    '''
    return abs((p2[0]-p1[0])*(p1[1]-p3[1])-(p1[0]-p3[0])*(p2[1]-p1[1]))
```

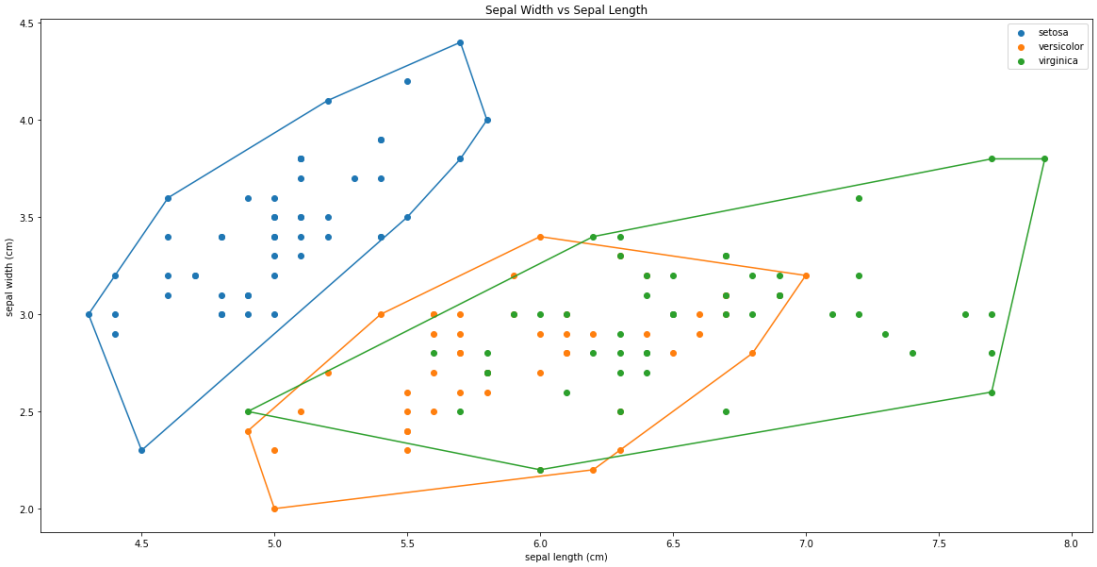
```python
def getLeftSide(M, p1, p2):
    '''
        Mendapatkan titik2 yang berada di kiri garis
        yg dibentuk oleh p1 dan p2
    '''
    leftSide = []
    for point in M:
        if checkPointPosition(p1,p2,point) == 1:
            leftSide.append(point)
    return leftSide

def getRightSide(M, p1, p2):
    '''
        Mendapatkan titik2 yang berada di kanan garis
        yg dibentuk oleh p1 dan p2
    '''
    rightSide = []
    for point in M:
        if checkPointPosition(p1,p2,point) == -1:
            rightSide.append(point)
    return rightSide
```

```python
def convexHull(M, p1, p2, points, side):
    '''
        Prosedur rekursif untuk mencari titik2 yg membentuk convex hull
        Menerima masukan array of titik M, titik p1 dan p2 sbg titik ekstrem,
        Titik2 hasil disimpan dalam points, side menentukan bagian yg
        dihitung (bernilai 1 jika sisi kiri atau -1 jika sisi kanan)

    '''
    max = 0
    max_point = [-999,-999]
    for point in M:
        dist = distanceBetweenLineAndPoint(p1, p2, point)
        if (dist > max):
            max = dist
            max_point = point

    if (max == 0): # tidak ditemukan titik lagi
        points.append(p1)
        points.append(p2)
        return

    # Pembagian titik2 ke dalam dua sisi
    if side == 1:
        side1 = getLeftSide(M, p1, max_point)
        side2 = getLeftSide(M, max_point, p2)
    else:
        side1 = getRightSide(M, p1, max_point)
        side2 = getRightSide(M, max_point, p2)
    convexHull(side1,p1,max_point,points,side)
    convexHull(side2,max_point,p2,points,side)

def myConvexHull(M):
    '''
        Menerima masukan array of titik yg ingin dicari convex hull-nya.
        Mengembalikan array of titik yg membentuk convex hull dari array of titik M
    '''
    M.sort(key=lambda k: [k[0], k[1]])
    points = []
    leftSide = getLeftSide(M, M[0], M[len(M)-1])
    rightSide = getRightSide(M, M[0], M[len(M)-1])

    # penerapan Divide & Conquer menentukan titik2 pada sisi atas (kiri) atau bawah (kanan)
    convexHull(leftSide, M[0], M[len(M)-1], points, 1) # sisi atas (kiri)
    convexHull(rightSide, M[0], M[len(M)-1], points, -1) # sisi bawah (kanan)
    points.sort(key=lambda k: [k[0], k[1]])

    # Algoritma penyusunan dan pengurutan ulang array of titik hasil convex hull untuk diplot
    p = points[0]
    q = points[len(points)-1]
    leftSide = getLeftSide(points, p, q) # titik2 yg berada di kiri (atau atas) garis yg dibentuk oleh p dan q
    rightSide = getRightSide(points, p, q) # titik2 yg berada di kanan (atau bawah) garis yg dibentuk oleh p dan q
    rightSide = rightSide + [p, q]
    rightSide.sort(key=lambda k: [k[0], k[1]])
    leftSide.sort(key=lambda k: [k[0], k[1]], reverse=True)
    result = rightSide + leftSide
    result.append(result[0]) # penambahan dgn elemen pertama untuk kebutuhan plotting
    return result
```

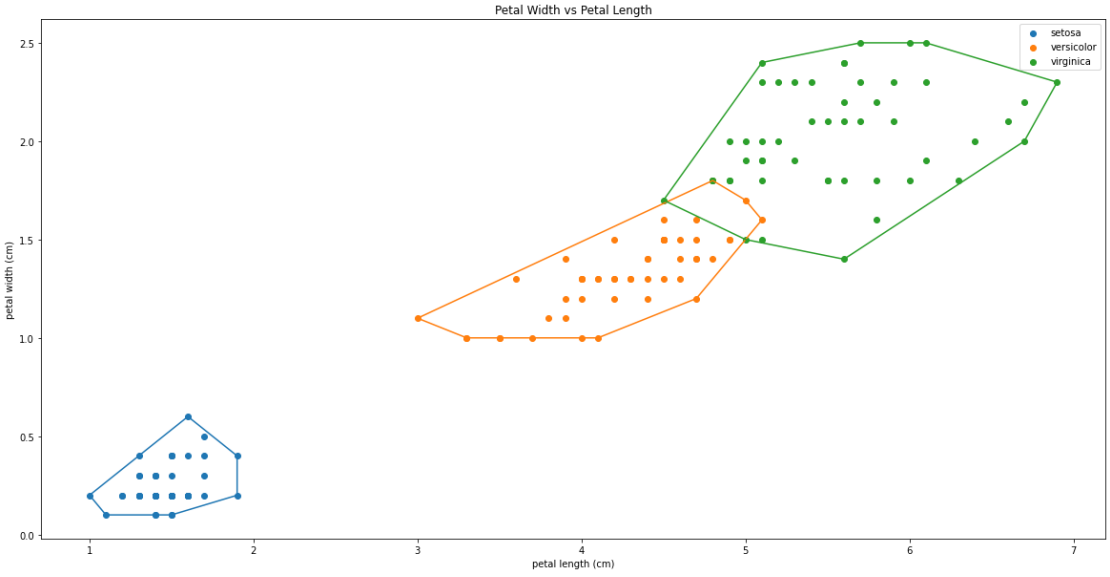
3. *Screenshot Input-Output* Program

3.1 Dataset Iris

Output pasangan atribut sepal-width dan sepal-length:

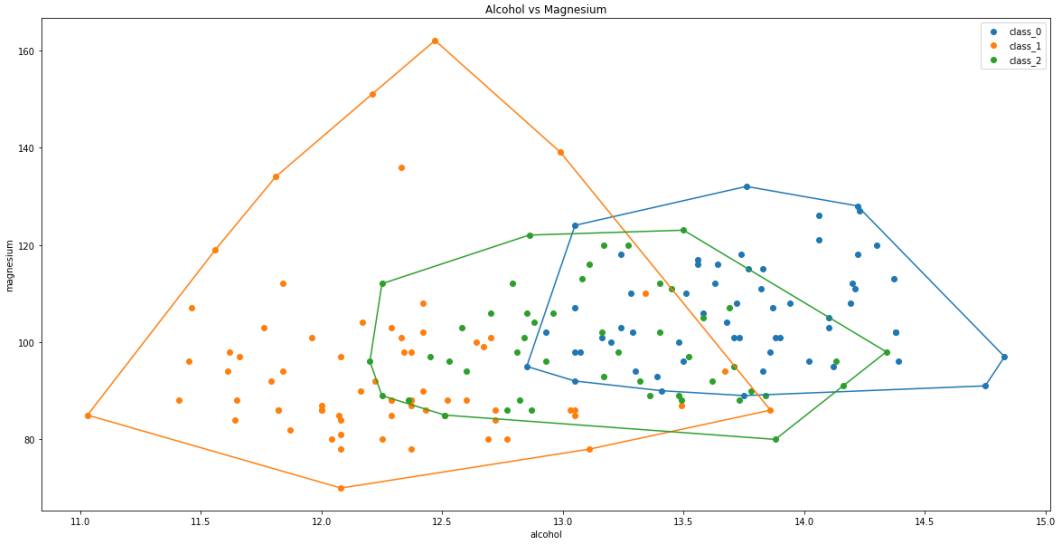


Output pasangan atribut petal-width dan petal-length:

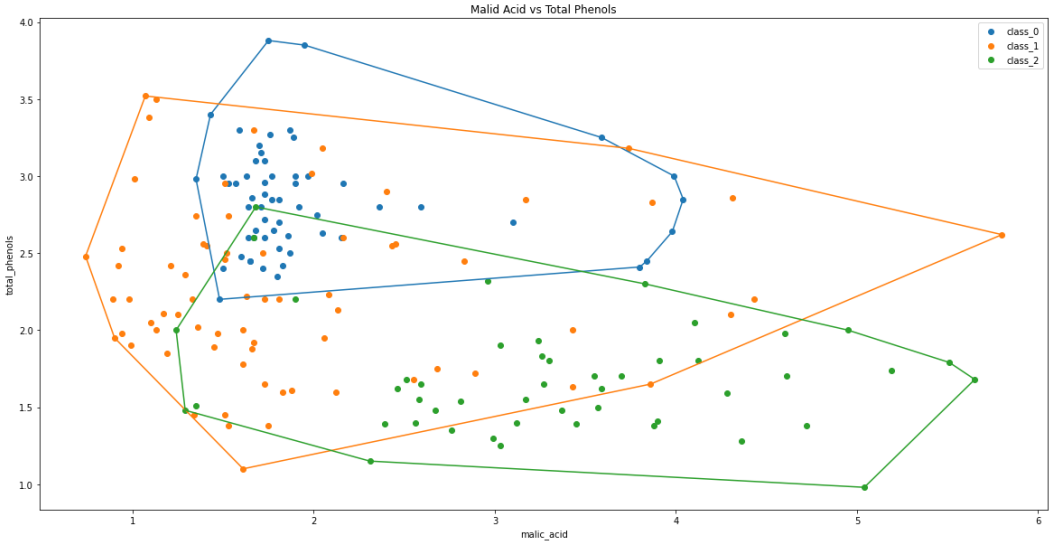


3.2 Dataset Wine

Output pasangan atribut alcohol dan magnesium:

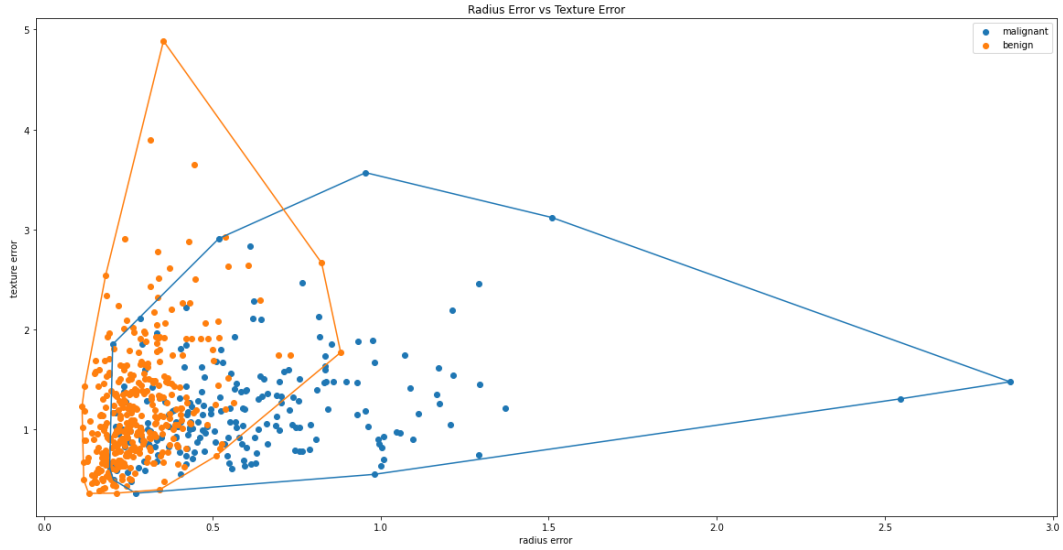


Output pasangan atribut malid_acid dan total_phenols:

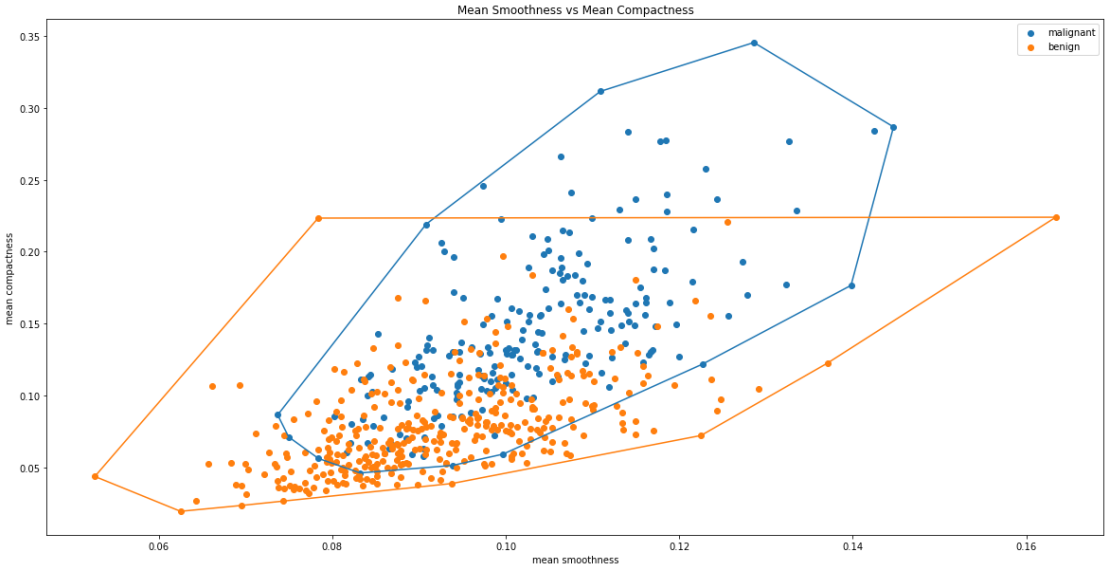


3.3 Dataset Digits

Output pasangan atribut radius_error dan texture_error:



Output pasangan atribut mean_smoothness dan mean_smoothness:



4. Alamat kode program

https://github.com/daffarg/Tucil2-Stima

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