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 1.1. 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 1.1: 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 penyusunan ulang larik hasil yang berisi titik-titik pembentuk *convex hull* agar siap di-*plot* menjadi grafik:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets

# Algoritma Pencarian Convex Hull

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): # di sebelah kiri garis
        return 1
    elif (det < 0): # di sebelah kanan garis
        return -1
    else: # tepat pada garis atau det = 0
        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: # jika hasil determinan lebih dari nol (fungsi mengembalikan 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: # jika hasil determinan kurang dari nol (fungsi mengembalikan -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: # pencarian titik dgn jarak terjauh dari garis
        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: # sedang melakukan pencarian titik yg berada di kiri (atas) garis
        side1 = getLeftSide(M, p1, max_point)
        side2 = getLeftSide(M, max_point, p2)
    else: # sedang melakukan pencarian titik yg berada di kanan (bawah) garis pq awal
        side1 = getRightSide(M, p1, max_point)
        side2 = getRightSide(M, max_point, p2)

    # rekurens untuk pada dua sisi yang terbentuk akibat pembagian
    convexHull(side1,p1,max_point,points,side)
    convexHull(side2,max_point,p2,points,side)
```

```python
def myConvexHull(M):
    '''
        Menerima masukan array of titik yg ingin dicari convex hull-nya.
        Mengembalikan array of titik yang siap di-plot sbg 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 digabung dengan titik yg membentuk garis pq
    rightSide.sort(key=lambda k: [k[0], k[1]]) # rightSide diurutkan menaik
    leftSide.sort(key=lambda k: [k[0], k[1]], reverse=True) # leftSide diurutkan menurun
    result = rightSide + leftSide # rightSide dan leftSide digabungkan
    result.append(result[0]) # penambahan dgn elemen pertama untuk kebutuhan plotting
    return result
```

3. *Screenshot Input-Output* Program

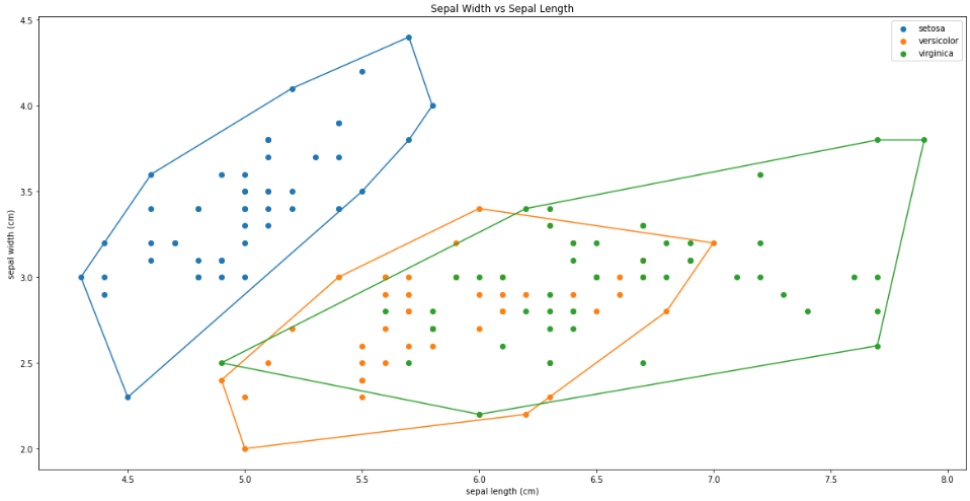
3.1 *Dataset* Iris

Potongan *dataset* Iris:

	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

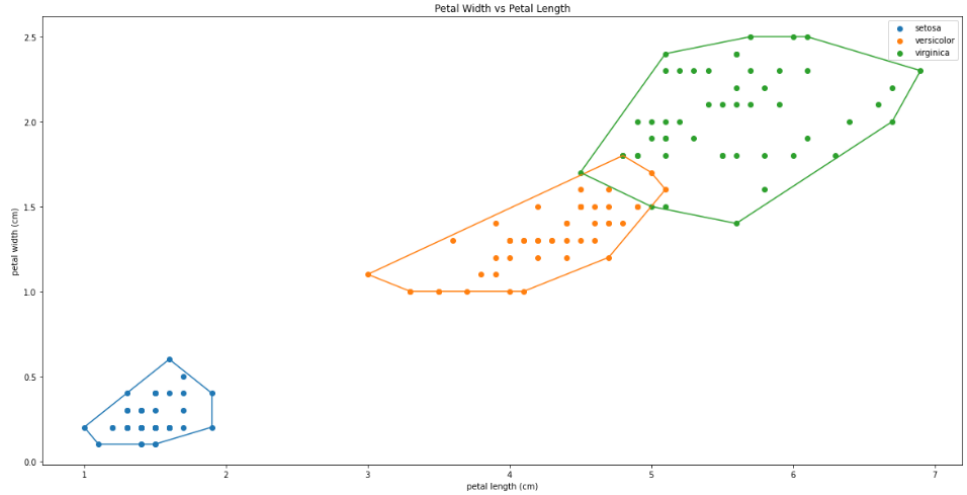
Gambar 3.1.1: Potongan *dataset* Iris

Output pasangan atribut sepal-width dan sepal-length:



Gambar 3.1.2: *Output* pasangan atribut sepal-width dan sepal-length

Output pasangan atribut petal-width dan petal-length:



Gambar 3.1.3: *Output* pasangan atribut petal-width dan petal-length

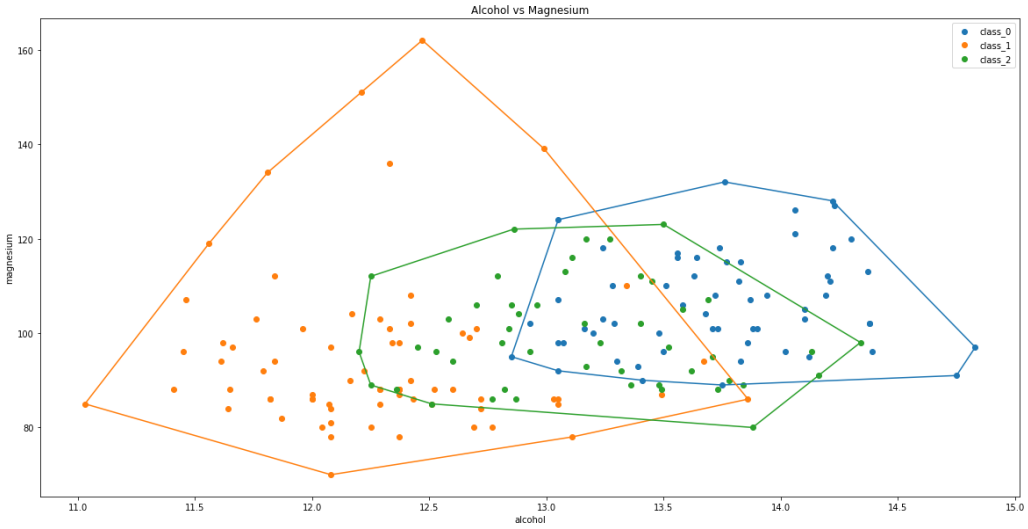
3.2 *Dataset* Wine

Potongan *dataset* Wine:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	Target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

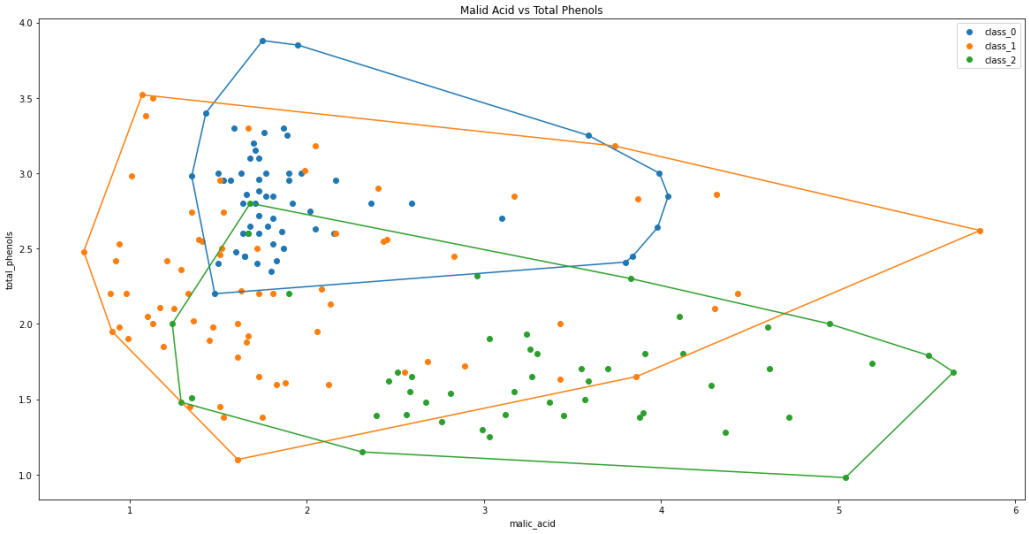
Gambar 3.2.1: Potongan *dataset* Wine

Output pasangan atribut alcohol dan magnesium:



Gambar 3.2.2: *Output* pasangan atribut alcohol dan magnesium

Output pasangan atribut malid_acid dan total_phenols:



Gambar 3.2.3: *Output* pasangan atribut malid_acid dan total_phenols

3.3 *Dataset* Breast Cancer

Potongan *dataset* Breast Cancer:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.0950	0.9053	8.589
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.585
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.1560	3.445
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	5.438

Gambar 3.3.1: Potongan *dataset* Breast Cancer bagian 1

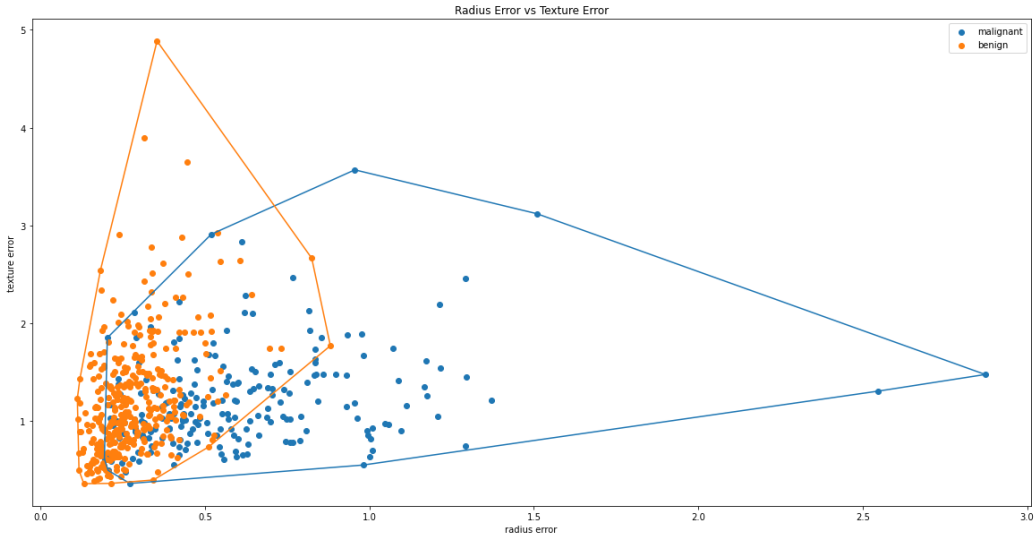
area error	smoothness error	compactness error	concavity error	concave points error	symmetry error	fractal dimension error	worst radius	worst texture	worst perimeter	worst area	worst smoothness
153.40	0.006399	0.04904	0.05373	0.01587	0.03003	0.006193	25.38	17.33	184.60	2019.0	0.1622
74.08	0.005225	0.01308	0.01860	0.01340	0.01389	0.003532	24.99	23.41	158.80	1956.0	0.1238
94.03	0.006150	0.04006	0.03832	0.02058	0.02250	0.004571	23.57	25.53	152.50	1709.0	0.1444
27.23	0.009110	0.07458	0.05661	0.01867	0.05963	0.009208	14.91	26.50	98.87	567.7	0.2098
94.44	0.011490	0.02461	0.05688	0.01885	0.01756	0.005115	22.54	16.67	152.20	1575.0	0.1374

Gambar 3.3.2: Potongan *dataset* Breast Cancer bagian 2

worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	Target
0.6656	0.7119	0.2654	0.4601	0.11890	0
0.1866	0.2416	0.1860	0.2750	0.08902	0
0.4245	0.4504	0.2430	0.3613	0.08758	0
0.8663	0.6869	0.2575	0.6638	0.17300	0
0.2050	0.4000	0.1625	0.2364	0.07678	0

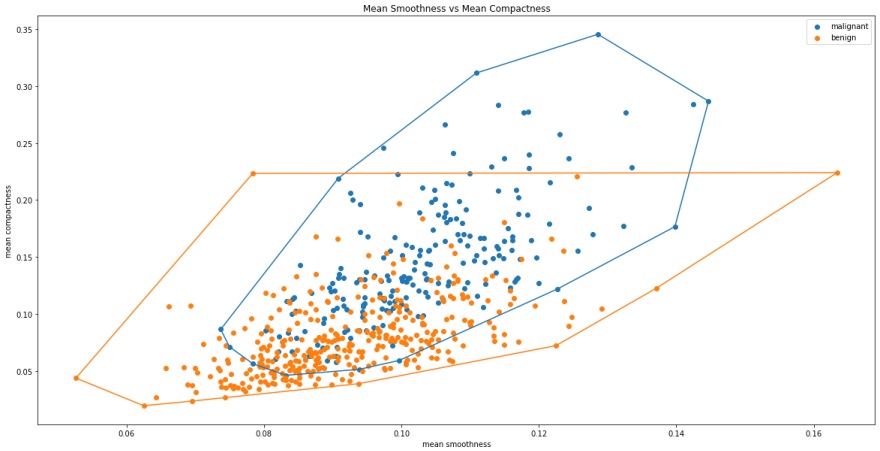
Gambar 3.3.3: Potongan *dataset* Breast Cancer bagian 3

Output pasangan atribut radius_error dan texture_error:



Gambar 3.3.4: *Output* pasangan atribut radius_error dan texture_error

Output pasangan atribut mean_smoothness dan mean_compactness:



Gambar 3.3.5: *Output* pasangan atribut mean_smoothness dan mean_compactness

3.4 Dataset Digits

Potongan *dataset* Digits:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	pixel_1_2	pixel_1_3
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	13.0	15.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	0.0	11.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	3.0	16.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	13.0	6.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0

Gambar 3.4.1: Potongan *dataset* Digits bagian 1

pixel_1_4	pixel_1_5	pixel_1_6	pixel_1_7	pixel_2_0	pixel_2_1	pixel_2_2	pixel_2_3	pixel_2_4	pixel_2_5	pixel_2_6	pixel_2_7	pixel_3_0
10.0	15.0	5.0	0.0	0.0	3.0	15.0	2.0	0.0	11.0	8.0	0.0	0.0
16.0	9.0	0.0	0.0	0.0	0.0	3.0	15.0	16.0	6.0	0.0	0.0	0.0
15.0	14.0	0.0	0.0	0.0	0.0	8.0	13.0	8.0	16.0	0.0	0.0	0.0
15.0	4.0	0.0	0.0	0.0	2.0	1.0	13.0	13.0	0.0	0.0	0.0	0.0
8.0	0.0	0.0	0.0	0.0	0.0	1.0	13.0	6.0	2.0	2.0	0.0	0.0

Gambar 3.4.2: Potongan *dataset* Digits bagian 2

pixel_3_1	pixel_3_2	pixel_3_3	pixel_3_4	pixel_3_5	pixel_3_6	pixel_3_7	pixel_4_0	pixel_4_1	pixel_4_2	pixel_4_3	pixel_4_4	pixel_4_5
4.0	12.0	0.0	0.0	8.0	8.0	0.0	0.0	5.0	8.0	0.0	0.0	9.0
7.0	15.0	16.0	16.0	2.0	0.0	0.0	0.0	0.0	1.0	16.0	16.0	3.0
0.0	1.0	6.0	15.0	11.0	0.0	0.0	0.0	1.0	8.0	13.0	15.0	1.0
0.0	2.0	15.0	11.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	12.0	12.0
0.0	7.0	15.0	0.0	9.0	8.0	0.0	0.0	5.0	16.0	10.0	0.0	16.0

Gambar 3.4.3: Potongan *dataset* Digits bagian 3

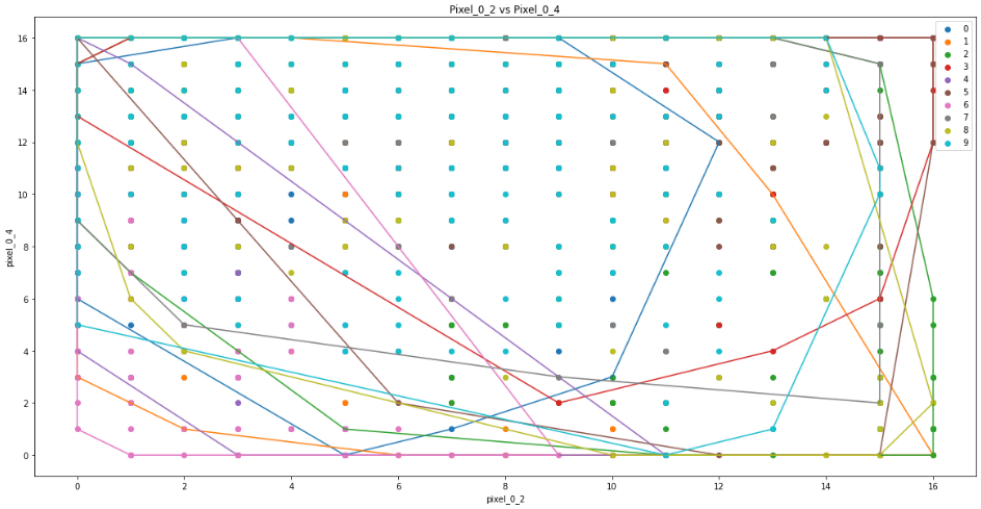
pixel_4_6	pixel_4_7	pixel_5_0	pixel_5_1	pixel_5_2	pixel_5_3	pixel_5_4	pixel_5_5	pixel_5_6	pixel_5_7	pixel_6_0	pixel_6_1	pixel_6_2
8.0	0.0	0.0	4.0	11.0	0.0	1.0	12.0	7.0	0.0	0.0	2.0	14.0
0.0	0.0	0.0	0.0	1.0	16.0	16.0	6.0	0.0	0.0	0.0	0.0	1.0
0.0	0.0	0.0	9.0	16.0	16.0	5.0	0.0	0.0	0.0	0.0	3.0	13.0
1.0	0.0	0.0	0.0	0.0	0.0	1.0	10.0	8.0	0.0	0.0	0.0	8.0
6.0	0.0	0.0	4.0	15.0	16.0	13.0	16.0	1.0	0.0	0.0	0.0	0.0

Gambar 3.4.4: Potongan *dataset* Digits bagian 4

pixel_6_3	pixel_6_4	pixel_6_5	pixel_6_6	pixel_6_7	pixel_7_0	pixel_7_1	pixel_7_2	pixel_7_3	pixel_7_4	pixel_7_5	pixel_7_6	pixel_7_7	Target
5.0	10.0	12.0	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0
16.0	16.0	6.0	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1
16.0	16.0	11.0	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
4.0	5.0	14.0	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
3.0	15.0	10.0	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4

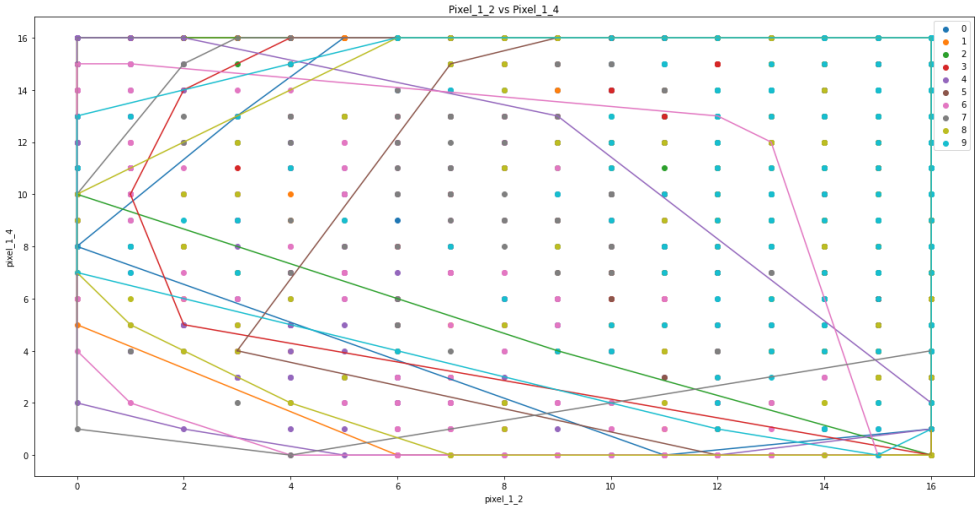
Gambar 3.4.5: Potongan *dataset* Digits bagian 5

Output pasangan atribut pixel_0_2 dan pixel_0_4:



Gambar 3.4.6: *Output* pasangan atribut pixel_0_2 dan pixel_0_4

Output pasangan atribut pixel_1_2 dan pixel_1_4:



Gambar 3.4.7: *Output* pasangan atribut pixel_1_2 dan pixel_1_4

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	