# LAPORAN TUGAS KECIL 2

# IF2211 STRATEGI ALGORITMA


Disusun oleh:

Hansel Valentino Tanoto

13520046

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG


2022

# DAFTAR ISI

# A. ALGORITMA *DIVIDE AND CONQUER*

Algoritma *Divide and Conquer* merupakan strategi pemecahan masalah yang besar dengan membagi-bagi permasalahan tersebut menjadi beberapa sub-persoalan sehingga lebih mudah diselesaikan. Secara umum, algoritma *divide and conquer* terdiri dari 3 proses utama, yaitu:

1. *Divide*, yaitu membagi persoalan menjadi beberapa sub-persoalan yang memiliki kemiripan karakteristik dengan persoalan awal tetapi memiliki lingkup / ukuran yang lebih kecil.
2. Conquer, yaitu menyelesaikan masing-masing sub-persoalan secara rekursif hingga sub-persoalan menjadi cukup kecil dan dapat diselesaikan secara langsung.
3. *Combine*, yaitu menggabungkan solusi dari masing-masing sub-persoalan sehingga membentuk solusi untuk persoalan awal.

Skema umum dari algoritma *divide and conquer* adalah berbentuk fungsi / prosedur yang bersifat refursif, yaitu sebagai berikut:

```
procedure DivideAndConquer(input P: problem, n: integer)
{Menyelesaikan persoalan P berukuran n dengan algoritma divide and
 conquer}
Deklarasi
    r: integer
    n₀: integer
Algoritma
    if (n ≤ n₀) then     {Basis}
        Solve P
    else                 {Rekurens}
        Divide menjadi {P₁, P₂, …, Pᵣ}
        for masing-masing {P₁, P₂, …, Pᵣ} do
            DivideAndConquer(Pᵢ,nᵢ)
        endfor
        Combine solusi P₁, P₂, …, Pᵣ
    endif
```

Variabel $n_0$ pada skema di atas merupakan batas yang menandakan suatu persoalan sudah cukup kecil sehingga tidak perlu dibagi lagi dan dapat langsung diselesaikan. Kompleksitas waktu dari algoritma *divide and conquer* secara umum adalah:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) + \ldots + T(n_r) + f(n) & , n > n_0 \end{cases} \quad \textbf{(1)}$$

dengan $T(n_i)$ menyatakan kompleksitas waktu untuk menyelesaikan sub-persoalan ke-$i$, $g(n)$ menyatakan kompleksitas waktu untuk menyelesaikan basis persoalan (persoalan yang sudah cukup kecil), dan $f(n)$ menyatakan kompleksitas waktu untuk menggabungkan solusi masing-masing sub-persoalan.

Pada tugas kali ini, algoritma *divide and conquer* tersebut akan diterapkan untuk menghasilkan sebuah *convex hull* dari kumpulan data 2 dimensi yang dapat dianggap juga sebagai kumpulan titik 2 dimensi. Convex hull adalah sebuah poligon yang tersusun dari subset himpunan titik sedemikian sehingga tidak ada titik dari himpunan awal yang berada di luar poligon tersebut dan jika digambarkan garis yang menghubungkan sembarang titik pada himpunan titik awal, tidak ada garis yang memotong garis batas luar dari poligon tersebut.

Cara kerja algoritma *divide and conquer* yang digunakan adalah sebagai berikut. Pertama-tama, akan dilakukan pengecekan jumlah titik yang menjadi *input* program terlebih dahulu. Jika titik awal berjumlah 0 atau 1, maka tidak ada *convex hull* yang terbentuk dan jika jumlah titik awal = 2, maka *convex hull* yang terbentuk hanya berupa garis lurus. Sedangkan, jika jumlah titik ≥ 3, maka himpunan titik tersebut akan dibagi menjadi 2 terlebih dahulu (*divide*). Pembagian ini dilakukan dengan terlebih dahulu mencari titik dengan absis terkecil (titik paling kiri, missal $P_1$) dan absis terbesar (titik paling kanan, missal $P_2$). Garis yang menghubungkan kedua titik tersebut akan menjadi batas untuk pembagian himpunan titik awal. Untuk menentukan sebuah titik berada di sisi kiri/atas atau kanan/bawah dari garis, dapat digunakan perhitungan determinan sebagai berikut:

$$det = \begin{vmatrix} x_{P_1} & y_{P_2} & 1 \\ x_{P_2} & y_{P_2} & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_{P_1}y_{P_2} + x_3y_{P_1} + x_{P_2}y_3 - x_3y_{P_2} - x_{P_2}y_{P_1} - x_{P_1}y_3 \quad \textbf{(2)}$$

Jika $det$ bernilai positif, maka titik $(x_3, y_3)$ berada di sebelah kiri/atas dari garis $(P_1, P_2)$ dan berlaku sebaliknya jika $det$ bernilai negatif. Jadi, sekarang sudah terbentuk 2 buah subset titik (misal $S_1$ dan $S_2$) yang masing-masing akan diterapkan algoritma *divide and conquer*. Untuk setiap subset ($S_1$ dan $S_2$), jika tidak terdapat titik lagi, maka titik pembentuk garis pembatas tadi merupakan anggota dari garis pembentuk *convex hull* sehingga kedua titik tersebut akan dimasukkan ke dalam himpunan solusi (*conquer*). Namun, untuk setiap subset, apabila masih terdapat titik, akan dicari titik dengan jarak terjauh dari garis pembatas $(P_1, P_2)$, misalkan $P_{max}$. Jarak antara suatu titik $(x_3, y_3)$ dengan suatu garis $(ax + by + c = 0)$ dapat dicari dengan rumus:

$$distance = \frac{|ax_3 + by_3 + c|}{\sqrt{a^2 + b^2}} \quad \textbf{(3)}$$

Karena hanya ingin dicari jarak titik terjauh dan bagian penyebut dari persamaan (3) di atas bernilai konstan untuk garis yang sama, maka nilai penyebut tersebut dapat diabaikan dalam implementasi algoritma. Jadi dalam implementasi algoritma yang saya buat, titik terjauh dapat dicari dengan hanya memaksimalkan nilai pembilang yaitu $|ax_3 + by_3 + c|$.

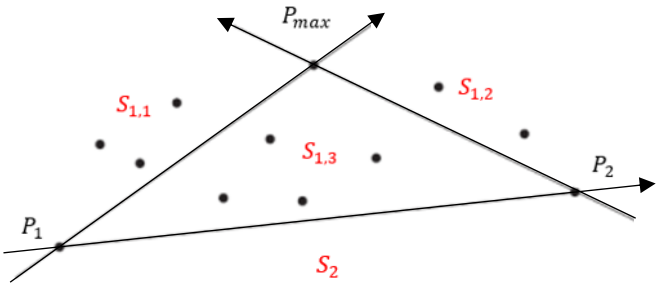
Mencari persamaan garis jika diketahui 2 buah titik dapat dilakukan menggunakan rumus:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \iff (y - y_1)(x_2 - x_1) - (x - x_1)(x_2 - x_1) = 0 \quad \textbf{(4)}$$

Jadi dengan mensubstitusi persamaan (4) ke persamaan (3), untuk mencari jarak titik maksimum hanya perlu dicari titik yang menghasilkan nilai terbesar jika dihitung menggunakan rumus berikut ini:

$$distance = \left|(y_3 - y_{P_1})(x_{P_2} - x_{P_1}) - (x_3 - x_{P_1})(y_{P_2} - y_{P_1})\right| \quad \textbf{(5)}$$

Kemudian akan dibentuk garis pembatas kembali yaitu garis $(P_1, P_{max})$ dan $(P_2, P_{max})$ sehingga terbentuk 3 buah daerah baru yaitu $S_{1,1}$, $S_{1,2}$, dan $S_{1,3}$ seperti yang terlihat pada gambar di bawah ini.




***Gambar 1 Ilustrasi algoritma divide and conquer pada pembentukan convex hull***

Himpunan titik pada daerah $S_{1,3}$ dapat diabaikan untuk pemeriksaan lanjut karena titik-titik tersebut berada di dalam *convex hull*. Sedangkan, untuk himpunan titik pada daerah $S_{1,1}$ dan $S_{1,2}$ akan dilakukan algoritma *divide and conquer* lagi secara rekursif hingga tidak ada lagi titik yang berada di luar *convex hull*. Setelah menyelesaikan semua sub-persoalan, solusi-solusi tersebut akan digabungkan (*combine*) sehingga menghasilkan *convex hull* yang utuh.

Jika diperhatikan, algoritma *divide and conquer* yang digunakan pada pembentukan *convex hull* ini selalu membagi persoalan menjadi 2 sub-persoalan. Proses penggabungan (*combine*) memiliki kompleksitas $O(n)$ dalam notasi Big-O karena program hanya perlu melakukan *merging* 2 buah array dari sub-partisi. Sedangkan proses penyelesaian (*solve*) pada basis memiliki kompleksitas $O(1)$ dalam notasi Big-O karena program hanya perlu menambahkan sepasang titik ke himpunan solusi. Jika persebaran titik dianggap merata, maka setiap kali dilakukan pembagian, jumlah titik di setiap partisi akan berjumlah kira-kira setengah dari jumlah titik pada tahapan sebelumnya sehingga fungsi kompleksitasnya adalah:

$$T(n) = \begin{cases} k & ,n \leq n_0 \\ 2T(n/2) + cn & ,n > n_0 \end{cases} \quad \textbf{(6)}$$

Dengan menggunakan Teorema Master, dapat diperoleh kompleksitas algoritma dari algoritma ini dalam *average case scenario*, yaitu $T(n) = O(n^1.logn) = O(n\ logn)$.

Daftar fungsi/prosedur yang terdapat pada program ini yaitu:

1. *File* `myConvexHull.py`
   a. `convexHull()`, fungsi ini akan merupakan fungsi utama pada Pustaka *myConvexHull* yang akan menjalankan algoritma sesuai yang sudah dijelaskan di atas dari awal hingga pembagian himpunan titik menjadi 2 *subset* yaitu $S_1$ dan $S_2$. Setelahnya, fungsi ini akan memanggil fungsi `recursiveConvexHull()` yang akan menangani masing-masing *subset* secara rekursif. Di akhir program akan mengembalikan nilai berupa penggabungan solusi dari kedua *subset*.
   b. `recursiveConvexHull()`, fungsi ini akan menghasilkan *convex hull* secara rekursif yang dimulai dengan pencarian titik dengan jarak terjauh dari garis partisi menggunakan algoritma yang telah dijelaskan di atas. Jika himpunan titik merupakan hasil partisi bagian kiri dari proses sebelumnya (seperti pada gambar 1), maka pada partisi kali ini, hanya partisi kiri dari garis $(P_1, P_{max})$ yaitu $S_{1,1}$ dan partisi kanan dari garis $(P_2, P_{max})$ yaitu $S_{1,2}$ saja yang akan diterapkan fungsi `recursiveConvexHull()` kembali. Hal ini dapat dilihat pada kode program bagian percabangan (`if (side == LEFT):`) yang hanya memproses bagian partisi kiri (`leftPointsIdx1`) garis pertama $(P_1, P_{max})$ dan bagian partisi kanan (`rightPointsIdx2`) dari garis kedua $(P_2, P_{max})$. Sedangkan jika himpunan titik yang ingin dipartisi adalah hasil partisi bagian kanan dari proses sebelumnya, yang diproses menggunakan fungsi

`recursiveConvexHull()` kembali hanya bagian partisi kanan (`rightPointsIdx1`) garis pertama $(P_1, P_{max})$ dan bagian partisi kiri (`leftPointsIdx2`) dari garis kedua $(P_2, P_{max})$.

c. `splitPoints()`, fungsi ini akan membagi himpunan titik menjadi 2 subset berdasarkan persamaan (2). Kompleksitas waktu dari fungsi ini adalah $O(n)$, dengan $n$ menyatakan jumlah titik yang menjadi parameter fungsi.
d. `MinMaxAbsisPoint()`, fungsi ini akan mengembalikan titik dengan absis terkecil dan absis terbesar pada himpunan titik dengan melakukan mekanisme traversal pada semua titik di himpunan. Kompleksitas waktu dari fungsi ini adalah $O(n)$, dengan $n$ menyatakan jumlah titik yang menjadi parameter fungsi.
e. `distanceFromLine()`, fungsi ini akan mengembalikan nilai jarak dari suatu titik terhadap suatu garis berdasarkan persamaan (5). Kompleksitas waktu dari fungsi ini adalah $O(1)$.

2. *File* `Input/Output.py`
   a. `chooseDataset()`, fungsi ini akan menampilkan *list* dataset yang dapat digunakan dan meminta *input* pilihan dataset dari *user*.
   b. `chooseAttributes()`, fungsi ini akan menampilkan *list* stribut yang terdapat pada dataset yang telah dipilih sebelumnya dan meminta *input* pilihan pasangan atribut dari *user*.
   c. `exitConfirm()`, fungsi ini akan meminta *input* dari *user* berupa konfirmasi untuk melanjutkan program (*input* = Y atau y) atau keluar dari program (*input* = N atau n).

3. *File* `driver_Iris.py` dan `driver_General.py`
   Pada *file* ini, tidak terdapat pendefinisian fungsi/prosedur melainkan hanya algoritma untuk menerima dan memroses *input* dengan pemanggilan fungsi/prosedur dari modul lain, serta penampilan *output*. Algoritma pada *file* ini kurang lebih sama dengan kode program yang terdapat pada *file* spesifikasi tugas.

# B. KODE PROGRAM

Pustaka *myConvexHull* ini dibuat menggunakan bahasa pemrograman Python. Berikut adalah kode program lengkapnya untuk semua modul baik pustaka *convex hull* itu sendiri maupun *main driver*-nya:

1. *File* `myConvexHull.py`

File ini merupakan *library myConvexHull* yang mengandung kelas dengan nama yang sama. Kelas `myConvexHull` memiliki atribut `allPoints` (himpunan semua titik / pasangan nilai atribut yang ingin dicari *convex hull*-nya), `countPoints` (jumlah / banyaknya titik-titik tersebut), allPointsIdx (himpunan indeks masing-masing titik pada himpunan semua titik) dan `simplices` (pasangan-pasangan titik pembentuk garis *convex hull*), serta beberapa *method* dengan kegunaan yang tertera pada komentar di salinan kode program di bawah ini. Fungsi utama pada *file* ini adalah `convexHull()` yang akan menginisiasi partisi (*divide*) himpunan titik untuk pertama kali kemudian akan memanggil fungsi `recursiveConvexHull()` yang akan mencari pasangan titik pembentuk garis *convex hull* secara rekursif (*divide and conquer*) untuk kedua partisi.

```
"""
Modul untuk menghasilkan sebuah convex hull dari masukan berupa himpunan titik
(data 2D)
"""

# Konstanta (untuk menentukan posisi (sisi kanan/kiri) suatu titik relatif
terhadap suatu garis)
RIGHT = 1
LEFT = -1

""" Kelas myConvexHull """
class myConvexHull:
    # Inisialisasi atribut
    def __init__(self, Points):
        self.allPoints = Points                                  ## Himpunan semua
titik
        self.countPoints = len(Points)                           ## Jumlah titik
dalam himpunan titik
        self.allPointsIdx = list(range(0,self.countPoints)) ## Indeks himpunan
titik
        self.simplices = self.convexHull()                       ## Himpunan garis
pembentuk convex hull


    # Method utama untuk menghasilkan vertices (garis-garis) yang membentuk
convex hull
    def convexHull(self):
        ## Kalau hanya ada < 2 titik, kembalikan 0 garis (tidak ada garis)
        if (self.countPoints < 2):
            return []
        ## Kalau hanya ada 2 titik, kembalikan sebuah garis yang menghubungkan
kedua titik
```

```
        elif (self.countPoints == 2):
            return [[0,1]]
        ## else (kalau ada > 2 titik)
        minAbsisPointIdx, maxAbsisPointIdx = self.MinMaxAbsisPoint()
        leftPointsIdx, rightPointsIdx = self.splitPoints(self.allPointsIdx,
self.allPoints[minAbsisPointIdx], self.allPoints[maxAbsisPointIdx])
        S1 = self.recursiveConvexHull(leftPointsIdx, minAbsisPointIdx,
maxAbsisPointIdx, LEFT)      ## convex hull pada partisi atas
        S2 = self.recursiveConvexHull(rightPointsIdx, minAbsisPointIdx,
maxAbsisPointIdx, RIGHT)    ## convex hull pada partisi bawah
        hullPointIdx = S1 + S2
        return hullPointIdx


    # Method berupa fungsi rekursif untuk menghasilkan vertices (garis-garis)
yang membentuk convex hull
    # Akan di panggil pada method convexHull
    def recursiveConvexHull(self, evaluatedPointsIdx, point1Idx, point2Idx,
side):
        hullPointIdx = []   ## Inisialisasi variabel penampung pasangan titik
pembentuk garis pada convex hull
        ## Mencari titik dengan jarak terjauh dari garis partisi /
garis(point1,point2)
        maxDistance = 0
        maxPointIdx = -1
        for idx in (evaluatedPointsIdx):
            distance = self.distanceFromLine(self.allPoints[idx],
self.allPoints[point1Idx], self.allPoints[point2Idx])
            if (distance > maxDistance):
                maxPointIdx = idx
                maxDistance = distance
        ## Jika tidak ketemu, garis pertisi / garis(point1,point2) adalah
bagian dari convex hull
        if (maxPointIdx == -1):
            hullPointIdx.append([point1Idx, point2Idx])
            return hullPointIdx
        ## else (jika ketemu, lakukan algortima convex hull secara rekursif)
        leftPointsIdx1, rightPointsIdx1 = self.splitPoints(evaluatedPointsIdx,
self.allPoints[point1Idx], self.allPoints[maxPointIdx])
        leftPointsIdx2, rightPointsIdx2 = self.splitPoints(evaluatedPointsIdx,
self.allPoints[point2Idx], self.allPoints[maxPointIdx])
        if (side == LEFT):
            S1 = self.recursiveConvexHull(leftPointsIdx1, point1Idx,
maxPointIdx, side)
            S2 = self.recursiveConvexHull(rightPointsIdx2, point2Idx,
maxPointIdx, -side)
        elif (side == RIGHT):
            S1 = self.recursiveConvexHull(rightPointsIdx1, point1Idx,
maxPointIdx, side)
```

```
            S2 = self.recursiveConvexHull(leftPointsIdx2, point2Idx,
maxPointIdx, -side)
        hullPointIdx = S1 + S2
        return hullPointIdx


    # Method untuk membagi himpunan/list titik menjadi 2 bagian dengan
garis(L1,L2) adalah pembatasnya
    def splitPoints(self, evaluatedPointsIdx, L1, L2):
        ## Inisialisasi variabel penampung hasil partisi
        leftPointsIdx = []
        rightPointsIdx = []
        for i in evaluatedPointsIdx:
            ## determinan = x1y2 + x3y1 + x2y3 - x3y2 - x2y1 - x1y3
            ## Jika determinan > 0, maka titik (x3,y3) ada di sebelah kiri
(atas) garis ((x1,y1),(x2,y2))
            det = L1[0]*L2[1] + self.allPoints[i][0]*L1[1] +
L2[0]*self.allPoints[i][1] - self.allPoints[i][0]*L2[1] - L2[0]*L1[1] -
L1[0]*self.allPoints[i][1]
            if (det > 0):
                leftPointsIdx.insert(len(leftPointsIdx), i)
            elif (det < 0):
                rightPointsIdx.insert(len(rightPointsIdx), i)
        return leftPointsIdx, rightPointsIdx


    # Method untuk menghasilkan titik dengan absis paling kecil dan paling
besar pada himpunan titik
    def MinMaxAbsisPoint(self):
        minXPointIdx = 0
        maxXPointIdx = 0
        for i in range (self.countPoints):
            if (self.allPoints[i][0] < self.allPoints[minXPointIdx][0]):
                minXPointIdx = i
            if (self.allPoints[i][0] > self.allPoints[maxXPointIdx][0]):
                maxXPointIdx = i
        return minXPointIdx, maxXPointIdx


    # Method untuk menghitung jarak titik P dari garis yang dibentuk oleh
titik L1 dan L2
    def distanceFromLine(self, P, L1, L2):
        return abs((P[1] - L1[1]) * (L2[0] - L1[0]) - (L2[1] - L1[1]) * (P[0]
- L1[0]))
```

2. *File* `driver_Iris.py`

*File* ini berisi *driver* untuk dataset *Iris* sesuai dengan yang tercantum pada *file* spesifikasi tugas.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import myConvexHull

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)
print(df.head())

#visualisasi hasil ConvexHull - iris-sepal
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Sepal Width vs Sepal Length')
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
    hull = myConvexHull.myConvexHull(bucket) #bagian ini diganti dengan hasil
implementasi ConvexHull Divide & Conquer
    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 %
len(colors)])
    plt.legend()
plt.show()

#visualisasi hasil ConvexHull - iris-petal
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Petal Width 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
    hull = myConvexHull.myConvexHull(bucket) #bagian ini diganti dengan hasil
implementasi ConvexHull Divide & Conquer
    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 %
len(colors)])
    plt.legend()
```

```
plt.show()
```

3. *File* `driver_General.py`

*File* ini berisi *driver* umum yang bisa digunakan untuk membentuk *convex hull* dari beberapa dataset, yaitu *Iris*, *Wine*, *Breast Cancer*, dan *Digits*. Pertama-tama program akan meminta pilihan dataset pada *user* dan meminta pilihan pasangan atribut yang diinginkan untuk membentuk *convex hull*. Kemudian, grafik hasil dari penerapan algoritma pada *library myConvexHull* akan ditampilkan ke layar. Alur program pada *file* ini tidak jauh berbeda dengan *file* `driver_iris.py` karena hanya terdapat beberapa penyesuaian untuk proses *input*/*output* dengan *user*.

```
import pandas as pd
import matplotlib.pyplot as plt
import myConvexHull
import InputOutput

# Load database
data = InputOutput.chooseDataset()

# Create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print("Data shape: {}".format(df.shape))
print(df.head())

while (True):
    # Pemilihan atribut yang ingin digunakan
    atr1, atr2 = InputOutput.chooseAttribute(data)

    # Visualisasi hasil ConvexHull - iris-sepal
    plt.figure(figsize = (10, 6))
    colors =
['r','g','b','c','m','y','k','grey','violet','saddlebrown','purple']
    plt.title("{} Vs {}".format(data.feature_names[atr1],
data.feature_names[atr2]).replace("_"," ").title())
    plt.xlabel(data.feature_names[atr1].title())
    plt.ylabel(data.feature_names[atr2].title())
    for i in range(len(data.target_names)):
        bucket = df[df['Target'] == i]
        bucket = bucket.iloc[:,[atr1,atr2]].values
        hull = myConvexHull.myConvexHull(bucket) #bagian ini diganti dengan
hasil implementasi ConvexHull Divide & Conquer
        plt.scatter(bucket[:, 0], bucket[:, 1], c=colors[i % len(colors)],
label=data.target_names[i])
        for simplex in hull.simplices:
            plt.plot(bucket[simplex, 0], bucket[simplex, 1], colors[i %
len(colors)])
        plt.legend()
```

```
    x = input("\nPress enter to show figure...")
    plt.show()

    # Konfirmasi keluar program
    confirm = InputOutput.exitConfirm()
    if (confirm == 'N' or confirm == 'n'):
        break
```

4. *File* `InputOutput.py`

*File* ini berisi fungsi-fungsi untuk menangani dan memvalidasi proses *input*/*output* pada *file* `driver_General.py`. Terdapat 3 fungsi pada *file* ini, yaitu fungsi `chooseDataset()`, `chooseAttributes()`, dan `exitConfirm()` dengan kegunaan yang tertera pada bagian komentar dari salinan kode program di bawah ini.

```
"""
Modul untuk menangani proses I/O dengan user beserta validasinya
"""

from sklearn import datasets

# Memilih dataset (ada 3 pilihan)
def chooseDataset():
    while (True):
        print("\nList of Datasets:")
        print("1. Iris plants dataset")
        print("2. Wine recognition dataset")
        print("3. Breast cancer wisconsin (diagnostic) dataset")
        print("4. Optical recognition of handwritten digits dataset")
        db = input("Choose database (1/2/3/4): ")
        if (db == '1' or db == '2' or db == '3' or db == '4'):
            break
        print("Invalid input, try again!")
    if (db == '1'):
        data = datasets.load_iris()
    elif (db == '2'):
        data = datasets.load_wine()
    elif (db == '3'):
        data = datasets.load_breast_cancer()
    elif (db == '4'):
        data = datasets.load_digits()
    return data

# Memilih pasangan atribut yang ingin dibuat convex hull-nya
def chooseAttribute(data):
    print("\nList of Attributes:")
    for i in range (len(data.feature_names)):
        print("[{:2d}] {}".format(i, data.feature_names[i]))
```

```
    while (True):
        atr1 = int(input("Choose first attribute (number)  : "))
        if (atr1 in range(0,len(data.feature_names))):
            break
        print("Invalid input, try again!\n")
    while (True):
        atr2 = int(input("Choose second attribute (number) : "))
        if (atr2 in range(0,len(data.feature_names))):
            break
        print("Invalid input, try again!\n")
    return atr1, atr2

# Konfirmasi keluar dari program
def exitConfirm():
    while (True):
        x = input("\nContinue? (Y/N): ")
        if (x == 'Y' or x == 'y' or x == 'N' or x == 'n'):
            break
        print("Invalid input, try again!")
    return x
```

# C. SCREENSHOTS

1. Tampilan Awal

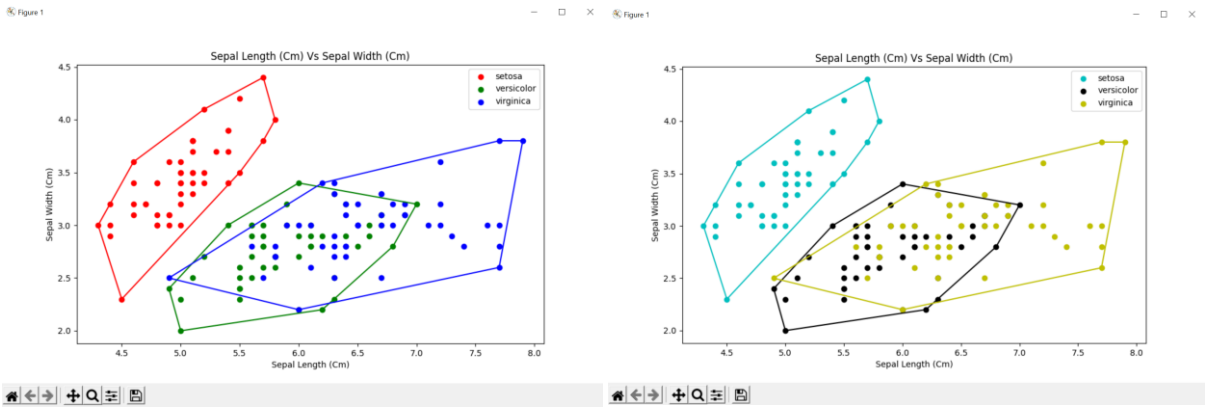
```
List of Datasets:
1. Iris plants dataset
2. Wine recognition dataset
3. Breast cancer wisconsin (diagnostic) dataset
4. Optical recognition of handwritten digits dataset
Choose database (1/2/3/4): 1
Data shape: (150, 5)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  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

List of Attributes:
[ 0] sepal length (cm)
[ 1] sepal width (cm)
[ 2] petal length (cm)
[ 3] petal width (cm)
Choose first attribute (number)  : 0
Choose second attribute (number) : 1

Press enter to show figure...
```

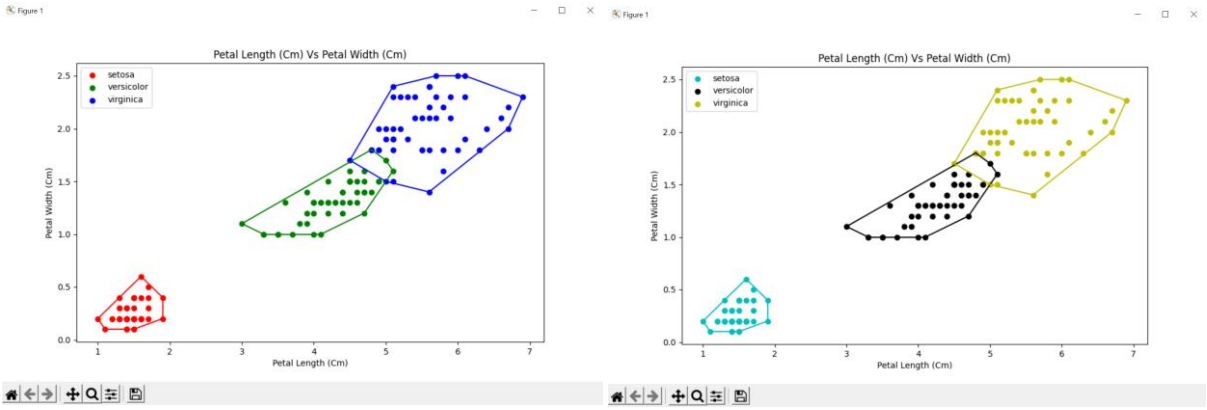
***Gambar 2 Tampilan awal program utama (driver_general.py)***

2. *Convex Hull* untuk dataset *Iris* dengan pasangan atribut (*petal-length*, *petal-width*)



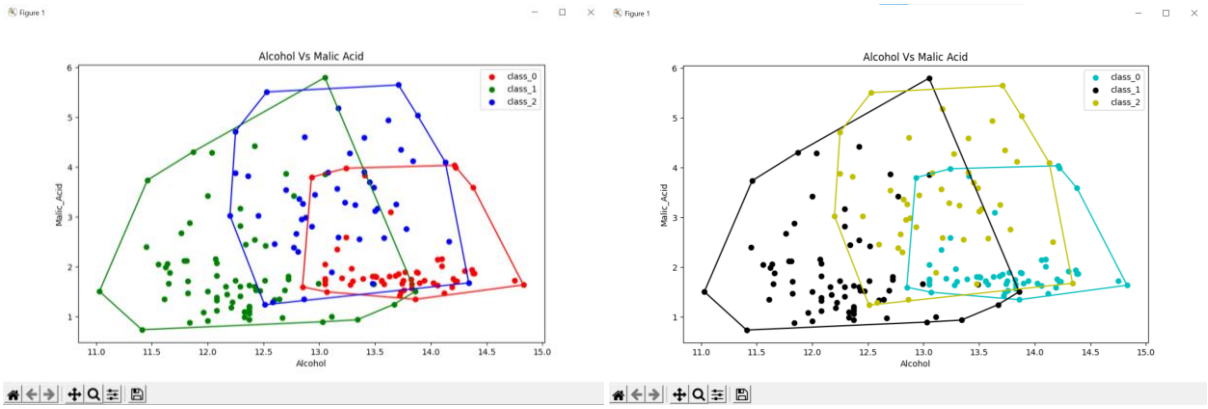

***Gambar 3 Tampilan convex hull Iris-1 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

3. *Convex Hull* untuk dataset *Iris* dengan pasangan atribut (*sepal-length*, *sepal-width*)



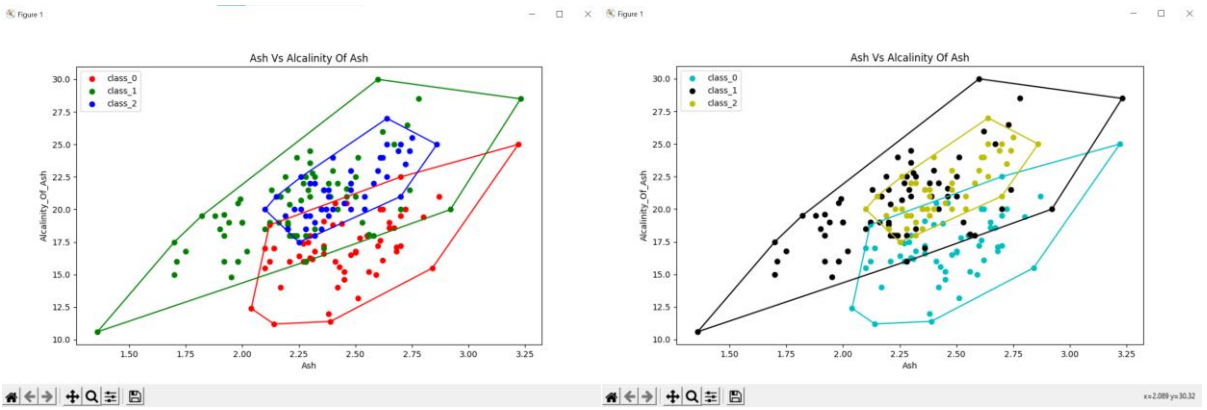

***Gambar 4 Tampilan convex hull Iris-2 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

4. *Convex Hull* untuk dataset *Wine* dengan pasangan atribut (*alcohol*, *malic-acid*)



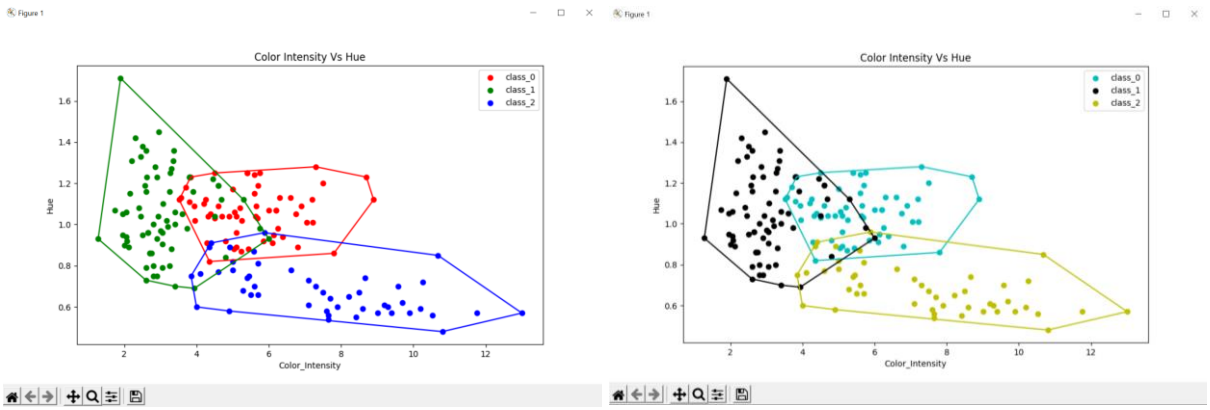

***Gambar 5 Tampilan convex hull Wine-1 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

5. *Convex Hull* untuk dataset *Wine* dengan pasangan atribut (*ash, alcalinity-of-ash*)



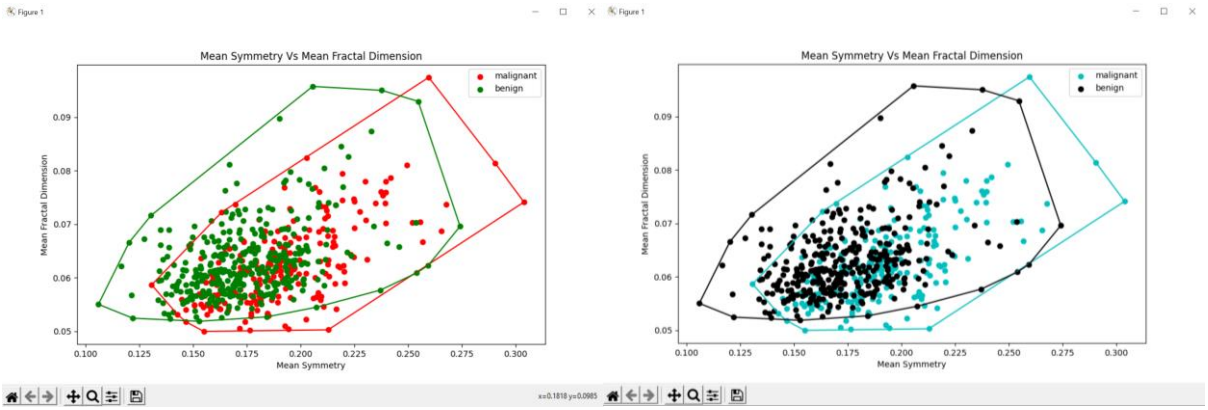

***Gambar 6 Tampilan convex hull Wine-2 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

6. *Convex Hull* untuk dataset *Wine* dengan pasangan atribut (*color-intensity, hue*)



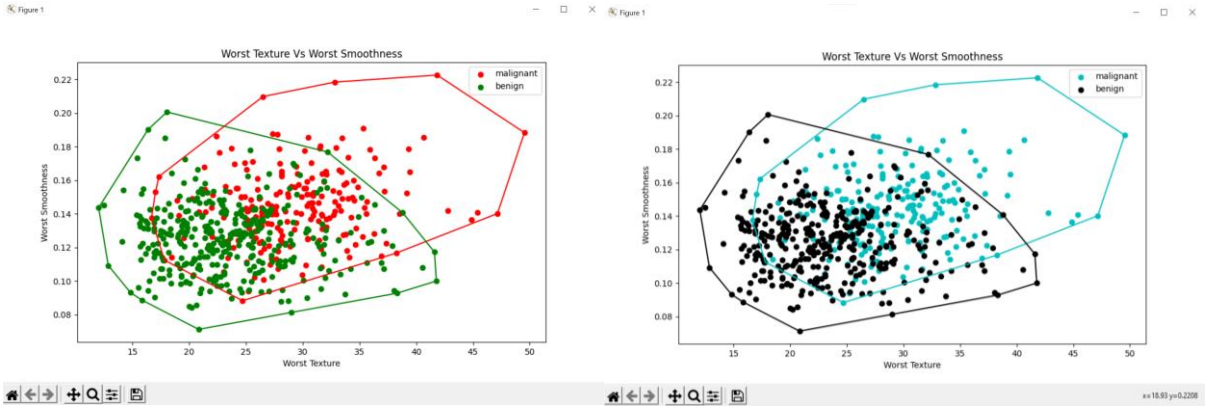

***Gambar 7 Tampilan convex hull Wine-3 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

7. *Convex Hull* untuk dataset *Breast Cancer* dengan pasangan atribut (*mean-symmetry, mean-fractal-dimension*)



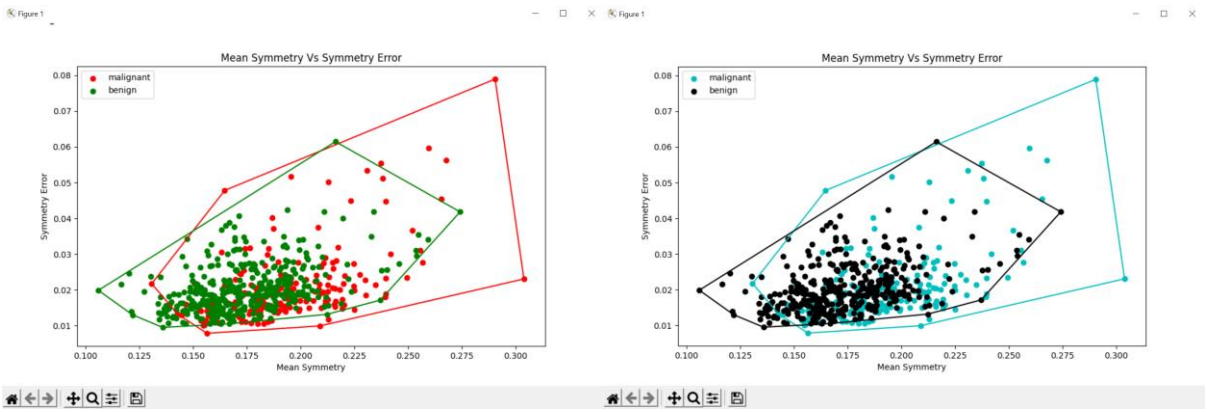

***Gambar 8 Tampilan convex hull Breast Cancer-1 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

8. *Convex Hull* untuk dataset *Breast Cancer* dengan pasangan atribut (*worst-texture, worst-smoothness*)



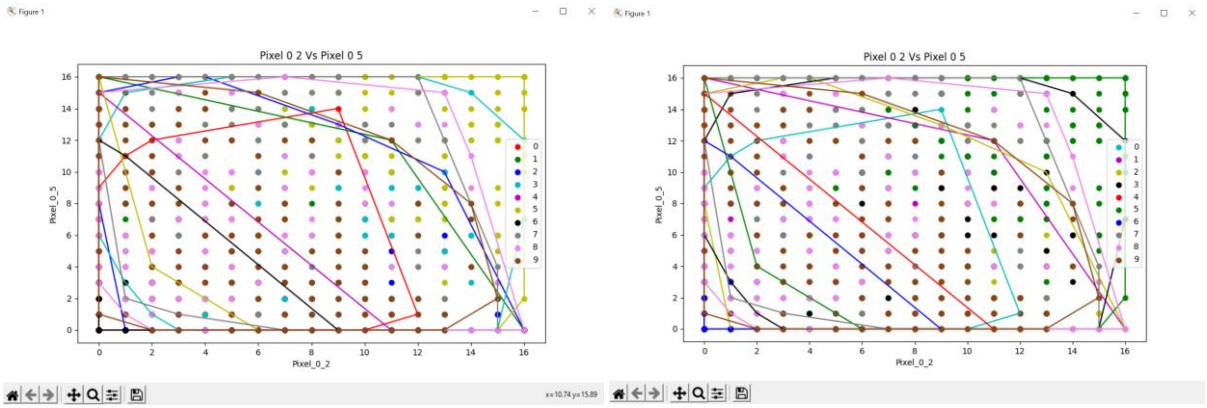

***Gambar 9 Tampilan convex hull Breast Cancer-2 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

9. *Convex Hull* untuk dataset *Breast Cancer* dengan pasangan atribut (*mean-symmetry, symmetry error*)



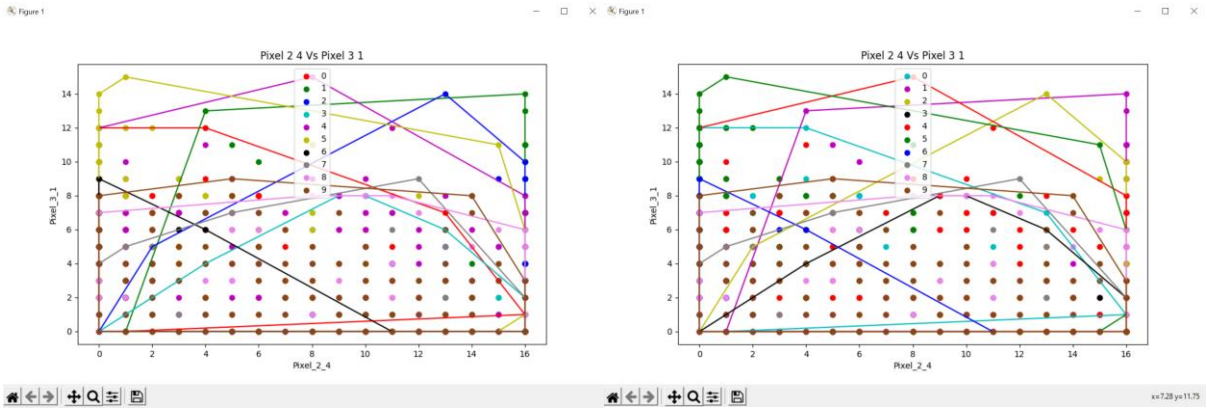

***Gambar 10 Tampilan convex hull Breast Cancer-3 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

10. *Convex Hull* untuk dataset *Digits* dengan pasangan atribut (*pixel-0-2, pixel-0-5*)




***Gambar 11 Tampilan convex hull Digits-1 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

11. *Convex Hull* untuk dataset *Digits* dengan pasangan atribut (*pixel-2-4, pixel-3-1*)




***Gambar 12 Tampilan convex hull Digits-2 menggunakan Pustaka myConvexHull (kiri) dan Pustaka SciPy (kanan)***

# D. TAUTAN KODE PROGRAM

Link GitHub: https://github.com/HanselTanoto/Convex-Hull

# E. *CHECKBOX*

| 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 | ✓ | |