Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

Implementasi Algoritma Divide and Conquer Untek penyelesaian Masalah Convex Hull

Oleh:

13520121 – Nicholas Budiono

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2022

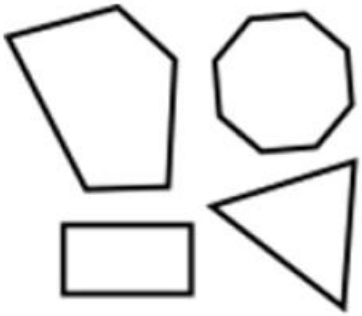
Penjelasan Divide and Conquer Dalam Convex Hull:

Algoritma Divide and Conquer:

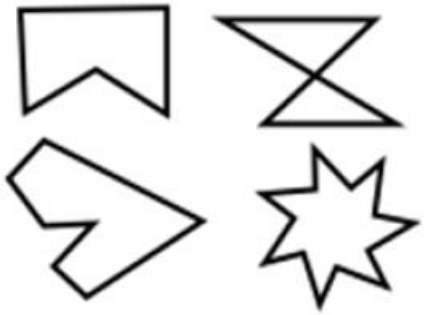
Divide and conquer merupakan algorita yang berprinsip memecah – mecah suatu permasalahan yang terlalu besar menjadi bagian – bagian kecil, sehingga lebih mudah untuk diselesaikan

Convex Hull:

- 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.
- Contoh gambar 1 adalah poligon yang convex , sedangkan gambar 2 menunjukkan contoh yang non convex.



Gambar 1: convex



Gambar 2:non convex

Pengaplikasian Algoritma Divide and Conquer Untuk Convex Hull:

- S : himpunan titik sebanyak n , dengan n >1, yaitu titik p 1 x 1 , y 1) hingga p n x n , y n) pada bidang k artesian dua dimensi
- Kumpulan titik diurutkan berdasarkan nilai absis yang menaik, dan jika ada nilai absis yang sama, maka diurutkan dengan nilai ordinat yang menaik
- p1 dan pn adalah dua titik ekstrim yang akan membentuk convex hull untuk kumpulan titik tersebut.

Deskripsi Jalan Kode:

Pembacaan Data Frame:

Pembacaan dataframe dilakukan dengan menggunakan fungsi bawaaan dari library pandas dan datasets, data yang digunakan adalah dataframe iris, dataframe wine, dan dataframe breast_cancer.

Pemrosessan data:

Dataframe yang ada akan diproses dengan cara Divide and Conquer, dengan awalnya mencari garis terendah dan terjauh dari kumpulan titik, disini, saya mengeceknya dengan Panjang nya dari titik O(0 , 0). Setelah mendapatkannya, akan di cek menggunakan fungsi DnC, jika ada titik di kanan/ kiri, maka akan dicek lagi jika dari dua garis yang digabungkan oleh titik awal memiliki titik di luar dari garis itu. Jika ada, akan dilakukan rekursif dengan garis pecahan titik terjauh tersebut menjadi base nya. Jika tidak ada, dibuat dua garis dari titik awal ke titik terjauh yang ada dari garis itu di satu sisi dan disimpan ke list tempat penyimpannan garis terluar. Setelah itu, garis tersebut akan di sort sehingga jadi sebuah polygon dan dipecah jadi titik-titik sehingga bisa di-plot.

Pengeluaran Data Frame:

Dari proses diatas, akan didapat titik-titik yang siap di-plot, menggunakan matplotlib, akan di-plot sehingga menjadi polygon yang sebagai covex hull dari kumpulan titik tersebut.

Deskripsi Kelas:

[1] find_length_from_o(x, y)
Mencari jarak titik dengan koordinat x dan y dari titik O (0, 0)

[2] Point_From_Line(line, point)
Mencari jarak dari line, yaitu dua titik, dengan point secara tegak lurus

[3] Side_From_Line(line, point)
Mencari apakah point berada di kanan, kiri, atau pada line

[4] sort(list_of_points)
Men-sort titik-titik sehingga menjadi polygon

[5] First_Line(df, target, x_coordinate_col, y_coordinate_col, diff_pattern_col)
Mencari garis awal (dua titik min dan maks) yang ada pada suatu kumpulan titik

[6] Any_Point_Side_Line(df, line, target, x_coordinate_col, y_coordinate_col, diff_pattern_col, direction)
Mengecek apakah ada titik yang masih ada di kanan atau kiri dari garis

[7] Get_Furtest_Point_From_Line_One_Side(df, line, target, x_coordinate_col, y_coordinate_col, diff_pattern_col, direction)
encari titik terjauh dari garis dengan kondisi di kiri atau kanan dari garis

[8] DnC(df, line, safed_line, target, x_coordinate_col, y_coordinate_col, diff_pattern_col)
Melakukan algoritma Divide and Conquer

[9] Finishing(df, target, x_coordinate_col, y_coordinate_col, diff_pattern_col)
Mengubah output dari DnC berupa set dari garis menjadi point unik sehingga bisa di-plot

Source Code:

MyConvexHull.py:

```
import math


def find_length_from_o(x, y):

    # mengembalikan jarak dari satu titik ke titik O
    return ((x - 0)**2 + (y - 0)**2)**(1/2)


def Point_From_Line(line, point):

    # memetakan data menjadi 3 titik
    x_a = line[0][0]
    y_a = line[0][1]
    x_b = line[1][0]
    y_b = line[1][1]
    x_p = point[0]
    y_p = point[1]

    # mengembalikkan jarak dari garis ke titik secara tegak lurus
    return (1/2) * abs((x_a - x_p) * (y_b-y_a) - (x_a - x_b) * (y_p -
y_a))


def Side_From_Line(line, point):

    # memetakan data menjadi 3 titik
    x_a = line[0][0]
    y_a = line[0][1]
    x_b = line[1][0]
    y_b = line[1][1]
    x_p = point[0]
    y_p = point[1]

    # mencari keadaan dari titik, di kiri, kanan, atau di garis
    ff = ((x_b - x_a)*(y_p - y_a) - (y_b - y_a)*(x_p - x_a))
    if ff > 0:
```

```
        return -1  # kiri
    elif ff < 0:
        return 1  # kanan
    else:
        return 0  # di garis


def sort(list_of_points):

    # men-sortir titik sehingga bisa di-plot menjadi polygon
    x = list(list_of_points)
    cent = (sum([p[0] for p in x])/len(x),
            sum([p[1] for p in x])/len(x))
    x.sort(key=lambda p: math.atan2(p[1]-cent[1], p[0]-cent[0]))
    return(x)


def First_Line(df, target, x_coordinate_col, y_coordinate_col,
diff_pattern_col):

    # inisialsasi
    df1 = df[df[diff_pattern_col] == target].reset_index(drop=True)
    index_0 = df.columns.get_loc(x_coordinate_col)
    index_1 = df.columns.get_loc(y_coordinate_col)
    # mencari x dan y yang minimum dan maksimum
    idx_1 = df1[x_coordinate_col].idxmin()
    idx_2 = df1[y_coordinate_col].idxmin()
    idx_3 = df1[x_coordinate_col].idxmax()
    idx_4 = df1[y_coordinate_col].idxmax()

    # dicari jarak nya dari titik O untuk dicari titik terdekat dan
terjauh
    len_1 = find_length_from_o(
        df1.loc[idx_1][index_0], df1.loc[idx_1][index_1])
    len_2 = find_length_from_o(
        df1.loc[idx_2][index_0], df1.loc[idx_2][index_1])
    len_3 = find_length_from_o(
        df1.loc[idx_3][index_0], df1.loc[idx_3][index_1])
    len_4 = find_length_from_o(
```

```
        df1.loc[idx_4][index_0], df1.loc[idx_4][index_1])

    # perbandingan untuk mencari terkecil dan terbesaar yang tepat
    if len_1 < len_2:
        if len_3 > len_4:
            return (((df1.loc[idx_1][index_0],
df1.loc[idx_1][index_1]), (df1.loc[idx_3][index_0],
df1.loc[idx_3][index_1])))
        else:
            return (((df1.loc[idx_1][index_0],
df1.loc[idx_1][index_1]), (df1.loc[idx_4][index_0],
df1.loc[idx_4][index_1])))
    else:
        if len_3 > len_4:
            return (((df1.loc[idx_2][index_0],
df1.loc[idx_2][index_1]), (df1.loc[idx_3][index_0],
df1.loc[idx_3][index_1])))
        else:
            return (((df1.loc[idx_2][index_0],
df1.loc[idx_2][index_1]), (df1.loc[idx_4][index_0],
df1.loc[idx_4][index_1])))


def Any_Point_Side_Line(df, line, target, x_coordinate_col,
y_coordinate_col, diff_pattern_col, direction):

    # men-sortir dataframe menjadi subset dataframe yang memiliki
target tertentu
    df1 = df[df[diff_pattern_col] == target].reset_index(drop=True)

    # mengecek apakah ada titik pada target tertentu ada pada sisi
kiri dari garis
    if(direction == -1):
        for i in range(len(df1[df1[diff_pattern_col] == target])):
            if (Side_From_Line(line, (df1[x_coordinate_col][i],
df1[y_coordinate_col][i])) == -1):
                return True
```

```
    # mengecek apakah ada titik pada target tertentu ada pada sisi
kanan dari garis
    elif(direction == 1):
        for i in range(len(df1[df1[diff_pattern_col] == target])):
            if (Side_From_Line(line, (df1[x_coordinate_col][i],
df1[y_coordinate_col][i])) == 1):
                return True

    return False


def Get_Furtest_Point_From_Line_One_Side(df, line, target,
x_coordinate_col, y_coordinate_col, diff_pattern_col, direction):

    # men-sortir dataframe menjadi subset dataframe yang memiliki
target tertentu
    df1 = df[df[diff_pattern_col] == target].reset_index(drop=True)
    max_length = 0
    max_point_idx = 0

    # mencari titik terjauh dari garis dengan mencoba satu-satu di
bagian kiri
    if(direction == -1):
        for i in range(len(df1[df1[diff_pattern_col] == target])):
            if(Side_From_Line(line, (df1[x_coordinate_col][i],
df1[y_coordinate_col][i])) == -1):
                length = Point_From_Line(
                    line, (df1[x_coordinate_col][i],
df1[y_coordinate_col][i]))
                if length >= max_length:
                    max_point_idx = i
                    max_length = length

    # mencari titik terjauh dari garis dengan mencoba satu-satu di
bagian kanan
    elif(direction == 1):
        for i in range(len(df1[df1[diff_pattern_col] == target])):
            if(Side_From_Line(line, (df1[x_coordinate_col][i],
df1[y_coordinate_col][i])) == 1):
```

```
                length = Point_From_Line(
                    line, (df1[x_coordinate_col][i],
df1[y_coordinate_col][i]))
                if length >= max_length:
                    max_point_idx = i
                    max_length = length

    return (df1[x_coordinate_col][max_point_idx],
df1[y_coordinate_col][max_point_idx])


def DnC(df, line, safed_line, target, x_coordinate_col,
y_coordinate_col, diff_pattern_col):

    # inisialisasi
    if(len(safed_line) == 0):
        safed_line.add((First_Line(df, target, x_coordinate_col,
                                   y_coordinate_col,
diff_pattern_col)))
    x_1 = set()
    x_2 = set()
    x_3 = set()
    x_4 = set()
    out_1 = set()
    out_2 = set()
    out_3 = set()
    out_4 = set()
    out = set()

    # mengecek jika di satu sisi ada titik dari target yang sama
    if(Any_Point_Side_Line(df, line, target, x_coordinate_col,
y_coordinate_col, diff_pattern_col, -1)):
        furtest_point = Get_Furtest_Point_From_Line_One_Side(df,
                                                             line,
target, x_coordinate_col, y_coordinate_col, diff_pattern_col, -1)

        # jika tidak ada di safed_line, maka akan dicari
        if not((((line)[0], furtest_point)) in safed_line):
```

```
            # jika ada titik yang masih ada di satu sisi, dengan
tumpuan titik awal dan titik terjauh, akan dilakukan rekursif
            if(Any_Point_Side_Line(df, ((line)[0], furtest_point),
target, x_coordinate_col, y_coordinate_col, diff_pattern_col, -1)):
                safed_line.add((((line)[0], furtest_point)))
                x_1 = DnC(df, (((line)[0], furtest_point)),
safed_line, target, x_coordinate_col,
                          y_coordinate_col, diff_pattern_col)

            # jika tidak ada titik, lansung masukkan titik tersebut ke
safed_line
            else:
                safed_line.add((((line)[0], furtest_point)))
            out_1 = set.union(safed_line, x_1)

        # jika tidak ada di safed_line, maka akan dicari
        if not(((furtest_point, (line)[1])) in safed_line):

            # jika ada titik yang masih ada di satu sisi, dengan
tumpuan titik awal dan titik terjauh, akan dilakukan rekursif
            if(Any_Point_Side_Line(df, ((line)[0], furtest_point),
target, x_coordinate_col, y_coordinate_col, diff_pattern_col, 1)):
                safed_line.add(((furtest_point, (line)[1])))
                x_2 = DnC(df, ((furtest_point, (line)[1])),
safed_line, target, x_coordinate_col,
                          y_coordinate_col, diff_pattern_col)

            # jika tidak ada titik, lansung masukkan titik tersebut ke
safed_line
            else:
                safed_line.add(((furtest_point, (line)[1])))
            out_2 = set.union(safed_line, x_2)

    # mengecek jika di satu sisi ada titik dari target yang sama
    if(Any_Point_Side_Line(df, line, target, x_coordinate_col,
y_coordinate_col, diff_pattern_col, 1)):
        furtest_point = Get_Furtest_Point_From_Line_One_Side(df,
                                                             line,
target, x_coordinate_col, y_coordinate_col, diff_pattern_col, 1)
```

```
        # jika tidak ada di safed_line, maka akan dicari
        if not((((line)[0], furtest_point)) in safed_line):

            # jika ada titik yang masih ada di satu sisi, dengan
tumpuan titik awal dan titik terjauh, akan dilakukan rekursif
            if(Any_Point_Side_Line(df, ((line)[0], furtest_point),
target, x_coordinate_col, y_coordinate_col, diff_pattern_col, -1)):
                safed_line.add((((line)[0], furtest_point)))
                x_3 = DnC(df, (((line)[0], furtest_point)),
safed_line, target, x_coordinate_col,
                          y_coordinate_col, diff_pattern_col)

            # jika tidak ada titik, lansung masukkan titik tersebut ke
safed_line
            else:
                safed_line.add((((line)[0], furtest_point)))
            out_3 = set.union(safed_line, x_3)

        # jika tidak ada di safed_line, maka akan dicari
        if not(((furtest_point, (line)[1])) in safed_line):

            # jika ada titik yang masih ada di satu sisi, dengan
tumpuan titik awal dan titik terjauh, akan dilakukan rekursif
            if(Any_Point_Side_Line(df, ((line)[0], furtest_point),
target, x_coordinate_col, y_coordinate_col, diff_pattern_col, 1)):
                safed_line.add(((furtest_point, (line)[1])))
                x_4 = DnC(df, ((furtest_point, (line)[1])),
safed_line, target, x_coordinate_col,
                          y_coordinate_col, diff_pattern_col)

            # jika tidak ada titik, lansung masukkan titik tersebut ke
safed_line
            else:
                safed_line.add(((furtest_point, (line)[1])))
            out_4 = set.union(safed_line, x_4)

        # menggabungkan semua garis yang ada menjadi satu set
        out = set.union(out_1, out_2, out_3, out_4)
```

```
    return (out)


def Finishing(df, target, x_coordinate_col, y_coordinate_col,
diff_pattern_col):

    # inisialisasi
    list_points = []
    set_point = set()

    # menerima hasil DnC berdasarkan data yang diminta
    list_line = list(DnC(df, First_Line(df, target, x_coordinate_col,
y_coordinate_col, diff_pattern_col),
                         set(), target, x_coordinate_col,
y_coordinate_col, diff_pattern_col))

    # membuat dari garis menjadi titik sehingga bisa di-plot
    for i in range(len(list_line)):
        for j in range(2):
            list_points.append(list_line[i][j])

    # diubah menjadi set sehingga unik
    for x in list_points:
        set_point.add(x)

    # sort output sehingga berbentuk polygon ketika di-plot
    output = sort(set_point)
    return(output)
```

Main.py:

```
# %%
import MyConvexHull as mch
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt

# input yang bisa digunakan sebagai contoh

# 0
# sepal length (cm)
# sepal width (cm)

# 0
# petal length (cm)
# petal width (cm)

# 1
# alcohol
# malic_acid

# 1
# ash
# total_phenols

# 2
# mean radius
# mean texture

# 2
# mean perimeter
# mean area

# minta input untuk menampilkan dari beberapa dataframe
dataset = int(
    input("0. Dataset iris 1. Dataset wine 2. Dataset breast_cancer"))
x_col = input("Nama kolom untuk x")
y_col = input("Nama kolom untuk y")
```

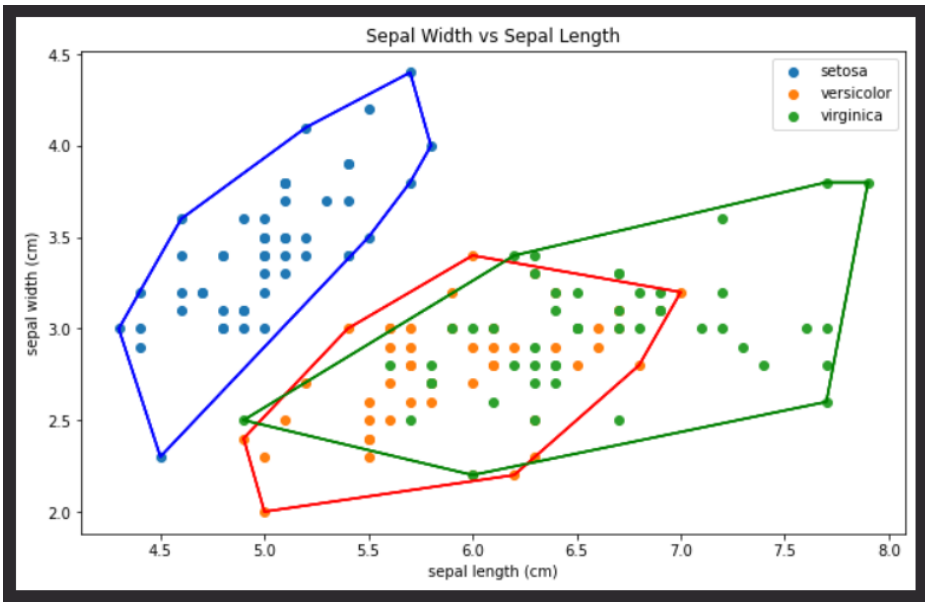
```
title = x_col + " vs " + y_col


if(dataset == 0):
    data = datasets.load_iris()
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df['Target'] = pd.DataFrame(data.target)
elif(dataset == 1):
    data = datasets.load_wine()
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df['Target'] = pd.DataFrame(data.target)
elif(dataset == 2):
    data = datasets.load_breast_cancer()
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df['Target'] = pd.DataFrame(data.target)

index_0 = df.columns.get_loc(x_col)
index_1 = df.columns.get_loc(y_col)

# tes visualisasi
plt.figure(figsize=(10, 6))
colors = ['b', 'r', 'g']
plt.title(title)
plt.xlabel(data.feature_names[index_0])
plt.ylabel(data.feature_names[index_1])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:, [index_0, index_1]].values
    plt.scatter(bucket[:, 0], bucket[:, 1],
label=data.target_names[i])
    for j in range(len(data.target_names)):
        coord = mch.Finishing(df, j, x_col, y_col, "Target")
        coord.append(coord[0])
        xs, ys = zip(*coord)
        plt.plot(xs, ys, c=colors[j])
plt.legend()
```

Screenshot:

Input:

0, sepal length (cm), sepal width (cm)

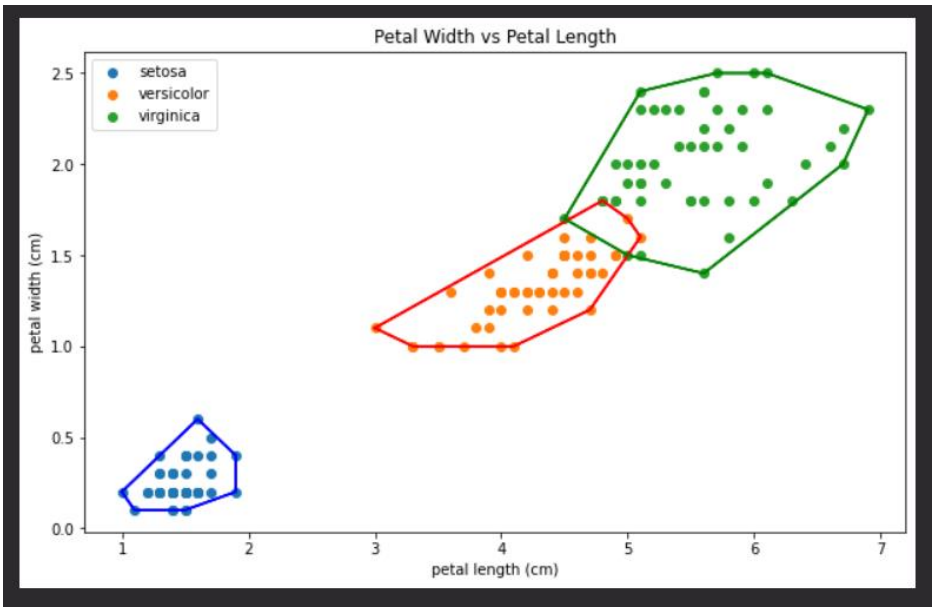
Output:



Input:

0, petal length (cm), petal width (cm)

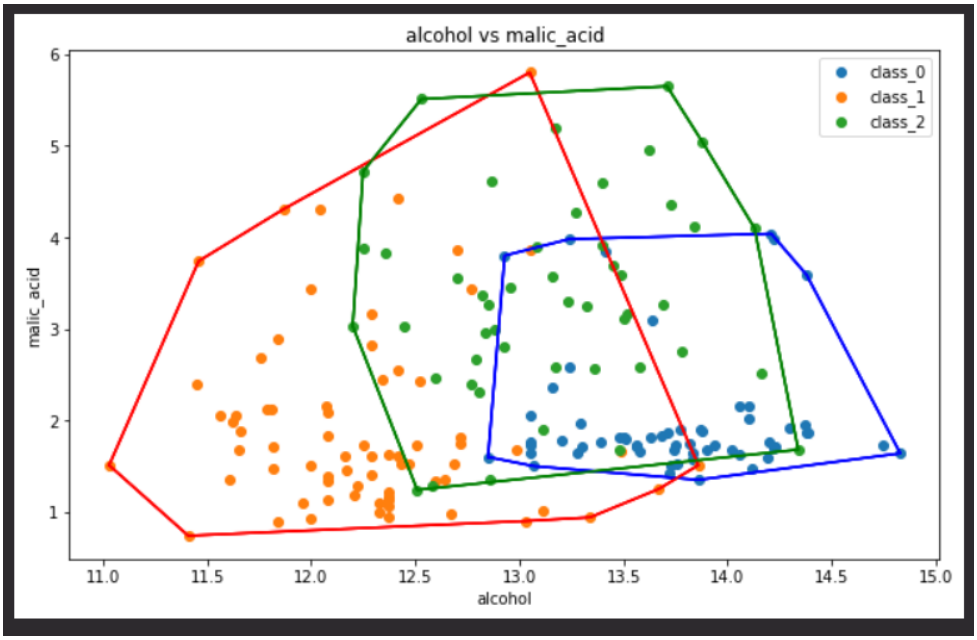
Output:



Input:

1, alcohol, malic_acid

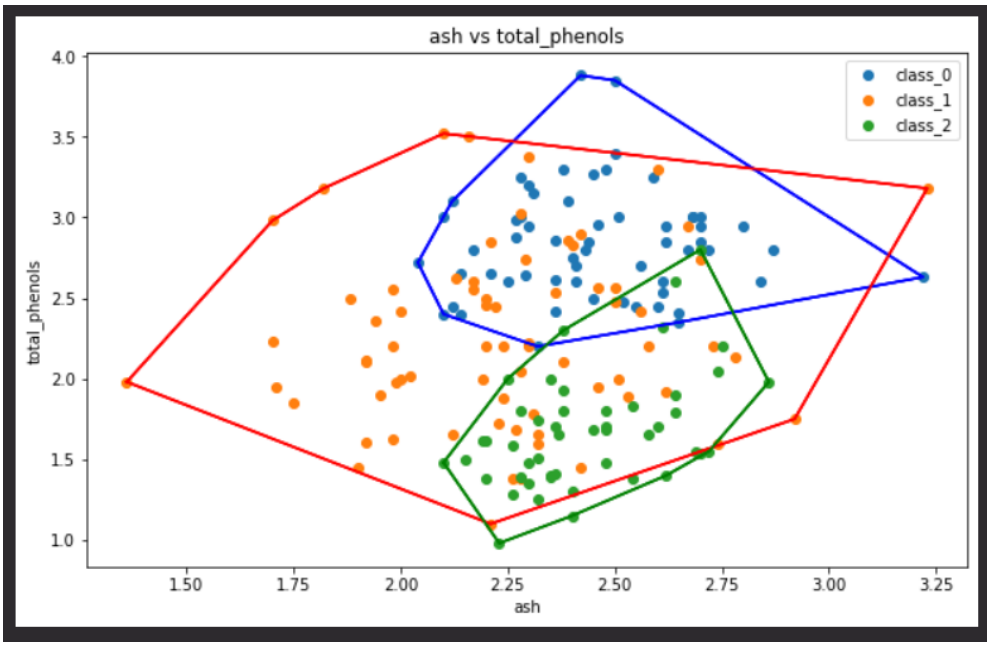
Output:



Input:

1, ash, total_phenols

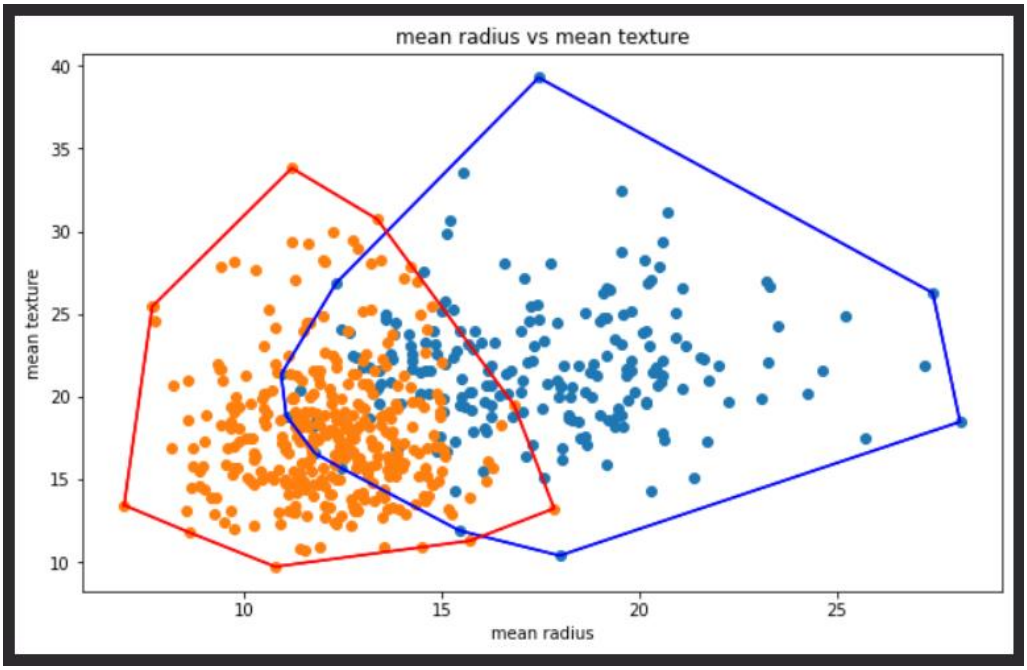
Output:



Input:

2, mean radius, mean texture

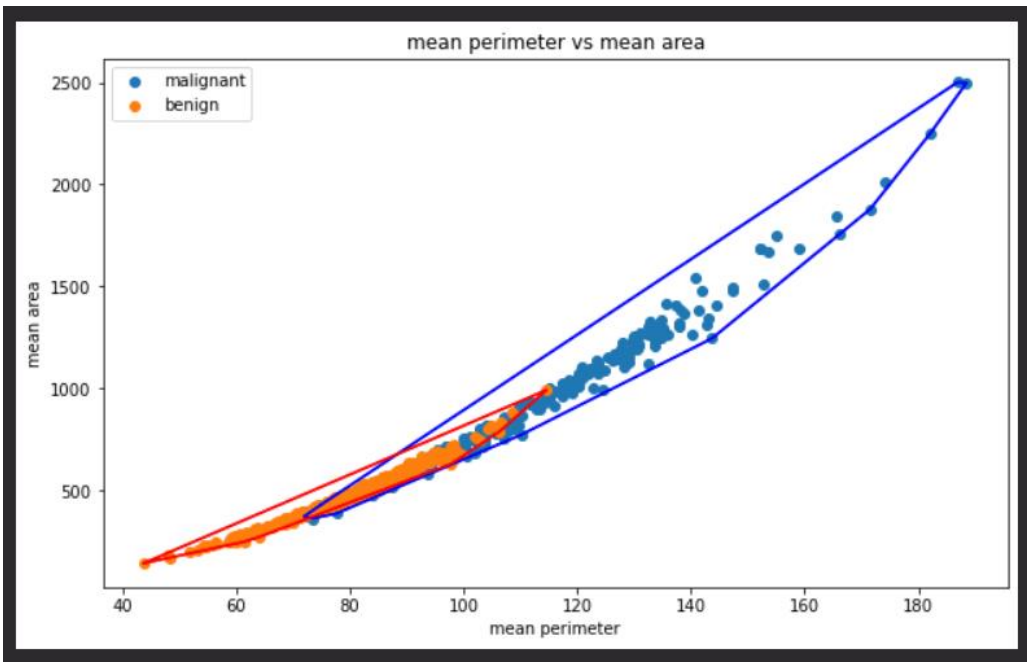
Output:



Input:

2, mean perimeter, mean area

Output:



Tabel:

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

Source Code:

https://github.com/nicholass25/Stima_Tucil_2.git