Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II tahun 2021/2022

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

disusun oleh :

Mahesa Lizardy 13520116

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

A. Algoritma

Pada Program Implementasi Convex Hull untuk Visualisasi Tes Linear Separability Dataset, Algoritma Divide and Conquer terdapat pada 2 fungsi berikut:

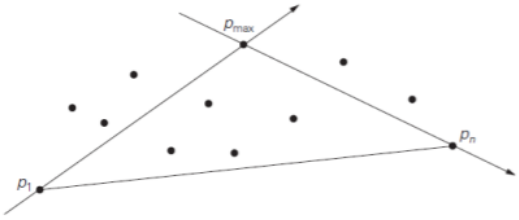
1. *convex_hull_up*
2. *convex_hull_down*
3. *quickSort*

Pertama Program akan mencari titik ekstrim P1 dan Pn. kedua titik tersebut yang akan membentuk *convex hull* untuk kumpulan titik tersebut. garis yang menghubungkan P1 dan Pn akan membagi kumpulan titik menjadi 2 bagian yaitu bagian sebelah kiri atau atas P1Pn dan bagian sebelah kanan atau bawah garis P1Pn. Pembagian titik ini akan dilakukan oleh fungsi *create_bucket_up_down*. Pengelompokan dari titik-titik tersebut dilakukan dengan menghitung nilai

Kumpulan titik atas akan dikelompokan pada array titik atas, yang terdapat di bawah akan dikelompokan pada array titik bawah,sedangkan yang berada pada garis akan diabaikan. Setelah kumpulan titik tersebut dibagi maka bagian atas akan diproses oleh fungsi *convex_hull_up* sedangkan bagian bawah akan diproses oleh fungsi *convex_hull_down*. pada fungsi convex_hull_up, terdapat 2 kondisi:

1. jika tidak ada titik lagi diatas garis maka titik P1 dan Pn akan membentuk *convex hull* pada bagian tersebut
2. jika masih terdapat titik maka akan dicari titik yang memiliki jarak terjauh dari garis P1Pn(misal Pmax). Setelah itu akan dilakukan rekursif pada bagian selanjutnya yaitu mencari *convex hull* dengan garis P1Pmax dan *convex hull* dengan garis PnPmax.

Algoritma Devide and Conquer terdapat pada proses *convex_hull_up*, dimana pada saat melakukan rekursif titik yang akan diperiksa hanya titik yang berada diatas garis(kanan/kiri). Titik yang berada dibawah garis akan diabaikan. Misal pada gambar berikut:



maka titik yang di dalam segitiga(P1PnPmax) akan diabaikan. Untuk c*onvex_hull_down* sama seperti pada *convex_hull_down* hanya saja saat rekursif titik yang diperiksa kebalikan dari *convex_hull_up* yaitu titik yang berada di bawah garis. Setelah kedua bagian tersebut berhasil di proses maka masing masing bagian akan diurutkan terlebih dahulu dengan *quickSort*. Setelah sudah terurut, kedua bagian tersebut disatukan sehingga membentuk convex hull yang utuh.

B. Program

1. main.py

```
# main.py
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn import datasets

# visualisasi hasil ConvexHull
import matplotlib.pyplot as plt
# library myConvexHull
import ConvexHull as Ch

data = datasets.load_iris() #data yang dipakai
#create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()


plt.figure(figsize=(10, 6))
colors = ['b', 'r', 'g']
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
    hull = Ch.myConvexHull(bucket)  # bagian ini diganti dengan
hasil implementasi ConvexHull Divide & Conquer
    hull = np.transpose(hull)
    plt.scatter(bucket[:, 0], bucket[:, 1],
label=data.target_names[i])
    plt.plot(hull[0], hull[1], colors[i])
plt.legend()
# menampilkan hasil ConvexHull
plt.show()
```

2. myConvexHull.py

a. fungsi myConvexHull

```
def myConvexHull(bucket):
    # Pemrosessan ConvexHull atas
    arrup =np.ndarray([0,2])
    # Pemrosessan ConvexHull bawah
    arrdown =np.ndarray([0,2])

    # menampung ConvexHull gabungan atas dan bawah
    arr =np.ndarray([0,2])

    # mencari titiks ekstrem P1, Pn
    maks,minn,raw_bucket = find_extreme(bucket)
    #assign P1 untuk pemrosesan atas
    arrup= np.append(arrup, [maks], axis=0)

    #assign Pn untuk pemrosesan bawah
    arrdown= np.append(arrdown,[minn], axis=0)

    #membagi titik menjadi bagian atas dan bawah dari garis P1Pn
    raw_bucket_up,raw_bucket_down =
create_bucket_updown(raw_bucket,maks,minn)

    # memproses convex hull atas
    arrup = convex_hull_up(raw_bucket_up,maks,minn,arrup)

    # memproses convex hull bawah
    arrdown = convex_hull_down(raw_bucket_down,maks,minn,arrdown)

    # flip for plotting
    arrup = np.flip(arrup,0)
    # sort for plotting
    arrup =quickSort(arrup,0,len(arrup)-1)

    # sort for plotting
    arrdown = quickSort(arrdown,0,len(arrdown)-1)
    # flip for plotting
```

```
    arrdown = np.flip(arrdown,0)

    # menggabungkan array convex hull atas dan bawah
    # tambahkan min diawal untuk keperluan plot
    arr = np.append(arr,[minn],axis =0)
    arr = np.append(arr,arrup,axis=0)
    arr = np.append(arr,arrdown,axis=0)

    return arr
```

b. fungsi find_extreme

```
def find_extreme(bucket):
    # mencari titik ekstreme yaitu P1 Pn
    #Pn
    maks = max(bucket[:, 0])
    indeksmaks = np.where(bucket[:, 0] == maks)
    maksordinat=bucket[indeksmaks, 1];
    i_maks = indeksmaks[0][0]

    # kasus jika terdapat nilai absis yang sama maka ditinjau nilai
ordinatnya
    for i in range(0,len(bucket)-1):
        if(maks == bucket[i, 0]):
            if (maksordinat[0,0] <= bucket[i, 1]):
                i_maks = i
    bucketmax = bucket[i_maks]

    # menghapus nilai ekstreme dari array
    bucket = np.delete(bucket,[i_maks],axis =0)

    #P1
    minn = min(bucket[:, 0])
    indeksmin = np.where(bucket[:, 0] == minn)
    minordinat=bucket[indeksmin, 1];
    i_min = indeksmin[0][0]

    for i in range(0,len(bucket)-1):
        if(minn == bucket[i, 0]):
            if(minordinat[0,0]>=bucket[i, 1]):
                i_min = i
    bucketmin = bucket[i_min]
```

```
    # menghapus nilai ekstreme dari array
    bucket = np.delete(bucket,[i_min],axis =0)
    return bucketmax, bucketmin ,bucket
```

c. fungsi detpoint

```
def detpoint(x1,y1,x2,y2,x3,y3):
    # mengirimkan nilai yang menandakan titik berada di bawah atau
atas garis
    # gradien
    m = (y3-y2)/(x3-x2)
    # offset
    c = y3 - m*x3
    # nilai titik yang akan dibandingkan
    dis = y1-m*x1
    if(dis > c):
        return 1 # berada di atas garis
    elif(dis < c):
        return -1 # berada di bawah garis
    else:
        return 0 # berada di garis
```

d. fungsi distance

```
def distance(x1,y1,x2,y2,x3,y3):
    # menghitung jarak titik ke garis
    return
abs((x2-x1)*(y1-y3)-(x1-x3)*(y2-y1))/math.sqrt((x2-x1)**2+(y2-y1)**2)
```

e. fungsi crete_bucket_updown

```
def create_bucket_updown(raw_bucket,P1,P2):
    # membagi titik menjadi bagian atas dan bawah atau kiri dan kanan
dari garis P1Pn
    raw_bucket_up=np.ndarray([0,2])
    raw_bucket_down=np.ndarray([0,2])
    for i in range(len(raw_bucket)):
        if(P2[0]==P1[0]):
            break
        determinant =
```

```
detpoint(raw_bucket[i,0],raw_bucket[i,1],P1[0],P1[1],P2[0],P2[1])
        if(determinant>0):

raw_bucket_up=np.append(raw_bucket_up,[raw_bucket[i]],axis =0)
        elif(determinant<0):

raw_bucket_down=np.append(raw_bucket_down,[raw_bucket[i]],axis =0)
        else:
            pass
    return raw_bucket_up,raw_bucket_down
```

f. fungsi convex_hull_up

```
def convex_hull_up(raw_bucket_up,P1,P2, arr):
    # convex hull untuk bagian atas
    if(len(raw_bucket_up)==0):
        # array kosong
        return arr
    elif (len(raw_bucket_up) == 1):
        # array satu elemen
        return np.append(arr,[raw_bucket_up[0]], axis=0)
    else:
        # set maks distance ke elemen ke-1
        maks_distance=
distance(P1[0],P1[1],P2[0],P2[1],raw_bucket_up[0,0],raw_bucket_up[0,1
])
        maks_indeks =0
        for i in range(len(raw_bucket_up)):
            distancepoint =
distance(P1[0],P1[1],P2[0],P2[1],raw_bucket_up[i,0],raw_bucket_up[i,1
])
            if (maks_distance*100 <= distancepoint*100):
                maks_distance=distancepoint
                maks_indeks = i

        maks_value =raw_bucket_up[maks_indeks]

        # mencari array atas dari garis P1Pn yang baru
        raw_next1,dummy =
create_bucket_updown(raw_bucket_up,P1,maks_value)
        raw_next2,dummy =
create_bucket_updown(raw_bucket_up,P2,maks_value)
```

```
        # rekursif array selanjutnya sebelah kanan
        arr = convex_hull_up(raw_next1,P1,maks_value,arr)
        # assign ke array dan delete pada raw_bucket
        arr = np.append(arr,[maks_value], axis=0)
        raw_bucket_up = np.delete(raw_bucket_up,maks_indeks,axis=0)

        # rekursif array selanjutnya sebelah kiri
        arr = convex_hull_up(raw_next2,P2,maks_value,arr)
        return arr
```

g. fungsi convex_hull_down

```
def convex_hull_down(raw_bucket_down,P1,P2, arr):
    # convex hull untuk bagian bawah
    if(len(raw_bucket_down)==0):
        return arr
    elif (len(raw_bucket_down) == 1):
        return np.append(arr,[raw_bucket_down[0]], axis=0)
    else:
        # set maks distance ke elemen ke-1
        maks_distance=
distance(P1[0],P1[1],P2[0],P2[1],raw_bucket_down[0,0],raw_bucket_down
[0,1])
        maks_indeks =0
        for i in range(len(raw_bucket_down)):
            distancepoint =
distance(P1[0],P1[1],P2[0],P2[1],raw_bucket_down[i,0],raw_bucket_down
[i,1])
            if (maks_distance*10000 <= distancepoint*10000):
                maks_distance=distancepoint
                maks_indeks = i

        maks_value =raw_bucket_down[maks_indeks]

        # mencari array bawah dari garis P1Pn yang baru
        dummy,raw_next1 =
create_bucket_updown(raw_bucket_down,P1,maks_value)
        dummy,raw_next2 =
create_bucket_updown(raw_bucket_down,P2,maks_value)

        # rekursif array selanjutnya sebelah kiri
```

```
        arr = convex_hull_down(raw_next2,P2,maks_value,arr)
        #assign ke array dan delete pada raw_bucket
        arr = np.append(arr,[maks_value], axis=0)
        raw_bucket_down =
np.delete(raw_bucket_down,maks_indeks,axis=0)
        # rekursif array selanjutnya sebelah kanan
        arr = convex_hull_down(raw_next1,P1,maks_value,arr)
        return arr
```

h. fungsi quicksort

```
def partition(array, low, high):
    # choose the rightmost element as pivot
    pivot = array[high,0]

    # pointer for greater element
    i = low - 1

    # bandingkan nilai elemen dengan pivot
    for j in range(low, high):
        if array[j,0] <= pivot:
            i = i + 1
            temp1 = array[i,0]
            temp2 = array[i,1]
            array[i]=array[j]
            array[j,0]=temp1
            array[j,1]=temp2

    temp1 = array[i+1,0]
    temp2 = array[i+1,1]
    array[i+1]=array[high]
    array[high,0]=temp1
    array[high,1]=temp2
    # return posisi partisi selesai dilakukan
    return i + 1

# function to perform quicksort
def quickSort(array, low, high):
    if low < high:
        # mencari pivot
        pi = partition(array, low, high)
```

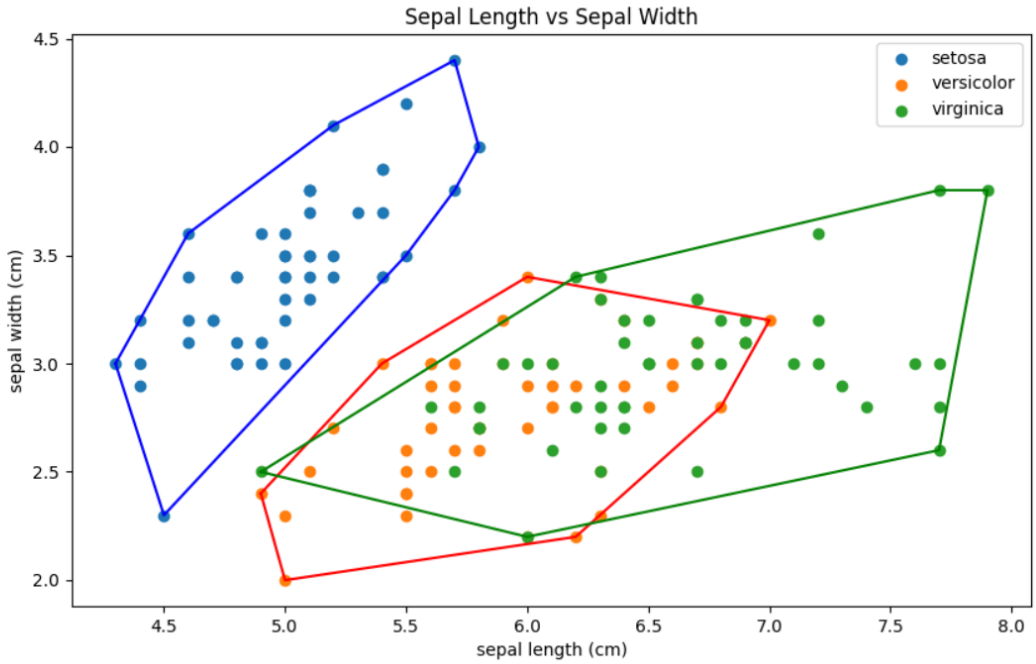
```
        # rekursif sebelah kiri pivot
        quickSort(array, low, pi - 1)
        # recursif sebelah kanan pivot
        quickSort(array, pi + 1, high)

    return array
```

C. Testing

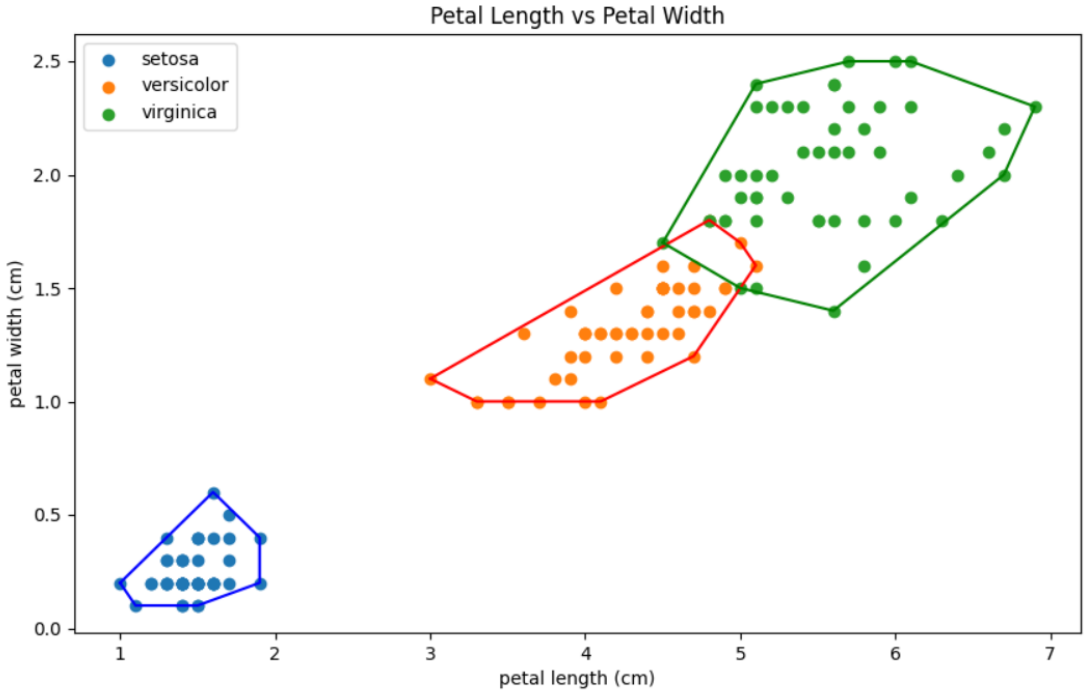
Pada Testing terdapat 3 dataset yang digunakan yaitu dataset iris,wine, dan breast_cancer

1. dataset iris
   a. sepal-length vs sepal-width



Gambar1.a convex hull sepal-length vs sepal-width

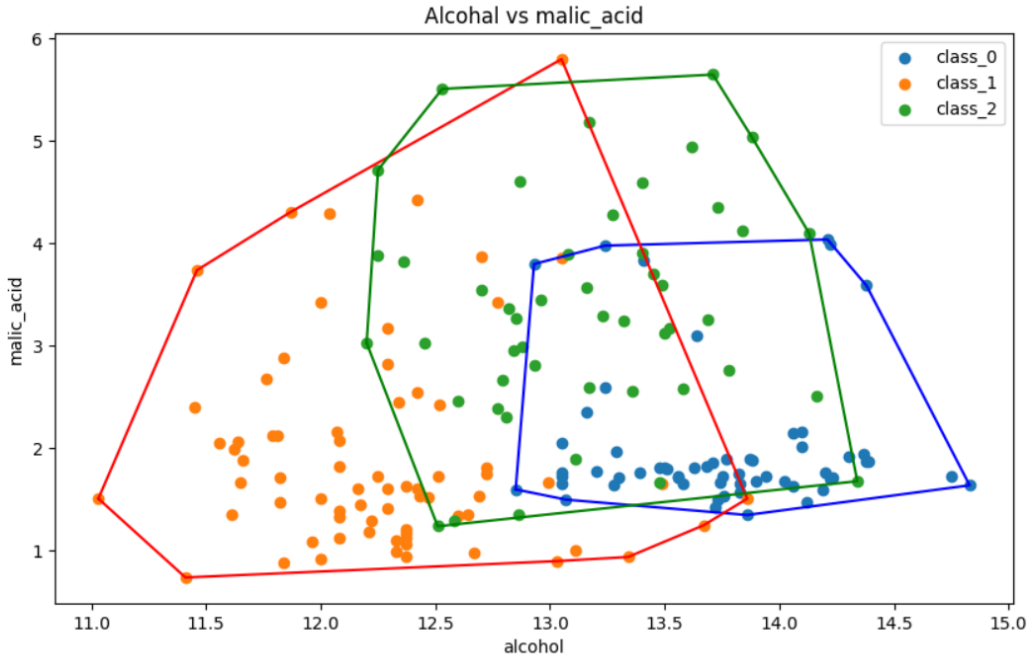
   b. petal length vs petal width



Gambar1.a convex hull petal length vs petal width

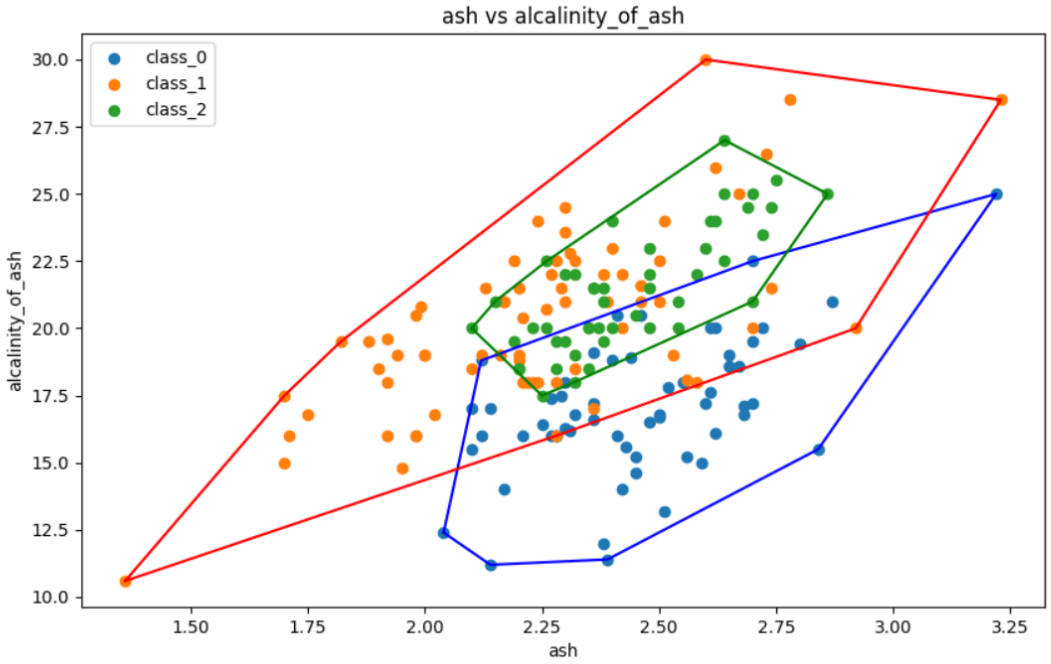
2. dataset wine

a. alcohal vs malic_acid



Gambar 2.a convex alcohol vs malic acid

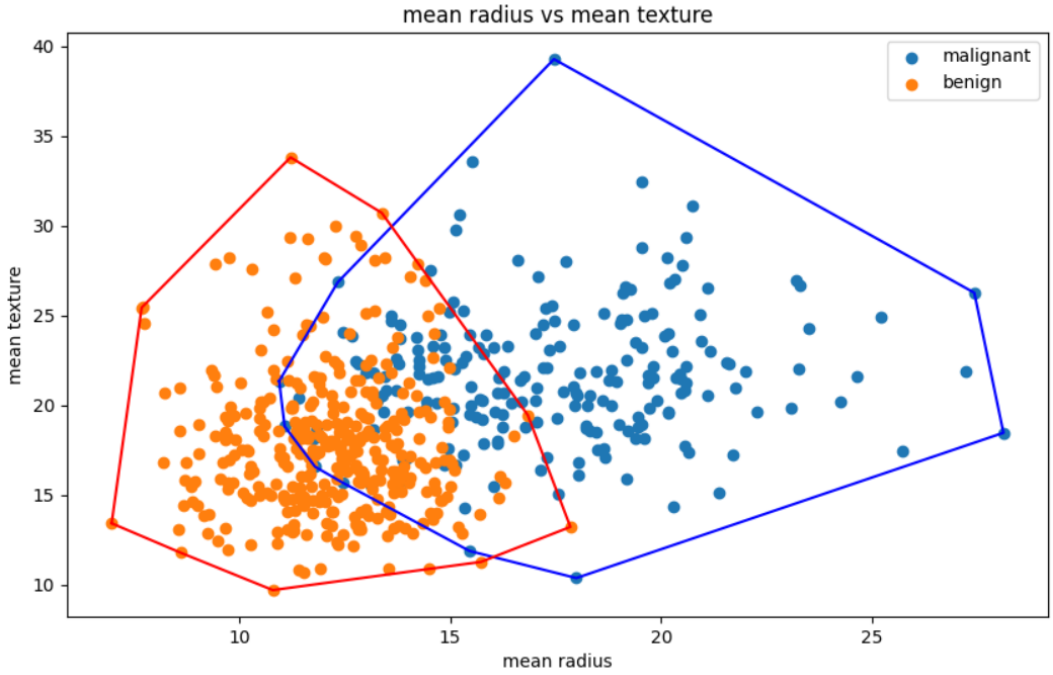
b. ash vs alcanity_of_ash



Gambar 2.b ash vs alcanity_of_ash

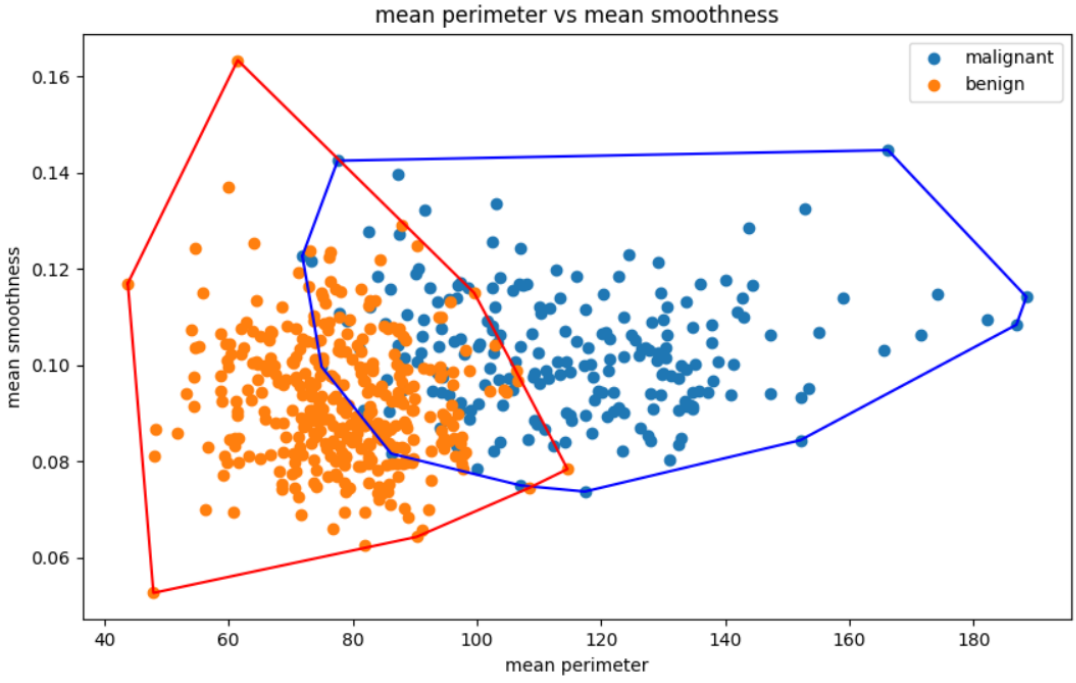
3. dataset breast_cancer

a. mean radius vs mean texture



Gambar 2.b mean radius vs mean texture

b. mean perimeter vs mean smoothness



Gambar 2.b mean perimeter vs mean smoothness

CHECKLIST

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	

LINK GITHUB
https://github.com/lizardyy/Tucil2_13520116