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

LAPORAN TUGAS KECIL

Diajukan Untuk Memenuhi Tugas IF2211 Strategi Algoritma
Semester II Tahun 2021/2022

Disusun oleh

Andika Naufal Hilmy **(13520098)**

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2021

A. ALGORITMA DIVIDE AND CONQUER

Divide and Conquer adalah salah satu strategi yang berguna dalam memecahkan masalah pemrograman. *Divide and Conquer* menggunakan langkah pembagian persoalan besar menjadi partisi persoalan yang lebih kecil yang kemudian diselesaikan membentuk solusi-solusi yang akan digabungkan menjadi solusi untuk persoalan semula.

Dalam persoalan ini, algoritma *divide and conquer* akan dipakai untuk memecahkan persoalan pembentukan *convex hull* dari kumpulan titik pada bidang dua dimensi. Secara garis besar, algoritma *divide and conquer* bekerja dengan cara berikut:

1. Mencari titik terkiri dan terkanan dari kumpulan titik
2. Membagi kumpulan titik menjadi dua zona: zona kiri atau atas garis dan zona kanan atau bawah garis dengan penghitungan determinan

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

Titik (x3,y3) berada di sebelah kiri dari garis ((x1,y1),(x2,y2)) jika hasil determinan positif

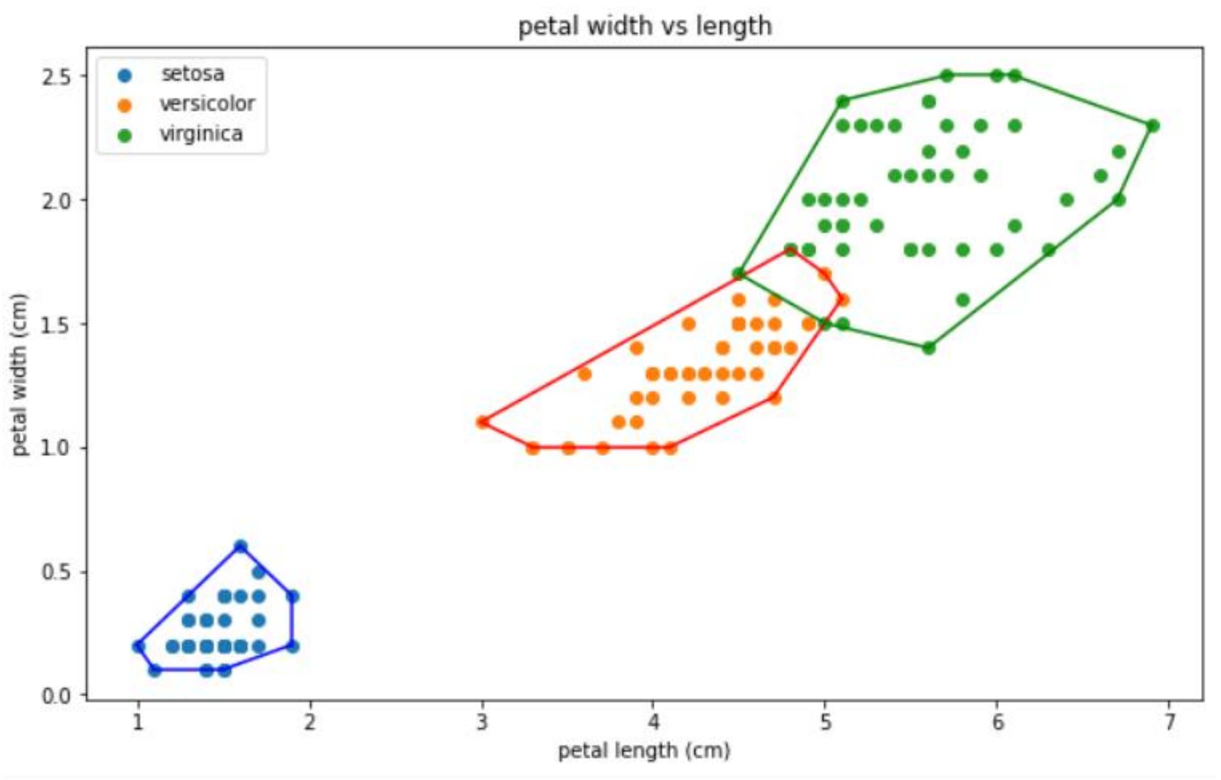
3. Mempartisi kedua zona sampai zona yang dipartisi tidak memiliki titik
4. Zona yang tidak memiliki titik akan dikembalikan hasilnya berupa dua titik pembentuk *convex hull*
5. Kumpulan titik pembentuk *convex hull* akan digabungkan membentuk *convex hull*.

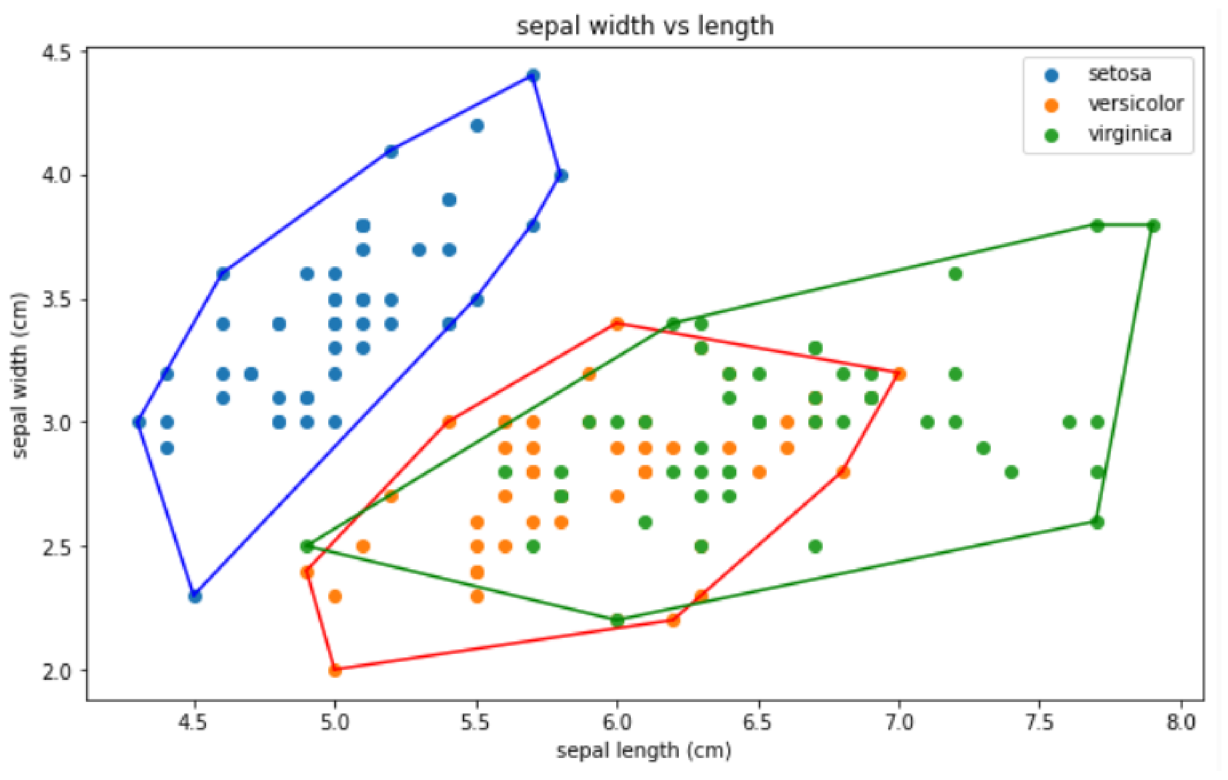
B. KODE PROGRAM

myConvexHull.py

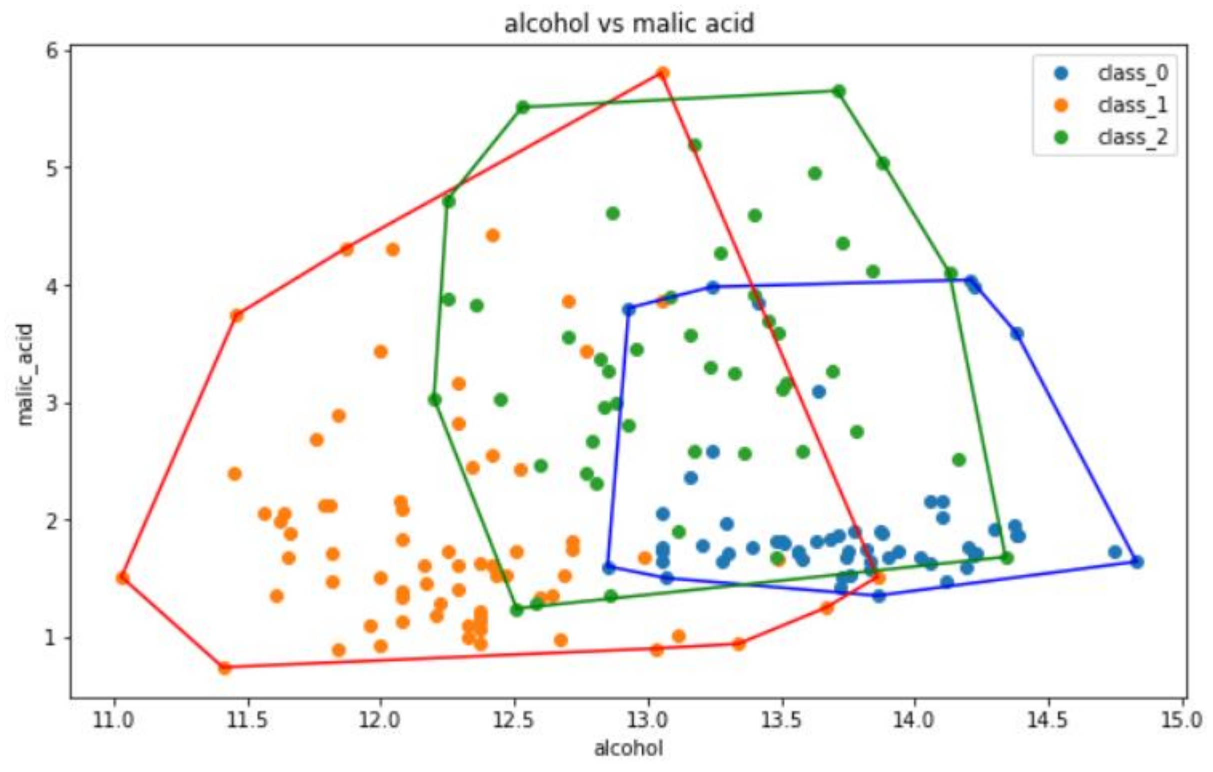
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math

# Constants for enumerations
X = 0
Y = 1

# Check if points are same or not
# fixed for floating points, accuracy: 0.001
def isSamePoint(point1, point2):
    diffX = abs(point1[X] - point2[X])
    diffY = abs(point1[Y] - point2[Y])
    # for integers, just == it
    samePoint = (point1[X] == point2[X] and point1[Y] == point2[Y])
    return samePoint or (diffX < 0.001 and diffY < 0.001)

# Get distance between two points
def getDistance(point1, point2):
    return math.sqrt((point1[X] - point2[X])**2 + (point1[Y] - point2[Y])**2)

# Get distance from a point to a line formed by two points
def getDistanceFromPointToLine(point1, point2, pointRef):
    a, b, c = getLinearEquationCoefficients(point1, point2)
    return abs(a * pointRef[X] + b * pointRef[Y] + c) / getDistance(point1,
point2)

# Get angle between two lines referring at one point
def getAngleBetweenLines(point1, point2, pointRef):
    beta = getDistance(pointRef, point1)
    gamma = getDistance(pointRef, point2)
    alpha = getDistance(point1, point2)
    if (beta != 0 and gamma != 0):
        cosine = (beta**2 + gamma**2 - alpha**2) / (2 * beta * gamma)
        return math.degrees(math.acos(cosine))
    else:
        return 0

# Get linear equation coefficient (a,b,c) from two points
# according to ax + by + c = 0
def getLinearEquationCoefficients(point1, point2):
    a = point2[Y] - point1[Y]
    b = point1[X] - point2[X]
    c = point2[X] * point1[Y] - point1[X] * point2[Y]
    return a, b, c

# Get leftmost and rightmost points of a set of points
def getLeftmostAndRightmostPoints(points):
    leftmost = points[0]
    rightmost = points[0]
    for i in range(1, len(points)):
        if points[i][X] < leftmost[X]:
            leftmost = points[i]
        if points[i][X] > rightmost[X]:
            rightmost = points[i]
    return leftmost, rightmost

# Get farthest point from a line formed by two points
def getFarthestPointFromLine(point1, point2, points):
    farthestIndex = 0
    farthestDistance = -999
    for i in range(len(points)):
        distance = getDistanceFromPointToLine(point1, point2, points[i])
        if distance > farthestDistance:
            farthestDistance = distance
            farthestIndex = i
        elif distance == farthestDistance:
            thisAngle = getAngleBetweenLines(point1, point2, points[i])
            farthestAngle = getAngleBetweenLines(point1, point2,
points[farthestIndex])
            if thisAngle > farthestAngle:
                farthestDistance = distance
                farthestIndex = i
    return points[farthestIndex]

# Divide points to two areas, left/above or right/below
def dividePoints(point1, point2, points):
    left = []
    right = []
    for point in points:
        if isSamePoint(point, point1) or isSamePoint(point, point2):
            continue

        x3 = point[X]
        y3 = point[Y]

        determinant = (point1[X] * point2[Y] + point2[X] * y3 + x3 * point1[Y])
- (point2[X] * point1[Y] + point1[X] * y3 + x3 * point2[Y])

        if determinant > 0:
            left.append(point)
        elif determinant < 0:
            right.append(point)
    return left, right

# Do partial convex hull
def partialConvexHull(leftpoint, rightpoint, points, category):
    # prepare hull points
    hullPoints = np.array([])

    # exclude points which are on the same line with leftpoint and rightpoint
    points = np.array([point for point in points if not isSamePoint(point,
leftpoint)])
    points = np.array([point for point in points if not isSamePoint(point,
rightpoint)])

    # divide points to two areas
    left, right = dividePoints(leftpoint, rightpoint, points)

    if (category == "above"):
        # check if left is empty
        if len(left) == 0:
            x1 = leftpoint[X]
            y1 = leftpoint[Y]
            x2 = rightpoint[X]
            y2 = rightpoint[Y]
            hullPoints = np.append(hullPoints, [[x1, x2], [y1, y2]])

        # else, partition again to two areas and get value
        else:
            farthestPoint = getFarthestPointFromLine(leftpoint, rightpoint,
left)
            hullPoints = np.append(hullPoints, partialConvexHull(leftpoint,
farthestPoint, left, "above"))
            hullPoints = np.append(hullPoints, partialConvexHull(farthestPoint,
rightpoint, left, "above"))

    elif (category == "below"):
        # check if right is empty
        if len(right) == 0:
            x1 = leftpoint[X]
            y1 = leftpoint[Y]
            x2 = rightpoint[X]
            y2 = rightpoint[Y]
            hullPoints = np.append(hullPoints, [[x1, x2], [y1, y2]])

        # else, partition again to two areas and get value
        else:
            farthestPoint = getFarthestPointFromLine(leftpoint, rightpoint,
right)
            hullPoints = np.append(hullPoints, partialConvexHull(leftpoint,
farthestPoint, right, "below"))
            hullPoints = np.append(hullPoints, partialConvexHull(farthestPoint,
rightpoint, right, "below"))

    hullPoints = np.reshape(hullPoints, (len(hullPoints) // 4, 2, 2))
    return hullPoints

# finally, do convex hull
def convexHull(leftpoint, rightpoint, points):
    hullPointsAbove = partialConvexHull(leftpoint, rightpoint, points, "above")
    hullPointsBelow = partialConvexHull(leftpoint, rightpoint, points, "below")
    hullPoints = np.vstack((hullPointsAbove, hullPointsBelow))
    return hullPoints

# Let's make the test
if __name__ == '__main__':
    points = 20 * np.random.rand(100,2)
    point1, point2 = getLeftmostAndRightmostPoints(points)
    hullPoints = convexHull(point1, point2, points)
    print(hullPoints)
    plt.plot(points[:,0], points[:,1], 'o')
    for pair in hullPoints:
        plt.plot(pair[0], pair[1])
    plt.show()
```

C. INPUT-OUTPUT

Petal-length vs petal-width (dari dataset iris)

Sepal-length vs sepal-width (dari dataset iris)



Alcohol vs malic acid (dari dataset wine)



D. ALAMAT GITHUB

https://github.com/dawetmaster/tucil-convex-hull

E. CHECK LIST

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