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

LAPORAN TUGAS KECIL

Diajukan Untuk Memenuhi Tugas Kecil IF2211 Strategi Algoritma

Semester II 2021/2022

Disusun oleh:

Jason Kanggara

13520080

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

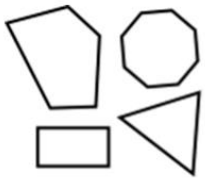
2022

A. Algoritma *Divide and Conquer*

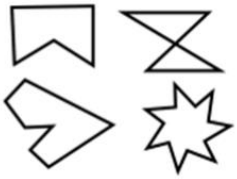
Algoritma *Divide and Conquer* adalah algoritma yang membagi suatu persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula, lalu menyelesaikan masing masing upa-persoalan secara langsung jika sudah kecil atau secara rekursif jika masih berukuran besar. Algoritma *Divide and Conquer* terbagi menjadi dua bagian, yaitu:

1. *Divide*: membagi persoalan menjadi beberapa persoalan yang lebih kecil.
2. *Conquer*: memproses upa-persoalan secara rekursif jika masih berukuran besar / langsung jika sudah berukuran kecil.

Convex hull merupakan salah satu hal penting dalam komputasi geometri. 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 poligon yang *convex* dan poligon yang *non-convex* dapat dilihat dari gambar berikut:



Gambar 1: convex



Gambar 2:non convex

Pada tugas kali ini, mahasiswa diminta untuk membuat suatu pustaka (*library*) dalam bahasa Python yang dapat mengembalikan *convex hull* dari kumpulan data 2 dimensi (dapat dianggap kumpulan titik 2 dimensi), dengan memanfaatkan algoritma *Divide and Conquer*. Pustaka tersebut nantinya akan digunakan dalam program visualisasi data. Adapun penjelasan algoritma *Divide and Conquer* yang digunakan dalam pencarian *convex hull* pada program ini adalah sebagai berikut:

1. Dari kumpulan titik 2 dimensi yang diperoleh dari *dataset*, lakukan pengurutan untuk mencari titik ekstrim dari kumpulan titik tersebut (misal **p1**, **pn**).
2. Garis yang menghubungkan **p1** dan **pn** membagi kumpulan titik menjadi dua bagian, yaitu **s1** (kumpulan titik di sebelah kiri/atas garis **p1pn**) dan **s2** (kumpulan titik di sebelah kanan/bawah garis **p1pn**).
3. Kedua kumpulan titik tersebut (**s1** dan **s2**) akan dibagi lagi menjadi lebih kecil hingga membentuk *convex hull*.
4. Untuk sebuah bagian (misal **s1**), terdapat dua kemungkinan:
    - Jika tidak ada titik lain pada **s1**, maka titik **p1** dan **pn** menjadi pembentuk *convex hull.*
    - Jika ada titik selain **p1** dan **pn**, maka cari titik terjauh dari garis **p1pn** (misal **pmax**). Hubungkan **pmax** dengan **p1pn** maka akan terbentuk 2 garis baru.

    p1, pn, dan **pmax** akan membentuk suatu segitiga. Titik titik yang terdapat di dalam segitiga akan diabaikan.
5. Dari 2 garis baru tersebut, kumpulkan titik yang berada di sebelah kiri garis **p1pmax** dan di sebelah kanan **pmaxpn**.
6. Ulangi langkah **4** dan **5** untuk bagian **s2** hingga bagian kiri dan kanan tidak ditemukan titik lagi.

7. Kembalikan pasangan titik yang dihasilkan.

B. Source Code

Bahasa pemrograman yang digunakan dalam pembuatan pustaka pencarian *convex hull* adalah Python

1. *Source Code* Implementasi Pustaka *myConvexHull*

```
import numpy as np

# Melakukan sorting array of points terurut membesar
# Algoritma yang digunakan untuk sorting adalah Merge Sort
def MergeSort(points):
  if len(points) > 1:
    idxMid = len(points) // 2
    leftArr = points[:idxMid]
    rightArr = points[idxMid:]

    # Pemanggilan rekursif
    MergeSort(leftArr)
    MergeSort(rightArr)

    # iterator
    i = 0
    j = 0
    k = 0

    while i < len(leftArr) and j < len(rightArr):
      if leftArr[i] <= rightArr[j]:
        points[k] = leftArr[i]
        i += 1
      else:
        points[k] = rightArr[j]
        j += 1

      k += 1

    while i < len(leftArr):
```

```
      points[k] = leftArr[i]
      i += 1
      k += 1

    while j < len(rightArr):
      points[k] = rightArr[j]
      j += 1
      k += 1

# Mengambil titik ekstrim dari list of points
# Karena sudah terurut,
# maka titik ekstrim cukup diambil dari indeks pertama dan terakhir
def getExtreme(points):
  p1 = points[0]
  pn = points[len(points) - 1]

  return p1, pn

# Mengambil lokasi suatu titik
# LOCATION: kiri/atas dan kanan/bawah
def getLocation(p1, p2, p3):
  value = (p1[0] * p2[1]) + (p3[0] * p1[1]) + (p2[0] * p3[1]) -
(p3[0] * p2[1]) - (p2[0] * p1[1]) - (p1[0] * p3[1])

  if value > 0: # Jika value positif, maka titik p3 di kiri/atas
garis p1,p2
    return 1
  elif value < 0: # jika value negatif, maka titik p3 di kanan/bawah
garis p1,p2
    return -1
  else:
    return 0

# Mencari jarak suatu titik terhadap garis
# USE CASE: mencari titik terjauh dari garis p1 pn
def getLineDistance(p1, p2, p3):
```

```
  value = (p1[0] * p2[1]) + (p3[0] * p1[1]) + (p2[0] * p3[1]) -
(p3[0] * p2[1]) - (p2[0] * p1[1]) - (p1[0] * p3[1])

  return abs(value)

# Rumus menghitung luas segitiga dengan perhitungan 3 titik
def area(p1, p2, p3):
  return abs((p1[0] * (p2[1] - p3[1]) + p2[0] * (p3[1] - p1[1]) +
p3[0] * (p1[1] - p2[1]) / 2))

# Memeriksa apakah suatu titik berada di dalam segitiga
def isInsideTriangle(p1, p2, p3, p):
  A = area(p1, p2, p3)
  A1 = area(p, p2, p3)
  A2 = area(p1, p, p3)
  A3 = area(p1, p2, p)

  if (A == A1 + A2 + A3):
    return True
  else:
    return False

# Mencari convex hull dengan algoritma divide and conquer
def findHull(hull, sn, p, q, simplices):
  if len(sn) == 0:
    return

  # inisiasi
  # pMax: titik terjauh
  # maxDist: jarak terjauh
  pMax = [0, 0]
  maxDist = 0

  # Mengupdate pMax dan maxDist
  for point in sn:
    if getLineDistance(p, q, point) > maxDist:
      maxDist = getLineDistance(p, q, point)
```

```
    pMax = point

# Prosedur penghapusan agar tidak ada titik duplikat
sn.remove(pMax)

if p in sn:
  sn.remove(p)
if q in sn:
  sn.remove(q)

# Memasukkan pMax ke list of hull
hull.append(pMax)

# Membagi kumpulan titik menjadi 2 bagian
# s1 dan s2
s1 = []
s2 = []

# Algoritma pengecekan titik
for x in sn:
  if isInsideTriangle(p, pMax, q, x):
    sn.remove(x)
  if (not isInsideTriangle(p, pMax, q, x)):
    if getLocation(p, pMax, x) > 0:
      s1.append(x)
    if getLocation(pMax, q, x) > 0:
      s2.append(x)

# Rekursif fungsi findHull
# Membagi sampai menjadi kecil
findHull(hull, s1, p, pMax, simplices)
findHull(hull, s2, pMax, q, simplices)

# Masukkan point yang berhubungan ke list of simplices
simplices.append([p, pMax])
simplices.append([pMax, q])
```

```
# Mencari titik titik yang saling berhubungan
# atau membentuk suatu garis
def getConnectedSimplex(simplices, hull):
  new_simplices = []
  new_simplices.append(simplices[0])

  for i in range(len(hull) - 1):
    for j in range(len(simplices)):
      if simplices[j][0] == new_simplices[-1][1]:
        new_simplices.append(simplices[j])
        break

  return new_simplices

# Mengubah point menjadi index agar dapat melakukan plotting pada
notebook
def convertPointToIndex(simplices, points):
  for i in range(len(simplices)):
    for j in range(len(points)):
      if simplices[i][0] == points[j]:
        simplices[i][0] = j
      if simplices[i][1] == points[j]:
        simplices[i][1] = j

# Fungsi MAIN
def myConvexHull(points):
  convert_points = np.ndarray.tolist(points)
  sorted_points = np.ndarray.tolist(points)

  # Lakukan pengurutan titik titik secara menaik
  MergeSort(sorted_points)

  # Ambil titik ekstrim inisial
  p1, pn = getExtreme(sorted_points)

  hull = [] # Inisiasi list of points yang membentuk convex hull
```

```
  simplices = [] # Inisiasi list of simplex (point yang saling
berhubungan)

  # masukkan titik ekstrim ke hull
  hull.append(p1)
  hull.append(pn)

  # bagi kumpulan titik menjadi 2 bagian
  s1 = []
  s2 = []

  for point in sorted_points:
    if getLocation(p1, pn, point) > 0:
      s1.append(point) # kumpulan titik sisi atas
    if getLocation(p1, pn, point) < 0:
      s2.append(point) # kumpulan titik sisi bawah

  # Jalankan fungsi findHull berdasarkan list s1 dan s2
  # Membagi pencarian menjadi 2 bagian (Divide)
  findHull(hull, s1, p1, pn, simplices)
  findHull(hull, s2, pn, p1, simplices)

  simplices = getConnectedSimplex(simplices, hull)

  convertPointToIndex(simplices, convert_points)

  return np.array(simplices)
```

2. *Source Code* Implementasi pada Jupyter Notebook (Contoh: *Dataset Iris*)

Dataset Iris

```python
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)
df.head()
```

Python

```
(150, 5)
```

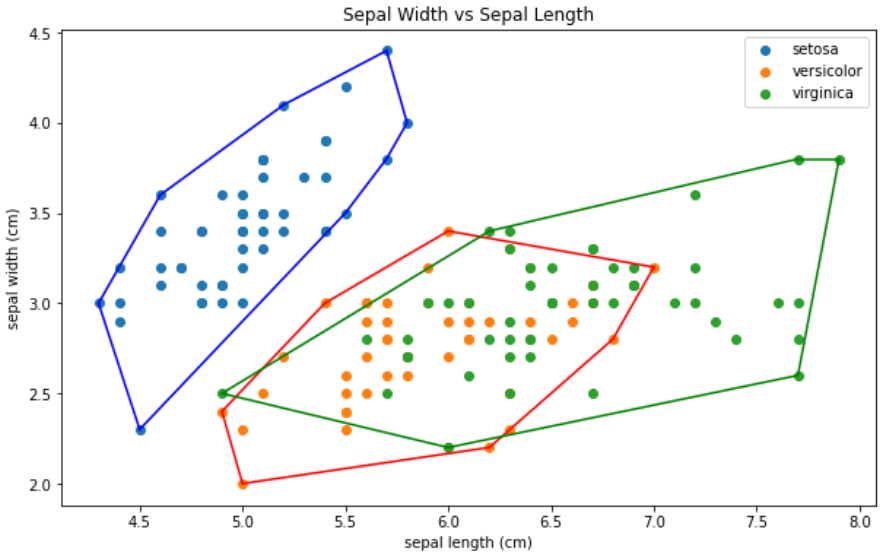
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

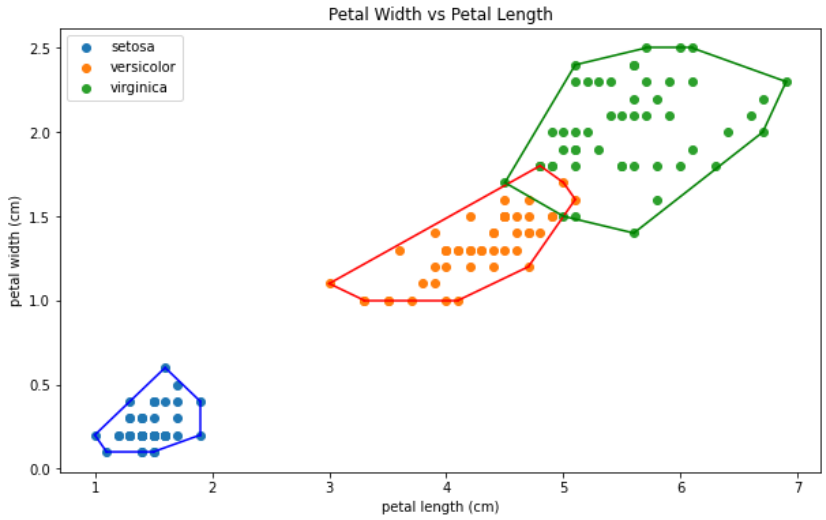
Sepal Width vs Sepal Length

```python
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(bucket)
  plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])

  for simplex in hull:
    plt.plot(bucket[simplex, 0], bucket[simplex, 1], colors[i])

plt.legend()
```

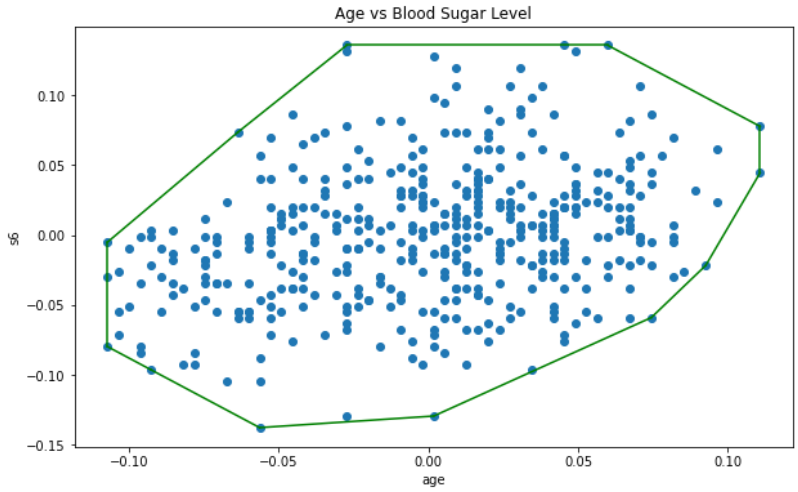
Python

C. Hasil Program

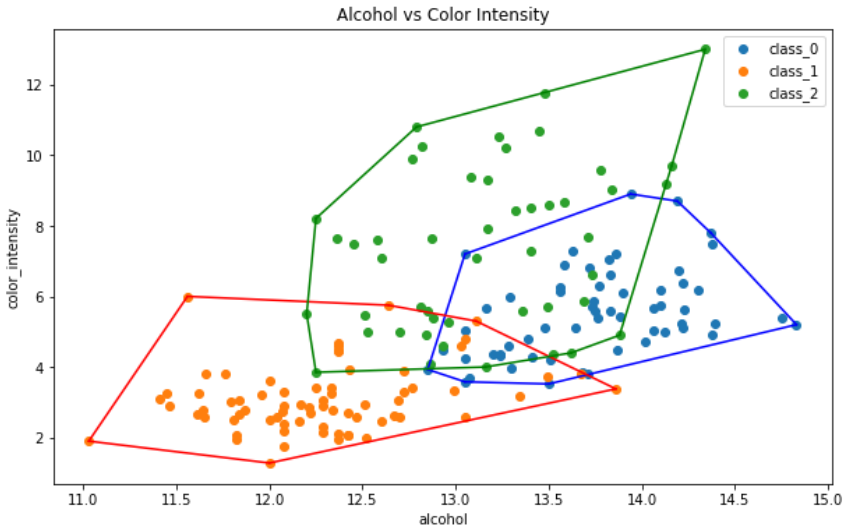
Dataset Iris: Sepal Width vs Sepal Length


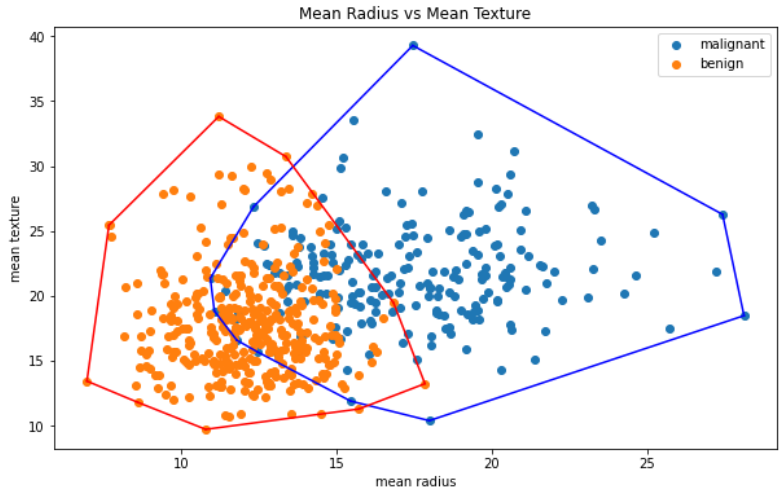
Dataset Iris: Petal Width vs Petal Length


Dataset Diabetes



Dataset Alcohol



Dataset Breast Cancer


D. Link Source Code

https://github.com/jasonk19/Tucil-Stima-2.git

E. Tabel Cek List

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

* *Bagian **BONUS**, dataset lainnya dari sklearn*