Laporan Tugas Kecil 2

13520017 / Diky Restu Maulana

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.	✓	

1. Algoritma Divide and Conquer

Langkah-langkah yang dilakukan untuk menemukan kumpulan titik yang membentuk *convex hull* berdasarkan algoritma *Divide and Conquer* sebagai berikut

1. *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 kartesian dua dimensi. Dalam implementasinya, S dibuat dalam sebuah array berdimensi $n \times 2$.
2. Urutkan array *S* berdasarkan nilai absis menaik. Jika nilai absisnya sama, urutkan berdasarkan nilai ordinat menaik.
3. Ada dua kemungkinan pada array *S*, yaitu
   - Jika hanya terdapat dua titik pada *S*, maka garis yang menghubungkan kedua titik adalah pembentuk *convex hull*. Kembalikan kedua titik tersebut.
   - Jika terdapat lebih dari dua titik, selanjutnya ambil elemen pertama p_1 (absis terkecil/titik paling kiri) dan elemen terakhir p_n (absis terbesar/titik paling kanan) dari array *S* yang sudah diurutkan.
4. Garis yang menghubungkan p_1 dan p_n (p_1p_n) membagi *S* menjadi dua segmen, yaitu *above* (kumpulan titik di sebelah kiri atau atas garis p_1p_n) dan *below* (kumpulan titik di sebelah kanan atau bawah garis p_1p_n).
5. Semua titik pada *S* yang berada pada garis p_1p_n (selain titik p_1 dan p_n) tidak mungkin membentuk *convex hull* sehingga bisa diabaikan.
6. Kumpulan titik pada *above* bisa membentuk *convex hull* bagian atas dan kumpulan titik pada *below* dapat membentuk *convex hull* bagian bawah.
7. Untuk sebuah segmen (misalnya *above*), terdapat dua kemungkinan:
   - Jika tidak ada titik di segmen *above*, maka titik p_1 dan p_n menjadi pembentuk *convex hull* bagian *above*.
   - Jika *above* tidak kosong, pilih sebuah titik yang memiliki jarak terjauh dari garis p_1p_n (misalnya p_{max}). Jika terdapat beberapa titik dengan jarak yang sama, pilih sebuah titik yang memaksimalkan sudut $p_{max}p_1p_n$.

8. Semua titik yang berada di dalam segitiga $p_{max}p_1p_n$ dapat diabaikan pada pemeriksaan selanjutnya karena tidak mungkin membentuk *convex hull*.
9. Tarik garis p_1p_{max} dan $p_{max}p_n$. Lalu, tentukan *above1*, yaitu kumpulan titik pada *above* yang berada di sebelah kiri atau atas garis p_1p_{max} dan *above2*, yaitu kumpulan titik pada *above* yang berada di sebelah kiri atau atas garis $p_{max}p_n$.
10. Lakukan hal yang sama (poin 4 dan 5) untuk segmen *below*, hingga tidak ada lagi titik yang berada di luar segitiga.
11. Kembalikan pasangan titik yang dihasilkan.

2. Source Code Program dalam Bahasa Python

Pustaka myConvexHull

Pustaka dapat dibagi menjadi beberapa fungsi, yaitu

1. ConvexHull
   Fungsi utama untuk mengembalikan hasil akhir berupa kumpulan titik yang membentuk *convex hull*.

```python
# Fungsi utama berparameter points (array of point) dan mengembalikan titik-titik pembentuk Convex Hull
def ConvexHull(points):
    # BASIS: Jika hanya ada dua titik, kembalikan keduanya
    if (len(points) <= 2):
        return points

    # Lakukan pengurutan membesar berdasarkan nilai x
    # Jika nilai x sama, urutkan nilai y membesar
    sort = sorted(points, key=lambda x:(x[0], x[1]))

    # Ambil titik paling kiri dan paling kanan
    p1 = sort[0]
    p2 = sort[-1]

    # Kedua titik menjadi pembentuk Convex Hull
    hull = [p1, p2]

    # Hapus kedua titik dari kumpulan titik agar tidak diperiksa lagi
    sort.pop(0)
    sort.pop(-1)

    # DIVIDE
    # Titik p1 dan p2 membentuk garis
    # Pisahkan titik-titik yang berada di atas dan bawah garis tersebut
    above, below = createSegment(p1, p2, sort)

    # CONQUER
    # Gabungkan hasil pemeriksaan segmen atas dan bawah
    hull += findHull(p1, p2, above, "above")
    hull += findHull(p1, p2, below, "below")

    # Menghapus titik yang muncul dua kali/duplikat
    hull = (np.unique(np.asarray(hull), axis=0)).tolist()

    # Mengurutkan titik untuk keperluan plotting
    hull = sortHull(hull)

    return hull
```

2. findHull
   Fungsi yang akan dipanggil secara rekursif sebagai implementasi algoritma *divide and conquer*. Fungsi ini mengembalikan kumpulan titik yang membentuk *convex hull* di segmen atas dan bawah.

```python
# Fungsi sampingan untuk memeriksa setiap segmen
# p1p2 adalah garis yang akan menjadi acuan
# segment adalah kumpulan titik yang akan diperiksa
# flag bernilai "above" atau "below"
def findHull(p1, p2, segment, flag):
    # Jika tidak ada titik di dalam segmen, kembalikan array kosong
    if (len(segment) == 0):
        return []

    # Inisialisasi
    hull = []
    farthest_distance = -1
    farthest_point = None

    # Mencari titik dengan jarak terjauh terhadap garis p1p2
    for point in segment:
        distance = findDistance(p1, p2, point)
        if(distance > farthest_distance):
            farthest_distance = distance
            farthest_point = point

    # Titik terjauh pasti akan membentuk Convex Hull
    hull += [farthest_point]

    # Menghapus titik dari segment agar tidak diperiksa lagi
    segment.remove(farthest_point)
    # Membuat segmen baru berdasarkan garis p1-farthest_point dan farthest_point-p2
    p1a, p1b = createSegment(p1, farthest_point, segment)
    p2a, p2b = createSegment(p2, farthest_point, segment)

    # Jika segmen yang diperiksa adalah segmen atas, terus menerus akan diperiksa segmen bagian atas
    # Begitupun sebaliknya
    if flag == "above":
        hull += findHull(p1, farthest_point, p1a, "above")
        hull += findHull(farthest_point, p2, p2a, "above")
    else:
        hull += findHull(p1, farthest_point, p1b, "below")
        hull += findHull(farthest_point, p2, p2b, "below")

    return hull
```

3. gradien
   Fungsi untuk menghitung kemiringan garis yang dibentuk oleh dua titik.

```python
# Fungsi untuk menghitung gradien garis p1p2
def gradien(p1, p2):
    return (p1[1] - p2[1]) / (p1[0] - p2[0])
```

4. constant
   Fungsi untuk menghitung konstanta c pada persamaan garis.

```python
# Fungsi untuk menghitung konstanta pada persamaan garis p1p2
def constant(p1, p2):
    m = gradien(p1, p2)
    return (p1[1] - m*p1[0])
```

5. findDistance
   Fungsi untuk menghitung jarak terpendek antara titik dan garis.

```python
# Fungsi untuk menghitung jarak terpendek antara garis p1p2 dan titik point
def findDistance(p1, p2, point):
    a = p1[1]-p2[1]
    b = p2[0]-p1[0]
    c = p1[0]*p2[1]-p2[0]*p1[1]
    return abs((a*point[0] + b*point[1] + c) / ((a*a + b*b)**0.5))
```

6. createSegment
   Fungsi untuk membagi kumpulan titik menjadi dua segmen, yaitu atas dan bawah yang dipisahkan oleh garis yang dibentuk oleh dua titik.

```python
# Fungsi untuk membagi kumpulan titik menjadi dua segmen (atas dan bawah) berdasarkan garis p1p2
def createSegment(p1, p2, points):
    above = []
    below = []

    # Jika garis yang dibentuk vertikal, tidak ada atas dan bawah
    # Sekaligus menghindari gradien yang tidak terdefinisi (pembagian dengan nol)
    if(p1[0] - p2[0] == 0):
        return above, below

    m = gradien(p1, p2)
    c = constant(p1, p2)

    for p in points:
        if (p[1] > m*p[0] + c):
            above.append(p)
        elif (p[1] < m*p[0] + c):
            below.append(p)
    return above,below
```

7. sortHull
   Fungsi untuk mengurutkan kumpulan titik secara melingkar berlawanan arah jarum jam dari titik paling kiri ke titik untuk keperluan plotting.

```python
# Fungsi untuk mengurutkan titik untuk keperluan plotting
def sortHull(hull):
    above, below = createSegment(hull[0], hull[-1], hull)
    newHull = []
    newHull.append(hull[0])
    newHull += below
    newHull.append(hull[-1])
    above.reverse()
    newHull += above
    return newHull
```

3. Input-Output Program

Visualisasi Data

a. Dataset Iris

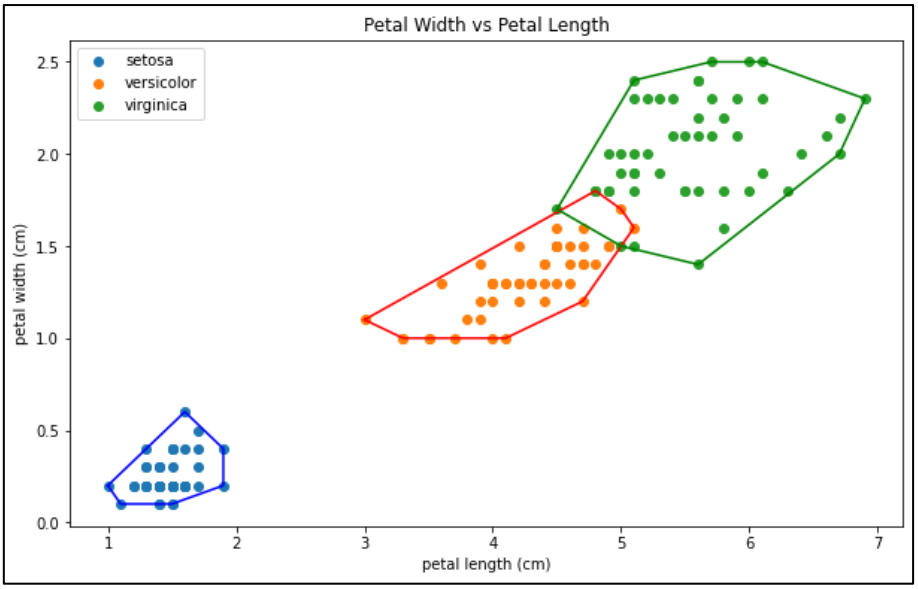
```python
data_iris = datasets.load_iris()

# Membuat dataframe
df = pd.DataFrame(data_iris.data, columns=data_iris.feature_names)
df['Target'] = pd.DataFrame(data_iris.target)

# Visualisasi hasil ConvexHull
plt.figure(figsize = (10, 6))

plt.title('Petal Width vs Petal Length')
plt.xlabel(data_iris.feature_names[2])
plt.ylabel(data_iris.feature_names[3])

for i in range(len(data_iris.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[2,3]].values
    bucket = bucket.tolist()
    hull = ConvexHull(bucket)
    bucket = np.asarray(bucket)
    hull.append(hull[0])
    xs, ys = zip(*hull)
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data_iris.target_names[i])
    plt.plot(xs, ys, colors[i])
    plt.legend()
```

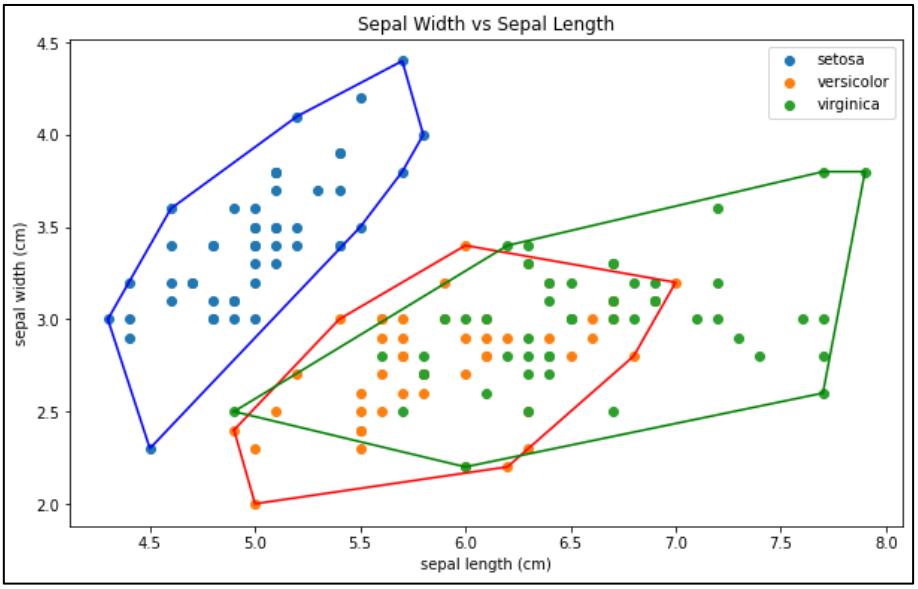



```python
# Visualisasi hasil ConvexHull
plt.figure(figsize = (10, 6))

plt.title('Sepal Width vs Sepal Length')
plt.xlabel(data_iris.feature_names[0])
plt.ylabel(data_iris.feature_names[1])

for i in range(len(data_iris.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[0,1]].values
    bucket = bucket.tolist()
    hull = ConvexHull(bucket)
    bucket = np.asarray(bucket)
    hull.append(hull[0])
    xs, ys = zip(*hull)
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data_iris.target_names[i])
    plt.plot(xs, ys, colors[i])
    plt.legend()
```

✓ 0.4s Python



b. Dataset digits

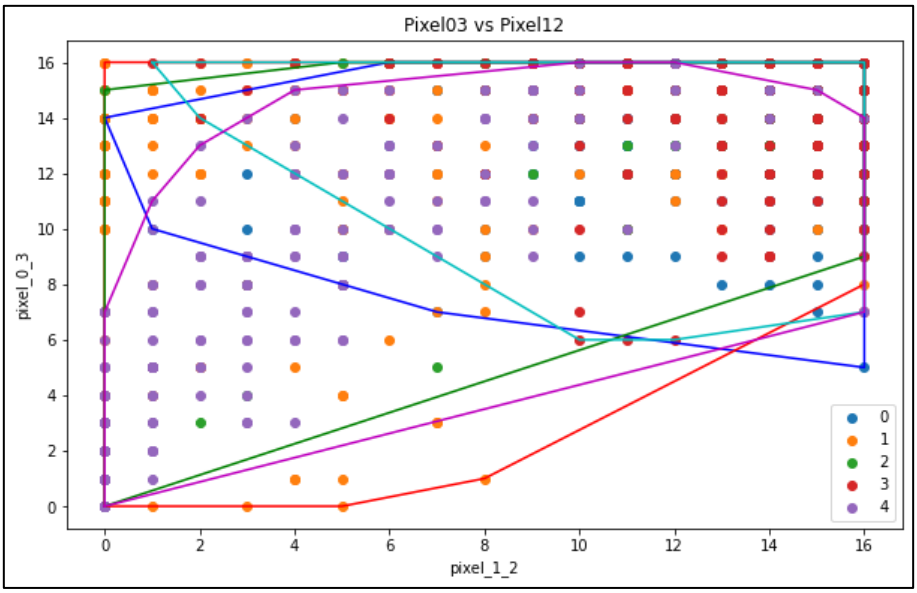
```
data_digits = datasets.load_digits()

# Membuat dataframe
df = pd.DataFrame(data_digits.data, columns=data_digits.feature_names)
df['Target'] = pd.DataFrame(data_digits.target)

# Visualisasi hasil ConvexHull
plt.figure(figsize = (10, 6))

plt.title('Pixel03 vs Pixel12')
plt.xlabel(data_digits.feature_names[10])
plt.ylabel(data_digits.feature_names[3])

for i in range(len(data_digits.target_names)-5):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[10,3]].values
    bucket = bucket.tolist()
    hull = ConvexHull(bucket) #bagian ini diganti dengan hasil implementasi ConvexHull Divide & Conquer
    bucket = np.asarray(bucket)
    hull.append(hull[0])
    xs, ys = zip(*hull)
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data_digits.target_names[i])
    plt.plot(xs, ys, colors[i])
    plt.legend()
```



c. Dataset Wine

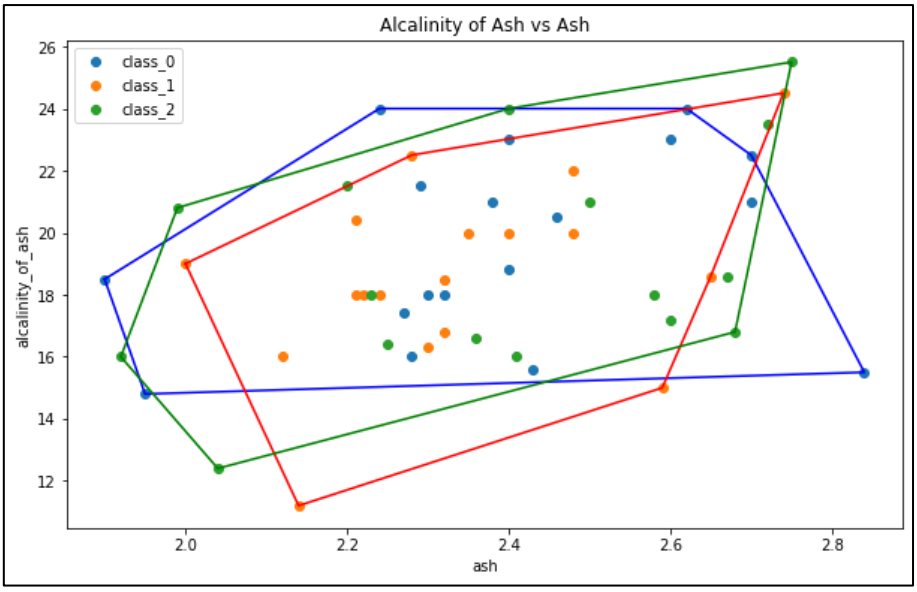
```
data_wine = datasets.load_wine()

# Membuat dataframe
df = pd.DataFrame(data_wine.data, columns=data_wine.feature_names)
df['Target'] = pd.DataFrame(data_digits.target)

# Visualisasi hasil ConvexHull
plt.figure(figsize = (10, 6))

plt.title('Alcalinity of Ash vs Ash')
plt.xlabel(data_wine.feature_names[2])
plt.ylabel(data_wine.feature_names[3])

for i in range(len(data_wine.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[2,3]].values
    bucket = bucket.tolist()
    hull = ConvexHull(bucket) #bagian ini diganti dengan hasil implementasi ConvexHull Divide & Conquer
    bucket = np.asarray(bucket)
    hull.append(hull[0])
    xs, ys = zip(*hull)
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data_wine.target_names[i])
    plt.plot(xs, ys, colors[i])
    plt.legend()
```



d. Dataset Breast Cancer

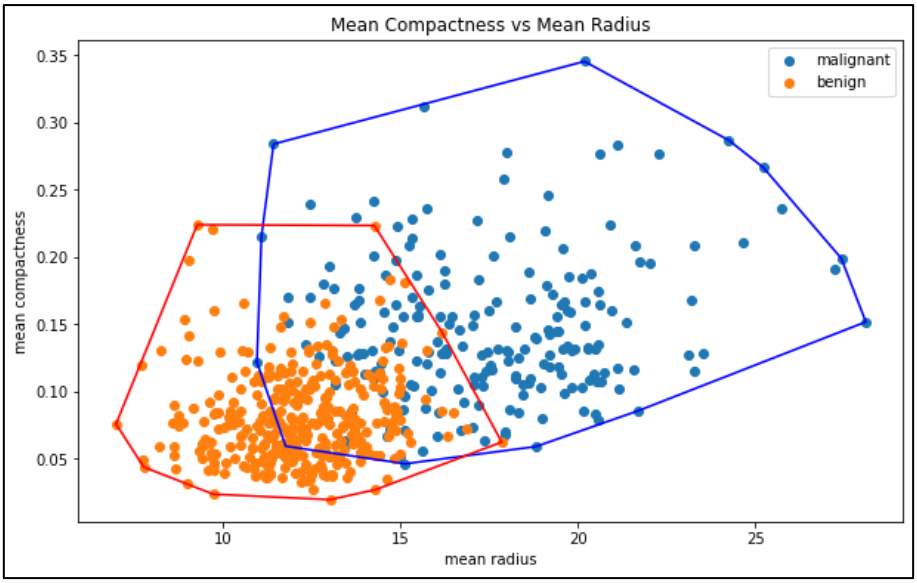
```
data_cancer = datasets.load_breast_cancer()

# Membuat dataframe
df = pd.DataFrame(data_cancer.data, columns=data_cancer.feature_names)
df['Target'] = pd.DataFrame(data_cancer.target)

# Visualisasi hasil ConvexHull
plt.figure(figsize = (10, 6))

plt.title('Mean Compactness vs Mean Radius')
plt.xlabel(data_cancer.feature_names[0])
plt.ylabel(data_cancer.feature_names[5])

for i in range(len(data_cancer.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[0,5]].values
    bucket = bucket.tolist()
    hull = ConvexHull(bucket) #bagian ini diganti dengan hasil implementasi ConvexHull Divide & Conquer
    bucket = np.asarray(bucket)
    hull.append(hull[0])
    xs, ys = zip(*hull)
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data_cancer.target_names[i])
    plt.plot(xs, ys, colors[i])
    plt.legend()
```



4. Tautan Github

https://github.com/dikyrest/Tucil_2_Stima_2022.git