Laporan Tugas Kecil 2 STIMA

Rifqi Naufal Abdjul
13520062

Algoritma Divide and Conquer

Dalam penyusunan algoritma program ini menggunakan pendekatan divide and conquer dengan maksud membagi permasalahan yang besar menjadi banyak permasalahan kecil yang lalu akan diselesaikan secara efisien, yang selanjutnya akan digabung untuk menyelesaikan permasalahan besar tersebut. Terdapat beberapa bagian dari algoritma ini, yang dijelaskan secara detail di bawah ini:

1. Object ConvexHull

a. Struktur Objek
   Objek ConvexHull menyimpan 2 properties, yaitu points dan hull. Dimana points merupakan seluruh titik yang disuplai ke dalam objek ConvexHull dan hull merupakan kumpulan garis yang membentuk hull terkecil yang memuat seluruh titik yang disuplai ke dalam objek.

b. Fungsi static get_hull(points)
   Fungsi ini merupakan fungsi utama untuk mendapatkan hull yang akan disimpan pada properties hull. Fungsi ini membutuhkan fungsi private pembantunya yaitu __get_top_hull dan __get_bot_hull untuk melakukan divide and conquer. Proses algoritma fungsi adalah sebagai berikut:
   1. Melakukan sorting terhadap titik berdasarkan nilai x agar urutan penggarisan dan pengambilan nilai minimum serta maksimum mudah untuk dilakukan
   2. Menarik garis antara titik paling awal dan akhir sehingga membagi daerah menjadi 2
   3. Memisahkan daerah di atas garis tersebut dan daerah di bawah garis tersebut. Lalu melakukan pemanggilan fungsi pembantu __get_top_hull dan __get_bot_hull sesuai daerah
   4. Setelah mendapatkan seluruh titik hull, kumpulan titik tersebut akan dibentuk dan diformat agar berbentuk garis agar mudah direpresentasikan dalam grafik

c. Fungsi private __get_top_hull(points) dan __get_bot_hull(points)
   Fungsi ini merupakan fungsi pembantu dari fungsi utama get_hull. Fungsi ini bersifat rekursif menggunakan pendekatan divide and conquer yang termasuk bagian merging juga. Proses algoritma fungsi adalah sebagia berikut:
   1. Mendapatkan titik awal dan akhir dalam array points
   2. Mendapatkan titik tertinggi/terendah yang merupakan titik yang mempunyai jumlah jarak terbesar dari titik awal dan akhir

3. Membuang titik yang berada di bawah garis antara titik awal, tertinggi/terendah, dan titik akhir
4. Melakukan pemanggilan fungsi kembali pada tiap bagian (bagian di atas titik awal dan tertinggi/terendah, dan bagian di atas titik tertinggi/terendah dan titik akhir), sesuai dengan titik yang memenuhi kondisi bagian tersebut.
5. Jika jumlah titik hanya 2 atau kurang, maka diberhentikan dan mulai dilakukan merging dengan menambahkan seluruh titik yang tidak terbuang.

2. Quicksort

Quicksort ini digunakan pada awal fungsi untuk mengurutkan titik berdasarkan nilai x nya. Jika nilai x sama, maka akan dibandingkan berdasarkan nilai y. Proses algoritma quicksort ini dijelaskan pada poin berikut ini.

1. Mengambil pivot yang merupakan titik pertama dari array titik
2. Melakukan filtering dan menaruh seluruh titik yang lebih rendah perbandingannya di bagian kiri, dan seluruh titik yang lebih tinggi perbandingannya di bagian kanan.
3. Memanggil fungsi quicksort kembali dengan array bagian kiri dan bagian kanan hingga ukuran array lebih kecil atau sama dengan 1

3. Extras

a. Fungsi get_distance(p1, p2)
   Merupakan fungsi yang mengembalikan jarak antar 2 titik menggunakan rumus pytagoras.
b. Fungsi isAbovetOrBelow(ref1,ref2, point)
   Merupakan fungsi yang mengembalikan “Above”, “On”, atau “Below” sesuai dari posisi point terhadap garis yang dibentuk oleh titik ref1, dan ref2

Kode Program

```
# %% [markdown]
# # ConvexHull Tugas Kecil STIMA 2
# > Made by Rifqi Naufal Abdjul (13520062)

# %% [markdown]
# ## Import Libraries

# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets

# %% [markdown]
# ### ConvexHull Libraries
# On comparing other convex Hull, You can use the scipy library or the
hand-built one

# %%
# Scipy library
# from scipy.spatial import ConvexHull

# %%
import numpy as np

# Fungsi quicksort yang diubah sedikit algoritma dari quicksort biasa agar
lebih efisien


def quicksort(arr, key=lambda x: x):
    if len(arr) <= 1:
        return arr
    pivot = key(arr[0])
    left = [x for x in arr[1:] if key(x) <= pivot]
    right = [x for x in arr[1:] if key(x) >= pivot]
    return quicksort(left, key) + [arr[0]] + quicksort(right, key)

# Fungsi untuk mencari jarak antar 2 point
```

```
def get_distance(p1, p2):
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# Predikat untuk menentukan apakah point di atas atau dibawah garis yang
dibentuk ref1,ref2
# Ref1[0] selalu lebih kecil daripada Ref2[0]


def isAboveOrBelow(ref1, ref2, point):
    # Jika garis referensi perfectly vertical
    if (ref1[0] - ref2[0] == 0):
        if (point[0] < ref1[0]):
            return "Above"
        elif (point[0] > ref1[0]):
            return "Below"
        else:
            return "On"
    grad = (ref1[1] - ref2[1]) / (ref1[0] - ref2[0])
    # Merupakan nilai x yang di dalam garis
    yline = ref1[1] + grad * (point[0] - ref1[0])
    if point[1] > yline:
        return "Above"
    elif point[1] < yline:
        return "Below"
    else:
        return "On"

# Kelas ConvexHull yang dapat di gunakan untuk mendapatkan convex hull
dari data


class ConvexHull:
    # Constructor, dengan anggapan points merupakan array dari points
(array of (X, Y))
    def __init__(self, points):
        self.points = np.array(points)
        self.hull = np.array(ConvexHull.get_hull(self.points))
```

```
    @staticmethod
    # Dianggap points yang sudah masuk telah di sort oleh fungsi get_hull
    def __get_top_hull(points):
        if len(points) <= 2:
            return points
        first = points[0]
        last = points[-1]
        # Mencari titik yang paling atas

        top = max(points, key=lambda x: get_distance(
            x, first) + get_distance(x, last))

        # Left Top merupakan titik yang di atas titik first, dan top
        # Right Top merupakan titik yang di atas titik top, dan last
        left_top = [first]
        right_top = [top]
        for i in range(1, len(points) - 1):
            if (isAboveOrBelow(top, first, points[i]) == "Above"):
                left_top.append(points[i])
            if (isAboveOrBelow(top, last, points[i]) == "Above"):
                right_top.append(points[i])
        left_top.append(top)
        right_top.append(last)

        # Mengembalikan hull dari kiri dan kanan
        return ConvexHull.__get_top_hull(left_top) +
ConvexHull.__get_top_hull(right_top)

    @staticmethod
    # Dianggap points yang sudah masuk telah di sort oleh fungsi get_hull
    def __get_bottom_hull(points):
        if len(points) <= 2:
            return points
        first = points[0]
        last = points[-1]
        # Mencari titik yang paling atas

        bot = max(points, key=lambda x: get_distance(
            x, first) + get_distance(x, last))
```

```
        # Left Bot merupakan titik yang di bawah titik first, dan bot
        # Right Bot merupakan titik yang di bawah titik bot, dan last
        left_bot = [first]
        right_bot = [bot]
        for i in range(1, len(points) - 1):
            if (isAboveOrBelow(bot, first, points[i]) == "Below"):
                left_bot.append(points[i])
            if (isAboveOrBelow(bot, last, points[i]) == "Below"):
                right_bot.append(points[i])
        left_bot.append(bot)
        right_bot.append(last)

        # Mengembalikan hull dari kiri dan kanan
        return ConvexHull.__get_bottom_hull(left_bot) +
ConvexHull.__get_bottom_hull(right_bot)

    @staticmethod
    # Merupakan fungsi utama yang mencari hull dari points
    def get_hull(points):
        # Mengurutkan points berdasarkan x
        points = quicksort(points, key=lambda x: (x[0],x[1]))

        # Mendapatkan titik pertama dan terakhi sebagai acuan
        first = points[0]
        last = points[-1]

        # Mendapatkan Hull atas dan Bawah secara Divide and Conquer
        top = [first] + [x for x in points[1:-1]
                         if isAboveOrBelow(first, last, x) == "Above"] +
[last]
        bottom = [first] + [x for x in points[1:-1]
                            if isAboveOrBelow(first, last, x) == "Below"]
+ [last]
        res = (ConvexHull.__get_top_hull(
            top))[::-1] + ConvexHull.__get_bottom_hull(bottom)

        # Merapihkan Hull menjadi bentuk simplices agar lebih mudah untuk
di gambarkan
        hull_points = [x for idx, x in enumerate(res) if idx % 2 == 0]
        simplices = []
```

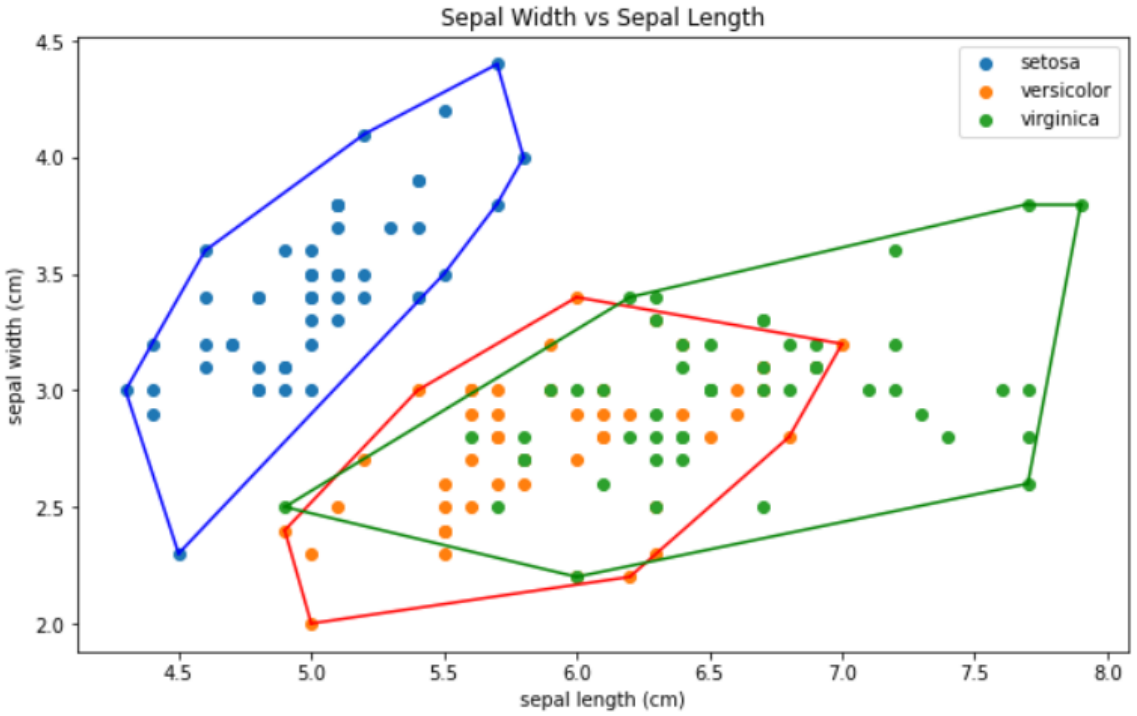
```
        for i in range(len(hull_points)):
            simplices.append([hull_points[i-1], hull_points[i]])
        return simplices


# %% [markdown]
# ## Testing the Libraries

# %% [markdown]
# ### Data 1 (Iris Datasets)

# %%
# Load and inspect the data
data = datasets.load_iris()

# Import into a dataframe
df = pd.DataFrame(data.data, columns=data.feature_names)
df["Target"] = data.target
print("Data size:",df.shape)
df.head()


# %%
# Visualize the datasets

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 = ConvexHull(bucket)
  plt.scatter(hull.points[:, 0], hull.points[:, 1],
label=data.target_names[i])
  for k in range(len(hull.hull)):
    plt.plot(hull.hull[k][:,0], hull.hull[k][:,1], colors[i])
plt.legend()
plt.show()
```

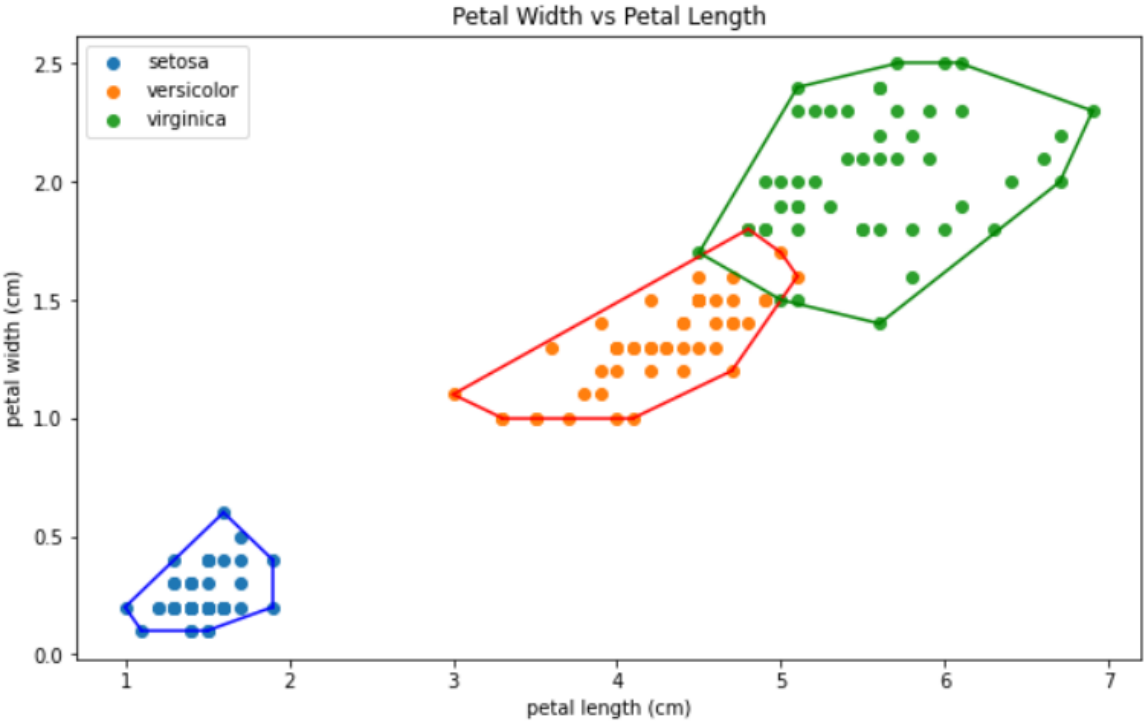
```
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Petal Width vs Petal Length')
plt.xlabel(data.feature_names[2])
plt.ylabel(data.feature_names[3])
for i in range(len(data.target_names)):
  bucket = df[df['Target'] == i]
  bucket = bucket.iloc[:,[2,3]].values
  hull = ConvexHull(bucket)
  plt.scatter(hull.points[:, 0], hull.points[:, 1],
label=data.target_names[i])
  for k in range(len(hull.hull)):
    plt.plot(hull.hull[k][:,0], hull.hull[k][:,1], colors[i])
plt.legend()
plt.show()


# %% [markdown]
# ### Data 2 (Wine Datasets)

# %%
# Load and inspect the data
data = datasets.load_wine()

# Import into a dataframe
df = pd.DataFrame(data.data, columns=data.feature_names)
df["Target"] = data.target
print("Data size:",df.shape)
df.head()

# %%
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Alcohol vs Malic Acid')
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 = ConvexHull(bucket)
  plt.scatter(hull.points[:, 0], hull.points[:, 1],
label=data.target_names[i])
  for k in range(len(hull.hull)):
    plt.plot(hull.hull[k][:,0], hull.hull[k][:,1], colors[i])
plt.legend()
plt.show()

plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Total Phenols vs Flavanoids')
plt.xlabel(data.feature_names[5])
plt.ylabel(data.feature_names[6])
for i in range(len(data.target_names)):
  bucket = df[df['Target'] == i]
  bucket = bucket.iloc[:,[5,6]].values
  hull = ConvexHull(bucket)
  plt.scatter(hull.points[:, 0], hull.points[:, 1],
label=data.target_names[i])
  for k in range(len(hull.hull)):
    plt.plot(hull.hull[k][:,0], hull.hull[k][:,1], colors[i])
plt.legend()
plt.show()
```

Screenshot Output

Dataset Iris





Dataset Diabetes