**TUGAS KECIL 2 IFF2211 STRATEGI ALGORITMA**

**IMPLEMENTASI CONVEX HULL UNTUK VISUALISASI TES LINEAR SEPARABILITY DATASET DENGAN ALGORITMA DIVIDE AND CONQUER**

**Logo

Description automatically generated**

Disusun oleh

Christine Hutabarat (13520005)

**TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG 2022**

**DAFTAR ISI**

[I. ALGORITMA DIVIDE AND CONQUER 2](#_Toc96948681)

[II. IMPLEMENTASI 3](#_Toc96948682)

[STRUKTUR KELAS 3](#_Toc96948683)

[SOURCE CODE 4](#_Toc96948684)

[III. HASIL PERCOBAAN DAN EVALUASI 9](#_Toc96948685)

[IV. ALAMAT KODE PROGRAM 12](#_Toc96948686)

1. ALGORITMA DIVIDE AND CONQUER

Algoritma *divide and conquer* dalam konteks ilmu komputer adalah salah satu strategi penyelesaian persoalan dengan cara membagi suatu persoalan dan menyelesaikannya. Terdapat tiga langkah utama yang digunakan dalam strategi algoritma ini, yaitu *divide, conquer,* dan *combine.* Pada langkah *divide*, persoalan dibagi menjadi upa-persoalan. Upa-persoalan adalah bagian dari persoalan utama yang juga memiliki karakteristik yang sama dengan persoalan induknya, namun memiliki ukuran yang lebih kecil. Langkah *conquer* yaitu langkah penyelesaian merupakan langkah yang dilakukan setelah persoalan telah dibagi menjadi ukuran terkecilnya. Langkah ini memberikan hasil yang kemudian akan digabung dengan hasil dari upa-persoalan lainnya dalam langkah *combine*.

Beberapa jenis persoalan yang umum diselesaikan dengan algoritma *divide and conquer* adalah persoalan yang memiliki ukuran *n*, seperti larik, matriks, pohon, eksponen, dan lain-lain. Penyelesaian persoalan menggunakan strategi ini umumnya diimplementasikan secara rekursif. Sementara itu, untuk suatu persoalan berukuran *n* yang diselesaikan menggunakan algoritma *divide and conquer* memiliki kompleksitas yang dapat dinyatakan menggunakan persamaan (1).

Salah satu persoalan yang dapat diselesaikan menggunakan algoritma ini adalah pencarian *convex hull* dari kumpulan titik-titik yang berada pada suatu bidang planar. Suatu himpunan titik dapat disebut *convex* jika untuk sembarang dua titik pada bidang, seluruh segmen garis yang berakhir di kedua titik tersebut berada pada himpunan. Pada himpunan yang terdiri hanya dari dua buah titik, maka *convex hull* adalah garis yang menghubungkan kedua titik tersebut. Jika terdapat titik-titik lainnya yang juga dilalui oleh garis yang sama, maka *convex hull* adalah garis yang menghubungkan titik-titik terjauh. Jika terdapat titik-titik lain yang tidak dilalui oleh garis yang sama, maka convex hull merupakan garis-garis yang membentuk sisi-sisi poligon.

Langkah-langkah umum pencarian *convex hull* dimulai dengan mencari garis yang menghubungkan dua titik terjauh dalam himpunan titik. Garis tersebut kemudian dijadikan acuan untuk membagi titik-titik dalam himpunan ke dalam dua himpunan baru berdasarkan daerahnya. Dimisalkan terdapat suatu garis yang dibentuk oleh titik () dan titik (). Untuk mengetahui di mana letak suatu titik (), dapat digunakan determinan yang dihitung dengan persamaan (2).

Posisi titik () dapat diketahui berdasarkan tanda positif dan negatif dari determinan.

Setelah himpunan awal terbagi menjadi dua himpunan baru, cari titik-titik terjauh dari garis pembagi dari masing-masing himpunan, sehingga terbentuk segitiga yang memiliki titik-titik sudut adalah titik-titik pembentuk garis pembagi dan titik terjauh. Abaikan seluruh titik yang berada di dalam segitiga yang terbentuk oleh titik-titik yang sudah dimiliki sebelumnya pada masing-masing daerah. Kemudian himpunan hasil pembagian dan garis baru yang dibentuk segitiga akan menjadi himpunan yang akan diproses pada iterasi berikutnya hingga rekursi dihentikan pada himpunan kosong. Pasangan titik pembentuk garis baru kemudian dikembalikan dan digabung menjadi penyelesaian dari persoalan utama.

1. IMPLEMENTASI

Program pengolahan *convex hull* ditulis dengan menggunakan bahasa python dalam bentuk kelas. Kelasyang digunakan diberi nama myConvexHull dan diimplementasikan pada dokumen berjudul myConvexHull.py.

STRUKTUR KELAS

Atribut dari kelas myConvexHull dijelaskan pada Tabel 2.1

Tabel 2.1. Daftar Atribut pada Kelas myConvexHull

|  |  |  |
| --- | --- | --- |
| **Nama Atribut** | **Tipe Data** | **Deskripsi** |
| points | *Array of array* | Points adalah larik dengan jenis elemen berupa titik (direpresentasikan sebagai *array of float*), menyatakan kumpulan titik yang akan dicari *convex hull*-nya. |
| vertices | *Array of array* | Vertices adalah larik dengan jenis elemen berupa titik yang menyatakan kumpulan titik pembentuk *convex hull*. |
| lines | *Array of array* | Lines adalah larik dengan jenis elemen berupa *tuple* dari dua titik pembentuk garis *convex hull*. |
| contained | *Set* | Set adalah himpunan bilangan bulat yang menyatakan indeks dari titik pada atribut points yang sudah berada di dalam *convex hull*. |

Seluruh metode yang digunakan dalam kelas myConvexHull dijabarkan pada Tabel 2.2.

Tabel 2.2. Daftar Metode pada Kelas myConvexHull

|  |  |
| --- | --- |
| **Nama Metode** | **Deskripsi** |
| \_\_init\_\_ | Konstruktor dari kelas. Menerima masukan data berupa data yang bertipe *numpy array*, dan berfungsi untuk mencari *convex hull* dan mengisi seluruh atribut kelas. |
| find\_outer\_points | Mencari dua titik dengan nilai absis terkecil dan terbesar (terluar) dari himpunan titik. Digunakan untuk melakukan inisialisasi garis pada algoritma *convex hull*. |
| determinant | Menerima masukan berupa dua titik pembentuk garis dan satu titik lainnya. Menghitung di mana posisi titik ketiga relatif terhadap garis. |
| get\_distance | Menerima masukan berupa dua titik pembentuk garis dan satu titik lainnya. Menghitung jarak dari titik ketiga ke garis. |
| find\_furthest\_idx | Menerima masukan berupa dua titik pembentuk garis. Mencari indeks dari titik pada atribut points yang memiliki jarak terjauh dari garis dibanding titik-titik lainnya. |
| is\_contained | Menerima masukan berupa tiga buah titik yang membentuk segitiga dan sebuah titik lainnya. Mencari tahu apakah titik tersebut berada di dalam atau luar segitiga. Perhitungan yang digunakan memanfaatkan koordinat barycentric. |
| update\_contained | Menerima masukan berupa tiga buah titik yang membentuk segitiga dan sebuah himpunan titik. Memasukkan indeks dari titik-titik pada himpunan yang berada di dalam segitiga ke atribut contained. |
| divide\_region | Menerima masukan berupa tiga buah titik dan suatu himpunan titik. Membagi himpunan ke dalam dua bagian berdasarkan posisinya relatif terhadap garis-garis yang dibentuk oleh tiga titik parameter. |
| update\_region | Menerima masukan berupa himpunan titik. Menghapus anggota himpunan apabila himpunan tersebut sudah berada pada atribut contained. |
| findConvexHull | Algoritma utama pencari *convex hull*. Menerima masukan berupa dua titik yang membentuk garis dan himpunan titik yang daerahnya terbagi menjadi dua oleh garis. Membagi daerah dan menyelesaikan permasalahan *convex hull* secara rekursif. |

SOURCE CODE

Realisasi dari kelas myConvexHull terdapat pada kode program yang ditulis dalam dokumen myConvexHull.py seperti sebagai berikut.

*# myConvexHull.py*

import math

**class** myConvexHull:

**def** \_\_init\_\_(self, data) :

        self.points = data.tolist() *# berisi point (point didefinisikan sebagai array of float dengan panjang 2 elemen)*

        self.vertices = [] *# berisi indeks dari point dalam points yang membentuk convex hull*

        self.lines = [] *# berisi point-point yang membentuk garis convex hull (didefinisikan sebagai array of integer dengan panjang 2 element)*

        self.contained = set() *# set yang berisi indeks dari point-point yang sudah berada di dalam area*

        outer = self.find\_outer\_points()

*# since outer points surely create the convex hull*

        self.vertices.append(self.points.index(outer[0]))

        self.vertices.append(self.points.index(outer[1]))

        self.lines.append([outer[0], outer[1]])

        self.contained.add(self.points.index(outer[0]))

        self.contained.add(self.points.index(outer[1]))

        init\_reg = self.points.copy()

        left\_point = outer[0]

        right\_point = outer[1]

        init\_reg.pop(init\_reg.index(left\_point))

        init\_reg.pop(init\_reg.index(right\_point))

        self.findConvexHull(left\_point, right\_point, init\_reg)

**def** find\_outer\_points (self) :

*# mencari point-point dengan nilai absis terendah dan tertinggi*

*# mengembalikan array dua elemen dalam bentuk [indeks point terkiri, indeks point terkanan]*

        res = [self.points[0],self.points[0]]

        min\_x = self.points[0][0]

        max\_x = self.points[0][0]

        for point in self.points :

            if point[0] < min\_x :

                min\_x = point[0]

                res[0] = point

            if point[0] > max\_x :

                max\_x = point[0]

                res[1] = point

        return res

**def** determinant(self, point\_line1, point\_line2, point) :

*# mencari nilai determinan untuk menentukan di daerah mana sebuah point berada*

*# point\_line1, point\_line2, point : point*

*# point\_line1 dan point\_line2 adalah ujung-ujung dari garis pembatas daerah*

        det = (point\_line1[0] \* point\_line2[1]) + (point[0] \* point\_line1[1]) + (point\_line2[0] \* point[1]) - (point[0] \* point\_line2[1]) - (point\_line2[0] \* point\_line1[1]) - (point\_line1[0] \* point[1])

        return det

**def** get\_distance (self, p1, p2, point) :

*# menghitung jarak point dari garis yang dibentuk p1 dan p2*

        a = p1[1] - p2[1]

        b = p2[0] - p1[0]

        c = (p1[0] \* p2[1]) - (p2[0] \* p1[1])

        denom = math.sqrt(a\*\*2 + b\*\*2)

        if (denom != 0) :

            return (abs(((a \* point[0]) + (b \* point[1]) + c)/denom))

        else :

            return -1

**def** find\_furthest\_idx(self, point\_line1, point\_line2, region) :

*# mengembalikan indeks dari titik terjauh dengan garis dan memasukkan indeks yang bukan merupakan paling jauh ke dalam contained*

*# dipastikan len(region) > 1*

        furthest = -1

        idx = -1

        for point in region :

            dist = self.get\_distance(point\_line1, point\_line2, point)

            if (dist != -1 and dist > furthest) :

                furthest = dist

                idx = self.points.index(point)

        return idx

**def** is\_contained(self, p1, p2, p3, point) :

*# menggunakan barycentric coordinate*

*# referensi : https://mathworld.wolfram.com/TriangleInterior.html#:~:text=The%20simplest%20way%20to%20determine,it%20lies%20outside%20the%20triangle.*

        denom = (p2[1] - p3[1])\*(p1[0] - p3[0]) + (p3[0] - p2[0])\*(p1[1] - p3[1])

        if (denom != 0) :

            a = ((p2[1] - p3[1])\*(point[0] - p3[0]) + (p3[0] - p2[0])\*(point[1] - p3[1])) / denom

            b = ((p3[1] - p1[1])\*(point[0] - p3[0]) + (p1[0] - p3[0])\*(point[1] - p3[1])) / denom

            c = 1 - a - b

            return ((0 <= a <= 1) and (0 <= b <= 1) and (0 <= c <= 1))

        else :

            return False

**def** update\_contained(self, p1, p2, p3, region) :

        for point in region :

            if (self.is\_contained(p1, p2, p3, point)) :

                self.contained.add(self.points.index(point))

**def** divide\_region(self, p1, p2, p3, region) :

*# cari garis yang tegak lurus dengan garis yang dibentuk oleh titik p1 dan p2 dan melewati titik p3*

*# bagi region berdasarkan tempat titik berada relatif terhadap garis tersebut*

        reg1 = [] *# berisi titik-titik yang berada di daerah yang sama dengan p1*

        reg2 = [] *# berisi titik-titik yang berada di daerah yang sama dengan p2*

*# cari titik-titik yang memiliki nilai x lebih kecil dan lebih besar*

*# x1 dan x2 tidak mungkin sama*

        if p1[0] < p2[0] :

            left = p1

            right = p2

        else :

            left = p2

            right = p1

*# cari persamaan garis*

        grad12 = (right[1] - left[1])/(right[0] - left[0])

        c12 = ((grad12 \* left[0]) \* -1) - left[1]

        if (grad12 != 0) :

            grad = -1/grad12

            c = ((grad \* p3[0]) \* -1) - p3[1]

            x = (c12 - c)/(grad - grad12)

            y = grad\*x + c

        else : *# p1 dan p2 memiliki nilai y yang sama*

            x = p3[0]

            y = p1[1]

*# sign1 dan sign2 menandakan tanda dari determinan, True untuk + dan False untuk -*

        sign1 = (self.determinant(p3, [x,y], p1) > 0)

        for point in region :

            sign = (self.determinant(p3, [x,y], point) > 0)

            if (sign==sign1) :

                reg1.append(point)

            else :

                reg2.append(point)

        return (reg1, reg2)

**def** update\_region (self, reg) :

*# menghapus elemen-elemen pada region yang sudah berada di dalam hull*

        new\_reg = []

        for point in reg :

*# abaikan point yang sudah berada di dalam bidang*

            if not (self.points.index(point) in self.contained) :

                new\_reg.append(point)

        reg = new\_reg

**def** findConvexHull (self, point\_line1, point\_line2, region) :

*# membentuk convex hull*

*# region : array of points*

*# point\_line1, point\_line2 : point*

        if (len(region) > 0) :

            regA = [] *# list dari point yang berada di daerah kiri/atas*

            regB = [] *# list dari point yang berada di daerah kanan/bawah*

*# Divide region*

            for point in region :

                det = self.determinant(point\_line1, point\_line2, point)

                if (abs(det) < 1e-12) :

*# dianggap ada di garis*

                    self.contained.add(self.points.index(point))

                elif (det > 0) :

                    regA.append(point)

                else :

                    regB.append(point)

            a\_is\_not\_contained = (len(regA) > 0 and not self.points.index(regA[0]) in self.contained)

            b\_is\_not\_contained = (len(regB) > 0 and not self.points.index(regB[0]) in self.contained)

            new\_triangleA\_created = False

            new\_triangleB\_created = False

*# if any of the region is contained already, just skip*

*# find furthest point in the not-contained area*

*# then, update the vertices and lines*

            if (a\_is\_not\_contained) :

                furthestA = self.find\_furthest\_idx(point\_line1, point\_line2, regA)

                self.vertices.append(furthestA)

                self.lines.append([point\_line1, self.points[furthestA]])

                self.lines.append([point\_line2, self.points[furthestA]])

*# setelah terbentuk segitiga dalam region, update dulu point-point mana aja yang masuk ke segitiga*

                self.update\_contained(point\_line1, point\_line2, self.points[furthestA], regA)

                self.update\_region(regA)

                new\_triangleA\_created = (furthestA != -1)

*# bagi region menjadi yang lebih dekat dengan masing-masing*

                regs = self.divide\_region(point\_line1, point\_line2, self.points[furthestA], regA)

                self.findConvexHull(point\_line1, self.points[furthestA], regs[0])

                self.findConvexHull(point\_line2, self.points[furthestA], regs[1])

            if (b\_is\_not\_contained) :

                furthestB = self.find\_furthest\_idx(point\_line1, point\_line2, regB)

                self.vertices.append(furthestB)

                self.lines.append([point\_line1, self.points[furthestB]])

                self.lines.append([point\_line2, self.points[furthestB]])

                self.update\_contained(point\_line1, point\_line2, self.points[furthestB], regB)

                self.update\_region(regB)

                new\_triangleB\_created = (furthestB != -1)

                regs = self.divide\_region(point\_line1, point\_line2, self.points[furthestB], regB)

                self.findConvexHull(point\_line1, self.points[furthestB], regs[0])

                self.findConvexHull(point\_line2, self.points[furthestB], regs[1])

*# delete the line created by point\_line1 and point\_line2*

            if(new\_triangleA\_created or new\_triangleB\_created) :

                self.lines.pop(self.lines.index([point\_line1, point\_line2]))

Adapun contoh penggunaan kelas yang kemudian akan digunakan untuk pengujian program juga terdapat pada dokumen testMyConvexHull.ipynb.

1. HASIL PERCOBAAN DAN EVALUASI

Untuk menguji program yang dibuat, digunakan tiga jenis dataset percobaan yang diambil dari pustaka sklearn. Dataset yang digunakan adalah dataset iris, wine, dan breast cancer. Nama dataset, pasangan atribut yang digunakan serta hasil percobaan terdapat pada Tabel 3.1.

Tabel 3.1. Hasil Percobaan

|  |  |  |
| --- | --- | --- |
| Nama Dataset | Pasangan Atribut | Hasil Convex Hull |
| iris | Petal-length, petal-width |  |
| iris | Sepal-length, sepal-width |  |
| wine | Alcohol, Malic Acid |  |
| wine | Color Intensity, Hue |  |
| breast\_cancer | Radius, Smoothness |  |
| breast\_cancer | Symmetry, Fractal Dimension |  |

Hasil percobaan dirangkum dalam evaluasi yang terdapat pada Tabel 3.2.

Tabel 3.2. Evaluasi Program

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

1. ALAMAT KODE PROGRAM

Kode implementasi kelas myConvexHull, panduan penggunaan pustaka, program untuk menguji kelas myConvexHull, serta program pengujian kelas ConvexHull dari scipy.spatial terdapat pada repository GitHub dengan alamat :

<https://github.com/chryes220/Tucil2-Stima.git>