Laporan Tugas Kecil 2 Strategi Algoritma IF2211 Semester II Tahun 2021/2022

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

Logo

Description automatically generated

Nayotama Pradipta - 13520089

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

1. Pendahuluan
2. Divide & Conquer

Divide & conquer merupakan algoritma rekursif yang digunakan untuk memecah suatu persolan menjadi sub permasalahan yang lebih kecil dengan tipe yang sama. Pemecahan dilakukan hingga sub masalah menjadi cukup mudah untuk diselesaikan secara langsung. Algoritma ini menjadi basis solusi dari berbagai macam permasalahan termasuk permasalahan pengurutan atau sorting. Secara garis besar, algoritma ini terdiri atas tiga pokok konsep:

1. Divide: Membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama)
2. Conquer: Menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar)
3. Combine: Menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula

Diagram, engineering drawing

Description automatically generated

Gambar 1.1 Diagram Penjelasan Algoritma Divide & Conquer. Diambil dari [1]

1. Convex Hull

Convex Hull merupakan himpanan titik S pada bidang planar yang 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. Untuk dua titik, maka convex hull berupa garis yang menghubungkan 2 titik tersebut. Untuk tiga titik yang terletak pada satu garis, maka convex hull adalah sebuah garis yang menghubungkan dua titik terjauh.

Diagram

Description automatically generated with medium confidence

Gambar 1.2 Contoh Convex Hull untuk 8 titik. Diambil dari [2]

1. Langkah-langkah Algoritma Divide & Conquer

Algoritma Divide & Conquer pada permasalahan convex hull diterapkan dengan membagi titik-titik menjadi dua daerah secara rekursif. Pertama-tama, titik yang paling kiri dan paling kanan dijadikan sebagai patokan untuk membuat garis. Kemudian titik yang berada di atas garis tersebut dikelompokkan menjadi satu, dan titik yang berada di bawah garis tersebut juga dikelompokkan mnejadi satu. Setelah itu untuk masing-masing daerah, diambil titik yang memiliki jarak paling jauh dengan garis. Jika terdapat lebih dari satu titik yang memiliki jarak terjauh, maka akan diambil titik yang paling besar membentuk sudut antara titik paling kiri dan paling kanan. Titik tersebut akan menjadi acuan lanjutan untuk mengelompokkan titik-titik lain yang tidak di dalam daerah yang sudah ditinjau. Cara ini akan dilakukan terus menerus hingga titik yang tersisa tinggal satu. Ketika titik yang tersisa hanya satu, maka titik tersebut akan masuk ke dalam komponen Convex Hull. Setelah semua titik tersimpan, maka langkah selanjutnya hanyalah menyambungkan antar titik hingga terbentuk Convex Hull sejati.

1. Source Program

Program dibuat dengan menggunakan bahasa Python dan dibagi menjadi dua file: myConvexhull.py dan main.py. File myConvexhull.py berisi semua fungsi untuk melakukan komputasi Convex Hull terhadap titik-titik. File main merupakan implementasi fungsi-fungsi yang ada. Berikut adalah kode program:

File myConvexhull.py

|  |
| --- |
| import numpy as np  import math  from collections import Iterable  *def* determinantThreePoints(*p1*, *p2*, *pi*):      # Mengembalikan determinan dari tiga titik koordinat      return (*p1*[0] \* *p2*[1] + *pi*[0] \* *p1*[1] + *p2*[0] \* *pi*[1] - *pi*[0] \* *p2*[1] - *p2*[0] \* *p1*[1] - *p1*[0] \* *pi*[1])  *def* up(*p1*, *p2*, *pi*):      return (determinantThreePoints(*p1*, *p2*, *pi*) > 0)  *def* down(*p1*, *p2*, *pi*):      return (determinantThreePoints(*p1*, *p2*, *pi*) < 0)  *def* ignoreCoordinate(*p1*, *p2*, *pi*):      return ((*p1*[0] \* (*p2*[1] - *pi*[1]) + *p2*[0] \* (*pi*[1] - *p1*[1]) + *pi*[0] \* (*p1*[1] - *p2*[1])) == 0)  *def* getExtremeLeftPoint(*bucket*):      minXval = min(*bucket*[:, 0])      minXidx = np.where(*bucket*[:,0]==minXval)      return *bucket*[minXidx]  *def* getExtremeRightPoint(*bucket*):      maxXval = max(*bucket*[:, 0])      maxXidx = np.where(*bucket*[:,0] == maxXval)      return *bucket*[maxXidx]  *def* getOuterUpPoints(*p1*, *p2*, *bucket*):      arrOfOuterPoints = np.zeros((len(*bucket*),2))      k = 0      for i in range(len(*bucket*)):          if (not ignoreCoordinate(*p1*, *p2*, *bucket*[i])):              if up(*p1*, *p2*, *bucket*[i]):                  arrOfOuterPoints[k] = *bucket*[i]                  k += 1      arrOfOuterPoints = arrOfOuterPoints[~np.all(arrOfOuterPoints == 0, *axis*=1)]      return arrOfOuterPoints  *def* getOuterDownPoints(*p1*, *p2*, *bucket*):      arrOfOuterPoints = np.zeros((len(*bucket*), 2))      k = 0      for i in range(len(*bucket*)):          if (not ignoreCoordinate(*p1*, *p2*, *bucket*[i])):              if (down(*p1*, *p2*, *bucket*[i])):                  arrOfOuterPoints[k] = *bucket*[i]                  k += 1      arrOfOuterPoints = arrOfOuterPoints[~np.all(arrOfOuterPoints == 0, *axis*=1)]      return arrOfOuterPoints  *def* getDistance(*p1*, *p2*, *p3*):      return np.abs(np.cross(*p2*-*p1*, *p3*-*p1*)/np.linalg.norm(*p2*-*p1*))  *def* getAngle(*a*, *b*, *c*):      ang = math.degrees(math.atan2(*c*[1]-*b*[1], *c*[0]-*b*[0]) - math.atan2(*a*[1]-*b*[1], *a*[0]-*b*[0]))      return ang + 360 if ang < 0 else ang    *def* getMaxPoint(*p1*, *p2*, *point*):      # Inisialisasi array untuk menyimpan jarak semua titik      arrOfDistance = [0 for i in range(len(*point*))]      for i in range(len(*point*)):          arrOfDistance[i] = getDistance(*p1*, *p2*, *point*[i])      maxDistance = max(arrOfDistance)      # Cek apakah titik dengan jarak maksimum ada lebih dari satu      arrOfMax = [i for i, value in enumerate(arrOfDistance) if value == maxDistance]      if (len(arrOfMax) > 1):          angle = [0 for i in range(len(arrOfMax))]          for i in range(len(arrOfMax)):              angle[i] = getAngle(*point*[arrOfMax[i]], *p1*, *p2*)          maxIdx = angle.index(max(angle))          return *point*[arrOfMax[maxIdx]]      else:          return *point*[arrOfMax[0]]  *def* convexHull(*p1*, *p2*, *arrOfPoints*):        if (len(*arrOfPoints*) <= 1):          if (len(*arrOfPoints*) == 1):              return *arrOfPoints*          else:              return 0      else:          p3 = getMaxPoint(*p1*,*p2*, *arrOfPoints*)          a = getOuterUpPoints(*p1*, p3, *arrOfPoints*)          b = getOuterUpPoints(p3, *p2*, *arrOfPoints*)          return convexHull(*p1*, p3, a), p3, convexHull(p3, *p1*, b)    *def* convertToLinear(*list*):      for item in *list*:          if isinstance(item, Iterable) and not isinstance(item, str):              for x in convertToLinear(item):                  yield x          else:              yield item  *def* myConvexHull(*bucket*):      arrOfPoints = []      p1 = getExtremeLeftPoint(*bucket*)      p2 = getExtremeRightPoint(*bucket*)      p1new = p1.flatten()      p2new = p2.flatten()      upPoints = getOuterUpPoints(p1new,p2new, *bucket*)      downPoints = getOuterDownPoints(p1new, p2new, *bucket*)      arrOfPoints.append(p1new)      s1 = convexHull(p1new, p2new, upPoints)      s2 = convexHull(p2new, p1new, downPoints)      arrOfPoints.append(s1)      arrOfPoints.append(p2new)      arrOfPoints.append(s2)      temp = []      for i in range(len(arrOfPoints)):          temp.append(arrOfPoints[i])      newtemp = convertToLinear(temp)      newtemp = [i for i in newtemp if i != 0]      coordinate = []      k = 0      for i in range(len(newtemp)//2):          coordinate.append([])          for j in range(2):              coordinate[i].append(newtemp[k])              k+=1      return coordinate |

File main.py

|  |
| --- |
| from myConvexhull import getMaxPoint  from myConvexhull import myConvexHull  import myConvexhull  import pandas as pd  import matplotlib.pyplot as plt  import numpy as np  from sklearn import datasets  data = datasets.load\_iris()  df = pd.DataFrame(data.data, *columns*=data.feature\_names)  df['Target'] = pd.DataFrame(data.target)  #visualisasi hasil ConvexHull  from scipy.spatial import ConvexHull  plt.figure(*figsize* = (10, 6))  colors = ['b','r','g']  plt.title('Petal Width vs Petal 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.myConvexHull(bucket)      plt.scatter(bucket[:, 0], bucket[:, 1], *label*=data.target\_names[i])      xval = [0 for i in range(len(hull)+1)]      yval = [0 for i in range(len(hull)+1)]      if (i >= 1):          hull.pop(len(hull)-1)          xval.pop(1)          yval.pop(1)      for j in range(len(hull)):          xval[j] = hull[j][0]          yval[j] = hull[j][1]      xval[len(hull)] = xval[0]      yval[len(hull)] = yval[0]      plt.plot(xval, yval, colors[i])  plt.show() |

1. Screenshot Input & Output
2. Input & Output Iris Dataset Sepal

Table

Description automatically generated with low confidence

Chart, scatter chart

Description automatically generated

1. Input & Output Iris Dataset Petal

Table

Description automatically generated with low confidence

Pada dataset petal, program saya mengalami *ValueError: operands could not be broadcast together with shapes (4,) (2,)* yang berhubungan dengan fungsi getDistance. Oleh karena itu, saya mencoba menerapkan getDistance menggunakan pendekatan lain seperti sebagai berikut:

|  |
| --- |
| *def* getDistance(*p1*, *p2*, *p3*):      a = *p2*[1] - *p1*[1]      b = *p1*[0] - *p2*[0]      c = a\*(*p1*[0]) + b\*(*p2*[1])      d = abs((a \* *p3*[0] + b \* *p3*[1] + c)) / (math.sqrt(a\*a + b\*b))        return d |

Program dapat menghasilkan jawaban, akan tetapi jawabannya tidak sesuai dengan hasil yang diinginkan:

|  |
| --- |
| Chart  Description automatically generated |

1. Input & Output Breast Cancer Dataset

Table

Description automatically generated

Chart, scatter chart

Description automatically generated

1. Checklist Fitur Program

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Pustaka myConvexHull berhasil dibuat dan tidak ada kesalahan | √ | √ (Ada kesalahan kecil tapi fatal untuk beberapa input) |
| 2. Program berhasil *running* | √ |  |
| 3. Pustaka *myConvexHull* dapat digunakan untuk menampilkan *convex hull* setiap label dengan warna yang berbeda | √ |  |
| 4. **Bonus**: program dapat menerima input dan menuliskan output untuk dataset lainnya | √ |  |

1. Link Drive Kode Program

|  |
| --- |
| <https://github.com/NayotamaPradipta/Tucil2STIMA> |

1. Kesimpulan

Setelah saya coba menggunakan beberapa dataset yang ada, saya sadar bahwa program saya memiliki kekurangan deteksi convex hull khususnya tepat di pojok kiri dan kanan. Sangat disayangkan waktunya sudah terbatas jadi saya akan mengumpulkan yang versi ini saja. Secara keseluruhan tugas kecil ini sangatlah membuka wawasan saya mengenai algoritma divide and conquer, dan saya menjadi lebih kreatif dalam menyelesaikan berbagai permasalahan. Penyelesaian algoritma ini dengan konsep rekursif juga menjadi tantangan yang sangat baik bagi saya, karena saya cukup lemah di bidang tersebut.

Daftar Pustaka

[1] Munir, Rinaldi. 2021. *Algoritma Divide and Conquer (Bagian 1)*. Diakses di: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf> (Diakses pada 26 Februari 2022)

[2] Munir, Rinaldi. 2021. *Algoritma Divide and Conquer (Bagian 4)*. Diakses di : <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Divide-and-Conquer-(2022)-Bagian4.pdf> (Diakses pada 28 Februari 2022)